ISM6218 Advanced Database Management
FINAL PROJECT

Praneeth venkata sai Eluri

Comprehensive Analysis and Application of IPL Data: A Portfolio Project in Database Design and Implementation

Database Design:

The project entails creating a database system of Indian Premier League (IPL). It includes detailed records for each ball in every match, player performances, team details, and seasonal information. The "BALL_BY_BALL" entity captures specifics of each delivery, like which batsman faced it and the runs scored. "MATCH" details the match specifics, "PLAYER" contains details of each player, "PLAYER_MATCH" links players with specific matches, and "TEAM" and "SEASON" entities hold information about the IPL teams and the tournament years, respectively. This setup would be ideal for in-depth analysis of matches and player statistics over various IPL seasons that enables cricket fans as well as those in the sports industry such as journalists, bloggers, writers, etc, to be able to accurately and effectively lookup some of their favourite players and teams, the matches they were involved in, and other aspects on those leagues that they were involved in, such as the category under which it falls, winners, scores, etc.

To effectively balance overhead costs as well as retrieval times for reads from the database, appropriate indexing must be allocated to either a single or a combination of columns, while maintaining integrity and accuracy of the data retrieved.

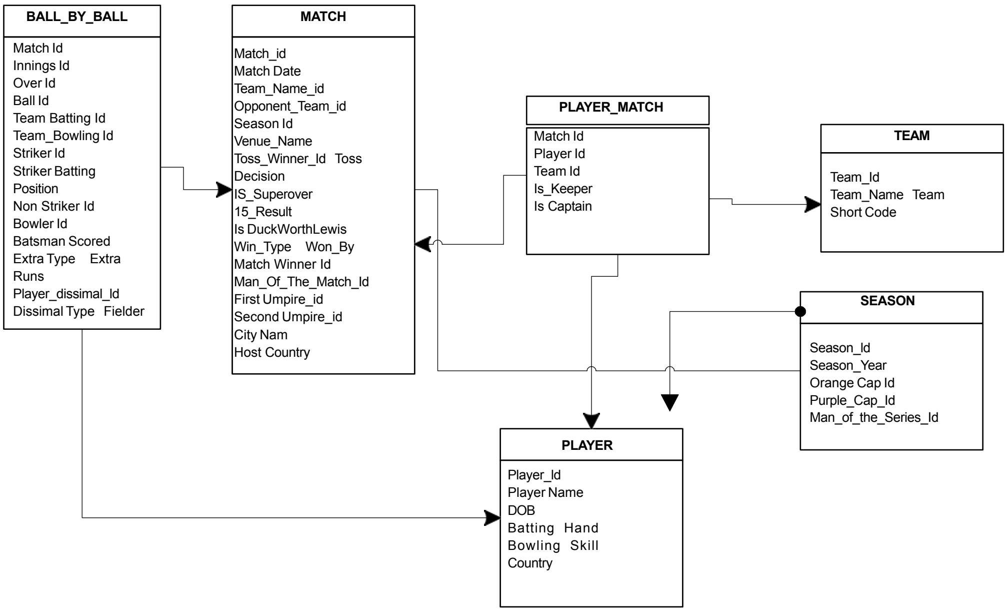
A brief summary of the involved tables is as follows;

TABLE NAME	DESCRIPTION	PRIMARY KEY

MATCH	Contains records of various aspects about the	Composite primary key of Match_id,Match_Winner_id

	match such as the Match Date,Venue,Winner,Team Ids, etc. This table has the most columns due to the several attributes a movie can have.	and Toss_Winner_Id
BALL_BY_BALL	This table accurately depicts the roles different personnel have on match, contributing in their respective positions	Composite primary key of Match_Id, innings_Id and Over Id
TEAM	This table expounds on teams those involved in the season, from their bio data.	Team_id
SEASON	This table records the details of which season, such as season_year, season_id, etc.	Season_Id
PLAYER	An associative entity table that enables mapping of the many-to-many relationships between Player_Match,season.	Player_Id
PLAYER_MATCH	This table records the details of the player in the particular match, such as match_id, player_id,team, etc.	Composite primary key of Match_Id and Player_Id

ER Diagram

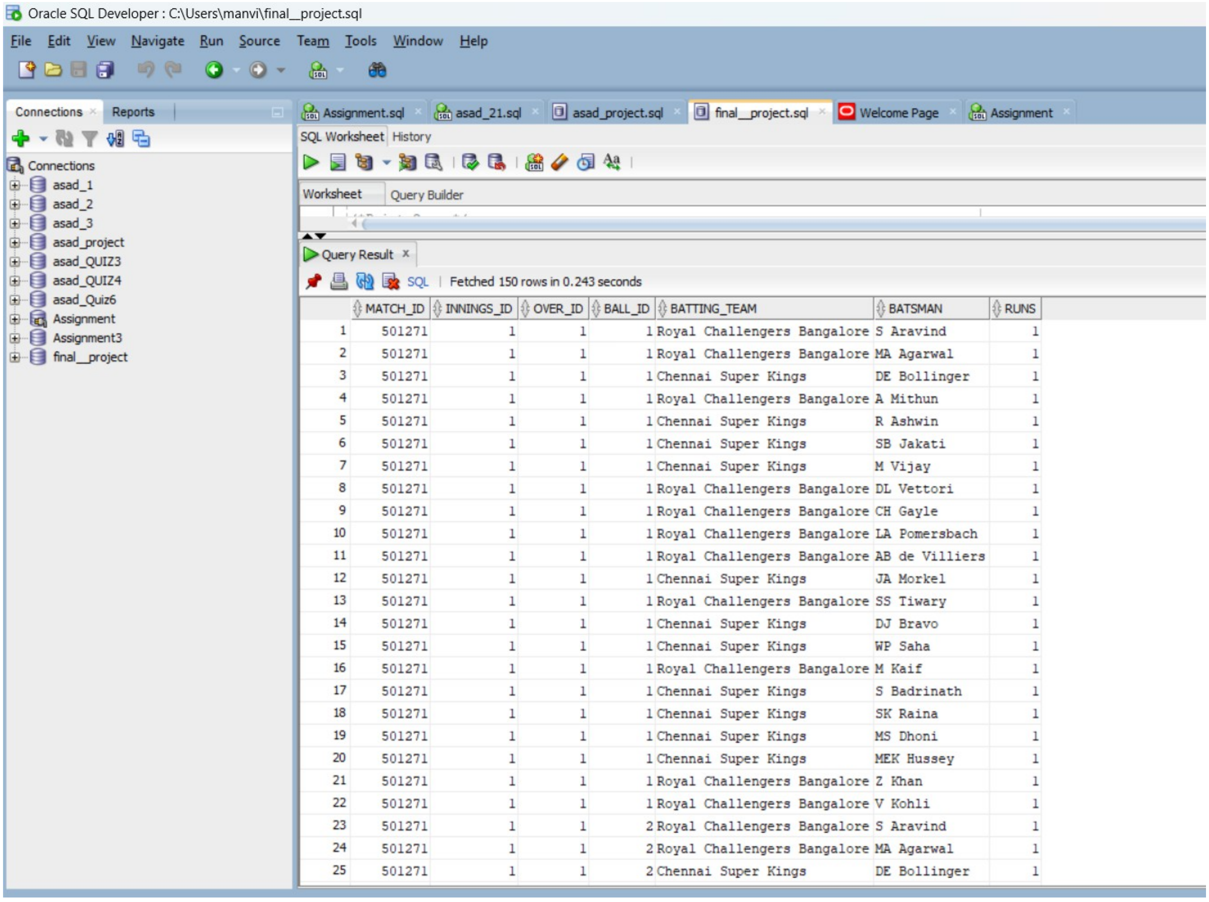


Query Writing

/*Point Query*/

1) Finding a specific match details like overs, team, batsmen and runs scored using match_id.

```
SELECT MATCH_ID, INNINGS_ID, OVER_ID, BALL_ID,
      TEAM_NAME AS BATTING_TEAM,
      PLAYER_NAME AS BATSMAN,
      BATSMAN_SCORED AS RUNS
FROM BALL_BY_BALL
INNER JOIN MATCH USING (MATCH_ID)
INNER JOIN PLAYER_MATCH USING (MATCH_ID)
INNER JOIN PLAYER USING (PLAYER_ID)
INNER JOIN TEAM USING (TEAM_ID)
WHERE MATCH_ID = 501271  ;
```



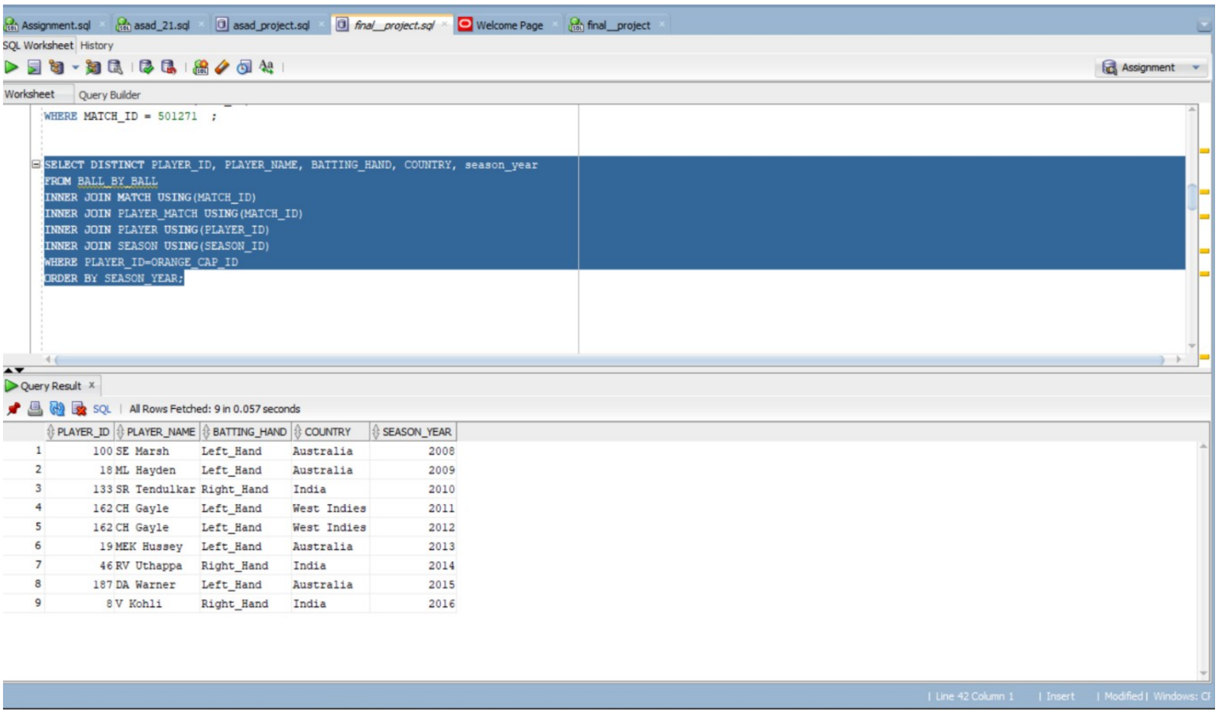
	MATCH_ID	INNINGS_ID	OVER_ID	BALL_ID	BATTING_TEAM	BATSMAN	RUNS
1	501271	1	1	1	Royal Challengers Bangalore	S Aravind	1
2	501271	1	1	1	Royal Challengers Bangalore	MA Agarwal	1
3	501271	1	1	1	Chennai Super Kings	DE Bollinger	1
4	501271	1	1	1	Royal Challengers Bangalore	A Mithun	1
5	501271	1	1	1	Chennai Super Kings	R Ashwin	1
6	501271	1	1	1	Chennai Super Kings	SB Jakati	1
7	501271	1	1	1	Chennai Super Kings	M Vijay	1
8	501271	1	1	1	Royal Challengers Bangalore	DL Vettori	1
9	501271	1	1	1	Royal Challengers Bangalore	CH Gayle	1
10	501271	1	1	1	Royal Challengers Bangalore	LA Pomersbach	1
11	501271	1	1	1	Royal Challengers Bangalore	AB de Villiers	1
12	501271	1	1	1	Chennai Super Kings	JA Morkel	1
13	501271	1	1	1	Royal Challengers Bangalore	SS Tiwary	1
14	501271	1	1	1	Chennai Super Kings	DJ Bravo	1
15	501271	1	1	1	Chennai Super Kings	WP Saha	1
16	501271	1	1	1	Royal Challengers Bangalore	M Kaif	1
17	501271	1	1	1	Chennai Super Kings	S Badrinath	1
18	501271	1	1	1	Chennai Super Kings	SK Raina	1
19	501271	1	1	1	Chennai Super Kings	MS Dhoni	1
20	501271	1	1	1	Chennai Super Kings	MEK Hussey	1
21	501271	1	1	1	Royal Challengers Bangalore	Z Khan	1
22	501271	1	1	1	Royal Challengers Bangalore	V Kohli	1
23	501271	1	1	2	Royal Challengers Bangalore	S Aravind	1
24	501271	1	1	2	Royal Challengers Bangalore	MA Agarwal	1
25	501271	1	1	2	Chennai Super Kings	DE Bollinger	1

2) Finding the top-scoring batsman for each season

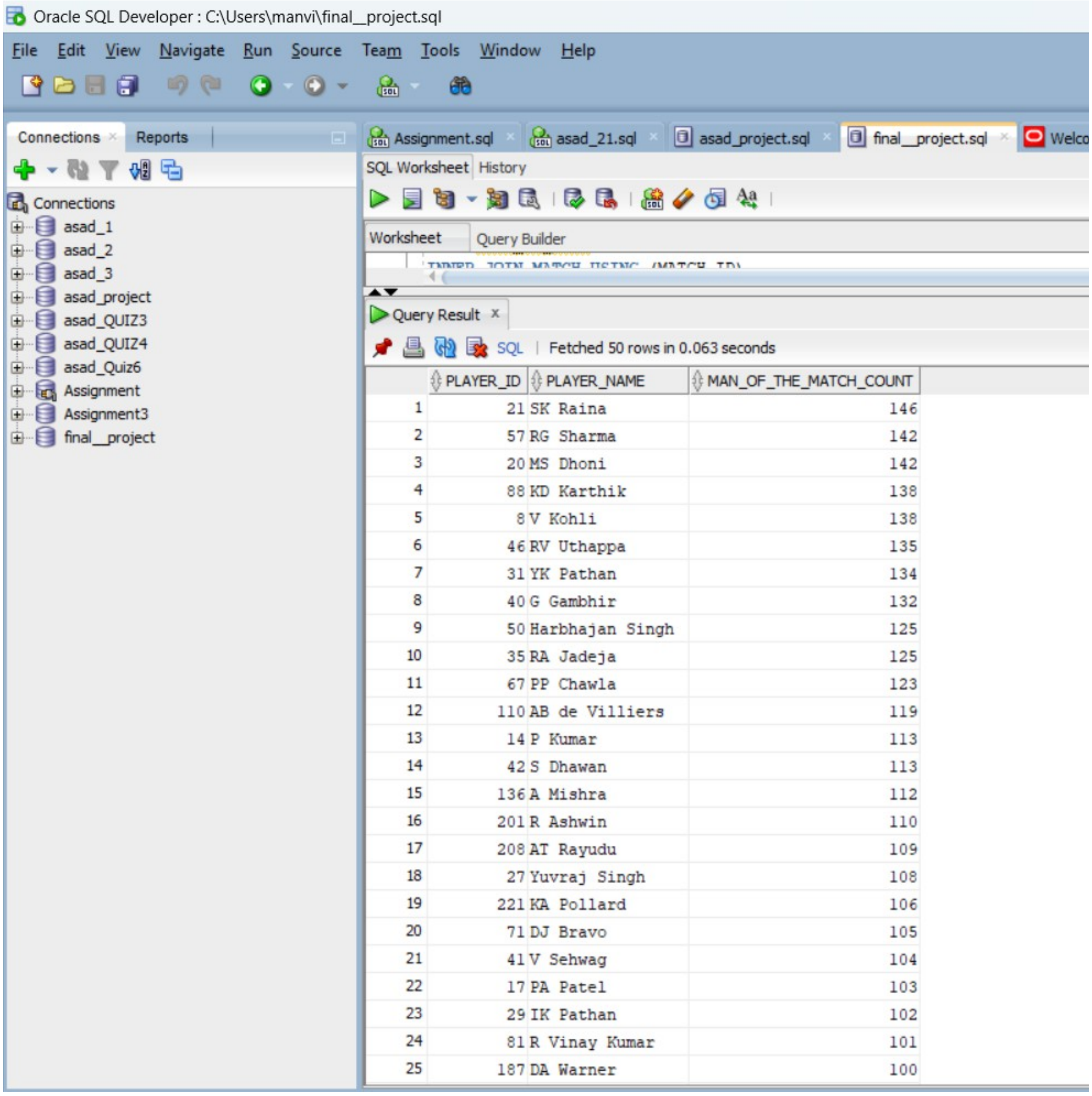
```
SELECT DISTINCT PLAYER_ID, PLAYER_NAME, BATTING_HAND,
COUNTRY, season_year
FROM BALL_BY_BALL
INNER JOIN MATCH USING(MATCH_ID)
INNER JOIN PLAYER_MATCH USING(MATCH_ID)
INNER JOIN PLAYER USING(PLAYER_ID)
INNER JOIN SEASON USING(SEASON_ID)
WHERE PLAYER_ID=ORANGE_CAP_ID
ORDER BY SEASON_YEAR;
```

/*Scan Query*/

1) Finding the Players with most man of the match awards




```
SELECT PLAYER_ID, PLAYER_NAME, COUNT(*) AS
MAN_OF_THE_MATCH_COUNT
FROM MATCH
INNER JOIN PLAYER_MATCH USING (MATCH_ID)
INNER JOIN PLAYER USING (PLAYER_ID)
GROUP BY PLAYER_ID, PLAYER_NAME
HAVING COUNT(*) > 10
ORDER BY MAN_OF_THE_MATCH_COUNT DESC;
```



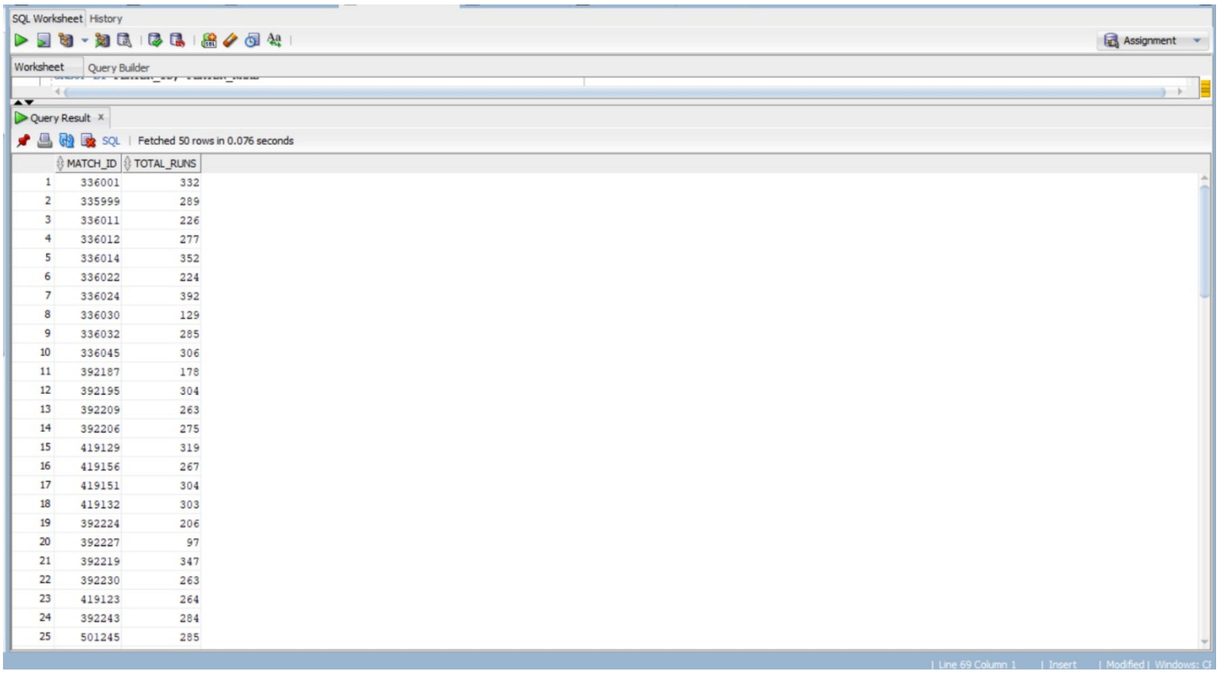
	PLAYER_ID	PLAYER_NAME	MAN_OF_THE_MATCH_COUNT
1	21	SK Raina	146
2	57	RG Sharma	142
3	20	MS Dhoni	142
4	88	KD Karthik	138
5	8	V Kohli	138
6	46	RV Uthappa	135
7	31	YK Pathan	134
8	40	G Gambhir	132
9	50	Harbhajan Singh	125
10	35	RA Jadeja	125
11	67	PP Chawla	123
12	110	AB de Villiers	119
13	14	P Kumar	113
14	42	S Dhawan	113
15	136	A Mishra	112
16	201	R Ashwin	110
17	208	AT Rayudu	109
18	27	Yuvraj Singh	108
19	221	KA Pollard	106
20	71	DJ Bravo	105
21	41	V Sehwag	104
22	17	PA Patel	103
23	29	IK Pathan	102
24	81	R Vinay Kumar	101
25	187	DA Warner	100

2) List all matches with total runs scored:

```
SELECT Match_Id, SUM(BATSMAN_Scored) AS Total_Runs
FROM Match
INNER JOIN Ball_by_Ball USING(MATCH_ID)
GROUP BY Match_Id;
```

/*RANGE QUERIES*/

1) Finding matches which was won by 50 runs or 3 wickets

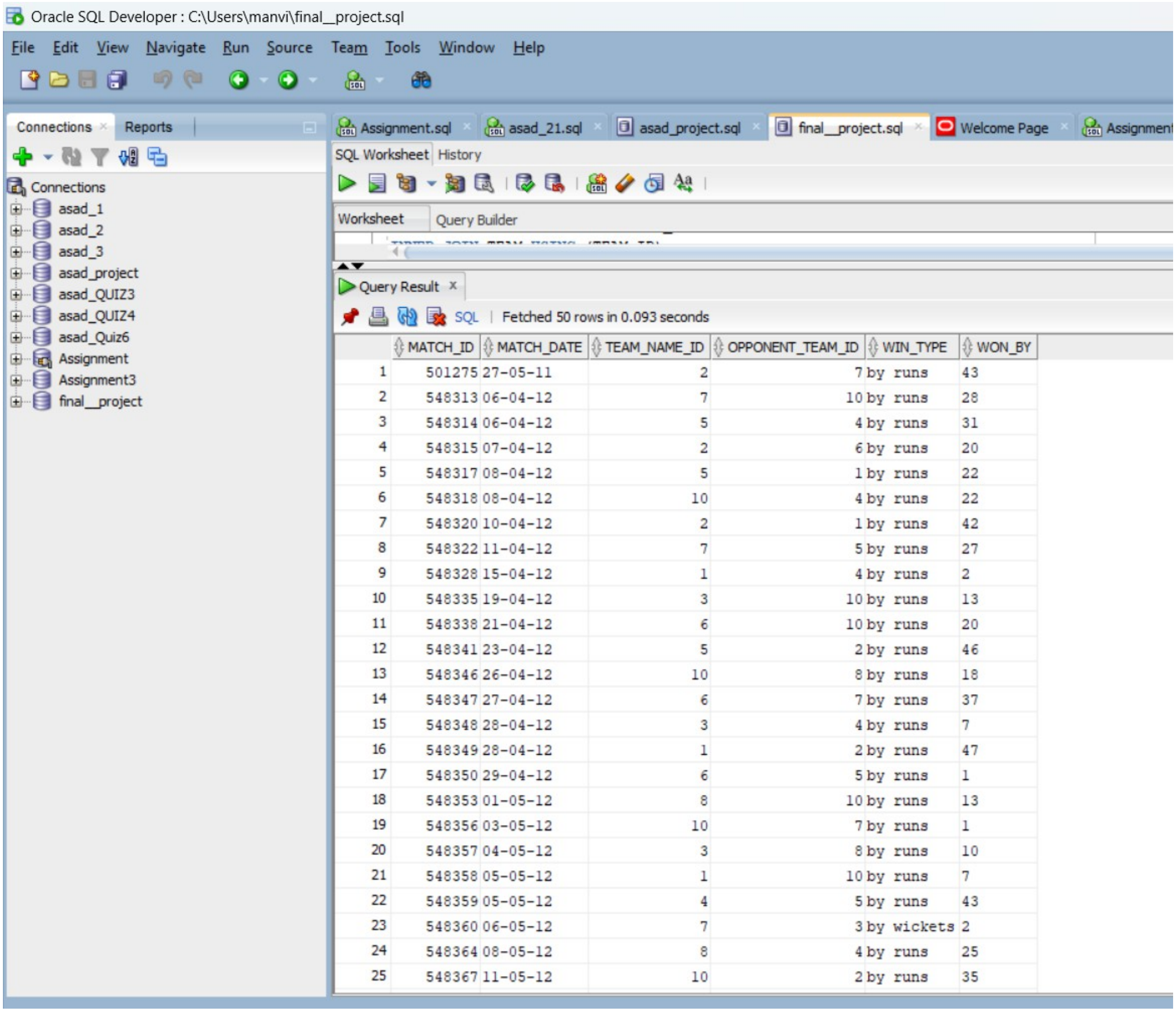


Query Result

Fetched 50 rows in 0.076 seconds

	MATCH_ID	TOTAL_RUNS
1	336001	332
2	335999	289
3	336011	226
4	336012	277
5	336014	352
6	336022	224
7	336024	392
8	336030	129
9	336032	285
10	336045	306
11	392187	178
12	392195	304
13	392209	263
14	392206	275
15	419129	319
16	419156	267
17	419151	304
18	419132	303
19	392224	206
20	392227	97
21	392219	347
22	392230	263
23	419123	264
24	392243	284
25	501245	285

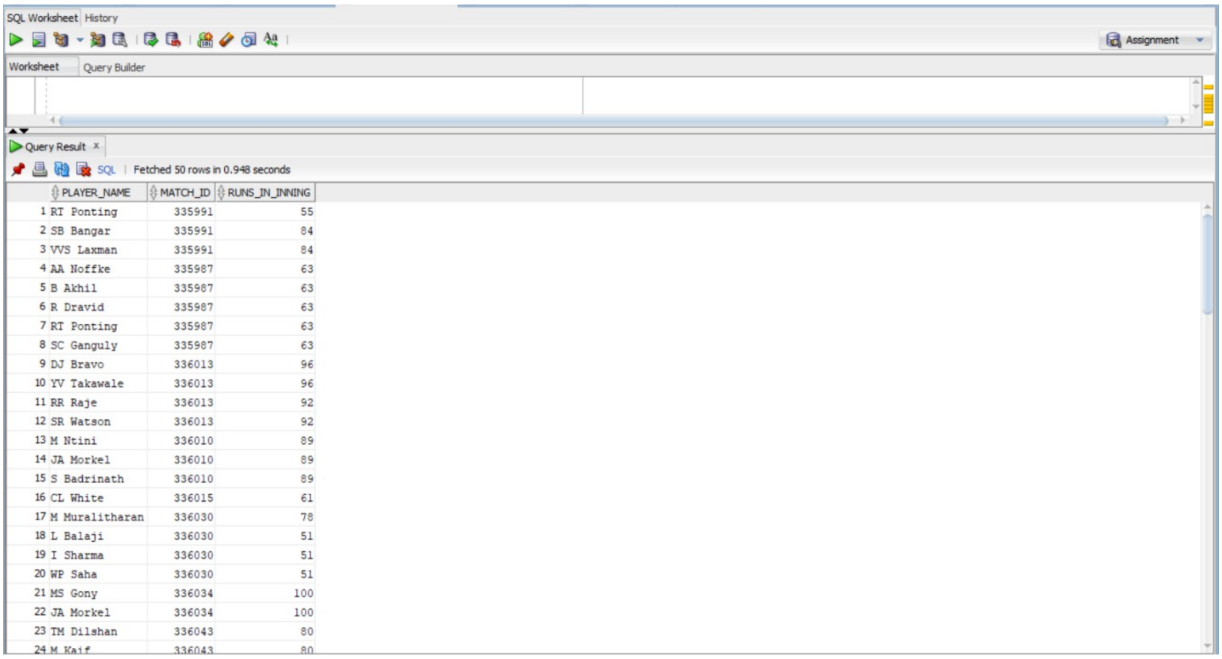
```
SELECT MATCH_ID, MATCH_DATE, TEAM_NAME_ID,
OPPONENT_TEAM_ID, WIN_TYPE, WON_BY
FROM MATCH
WHERE (WIN_TYPE = 'by runs' AND WON_BY < 50)
   OR (WIN_TYPE = 'by wickets' AND WON_BY < 3);
```



	MATCH_ID	MATCH_DATE	TEAM_NAME_ID	OPPONENT_TEAM_ID	WIN_TYPE	WON_BY
1	501275	27-05-11	2	7	by runs	43
2	548313	06-04-12	7	10	by runs	28
3	548314	06-04-12	5	4	by runs	31
4	548315	07-04-12	2	6	by runs	20
5	548317	08-04-12	5	1	by runs	22
6	548318	08-04-12	10	4	by runs	22
7	548320	10-04-12	2	1	by runs	42
8	548322	11-04-12	7	5	by runs	27
9	548328	15-04-12	1	4	by runs	2
10	548335	19-04-12	3	10	by runs	13
11	548338	21-04-12	6	10	by runs	20
12	548341	23-04-12	5	2	by runs	46
13	548346	26-04-12	10	8	by runs	18
14	548347	27-04-12	6	7	by runs	37
15	548348	28-04-12	3	4	by runs	7
16	548349	28-04-12	1	2	by runs	47
17	548350	29-04-12	6	5	by runs	1
18	548353	01-05-12	8	10	by runs	13
19	548356	03-05-12	10	7	by runs	1
20	548357	04-05-12	3	8	by runs	10
21	548358	05-05-12	1	10	by runs	7
22	548359	05-05-12	4	5	by runs	43
23	548360	06-05-12	7	3	by wickets	2
24	548364	08-05-12	8	4	by runs	25
25	548367	11-05-12	10	2	by runs	35

2) Find all players who have scored between 50 and 100 runs in a single-inning

```
SELECT Player_Name, Match_Id, SUM(BATSMAN_Scored) AS
Runs_In_Inning
FROM BALL_BY_BALL
INNER JOIN MATCH USING(MATCH_ID)
INNER JOIN PLAYER_MATCH USING(MATCH_ID)
INNER JOIN PLAYER USING(PLAYER_ID)
GROUP BY Player_Name, Match_Id, Innings_Id
HAVING SUM(Batsman_Scored) BETWEEN 50 AND 100;
```



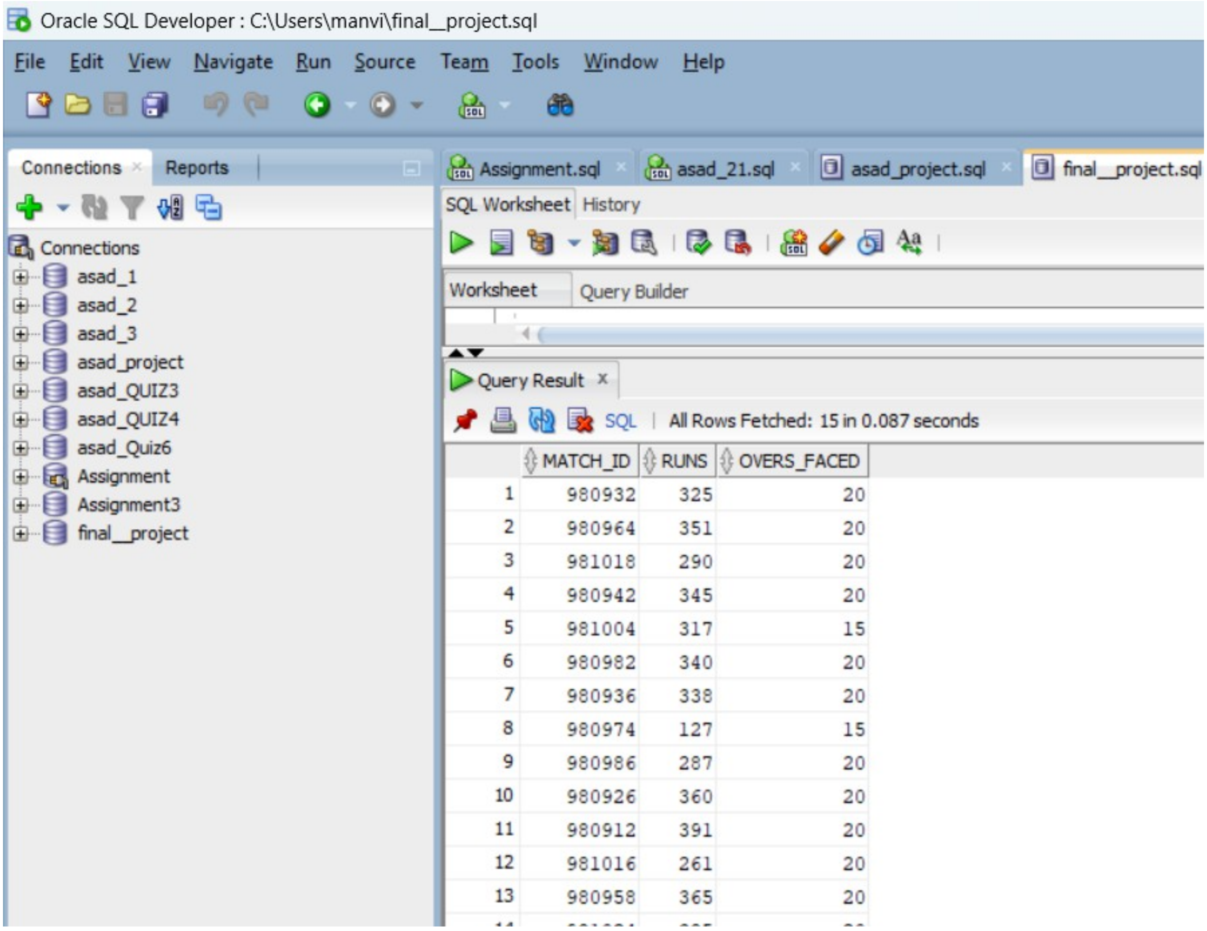
Query Result

Fetched 50 rows in 0.948 seconds

	PLAYER_NAME	MATCH_ID	RUNS_IN_INNING
1	RT Ponting	335991	55
2	SB Bangar	335991	84
3	VVS Laxman	335991	84
4	AA Noffke	335987	63
5	B Akhil	335987	63
6	R Dravid	335987	63
7	RT Ponting	335987	63
8	SC Ganguly	335987	63
9	DJ Bravo	336013	96
10	YV Takawale	336013	96
11	RR Raje	336013	92
12	SR Watson	336013	92
13	M Ntini	336010	89
14	JA Morkel	336010	89
15	S Badrinath	336010	89
16	CL White	336015	61
17	M Muralitharan	336030	78
18	L Balaji	336030	51
19	I Sharma	336030	51
20	WP Saha	336030	51
21	MS Gony	336034	100
22	JA Morkel	336034	100
23	TM Dilshan	336043	80
24	M Kaif	336043	80

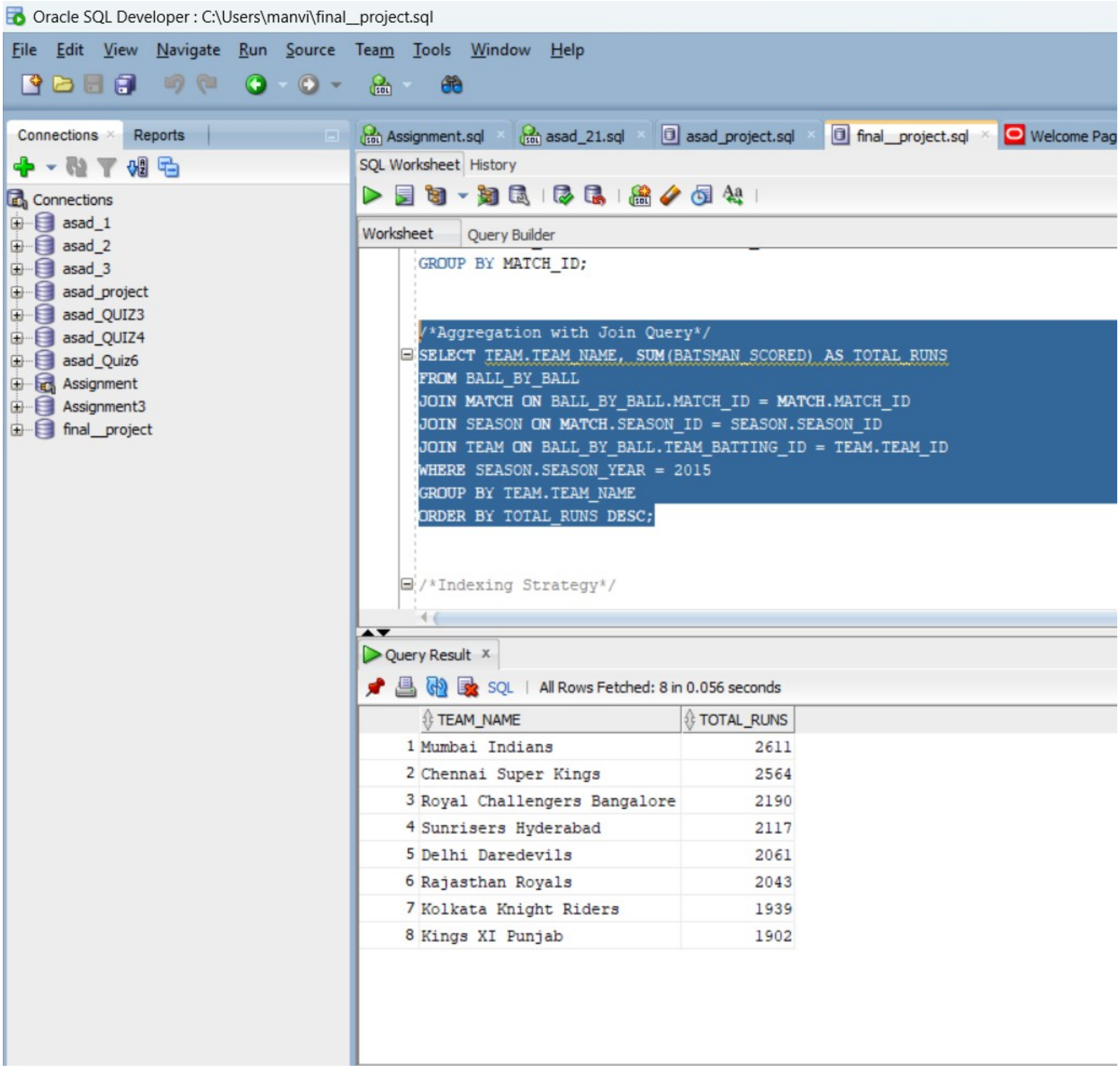
/*Combined Range and Scan Query*/

```
SELECT MATCH_ID, SUM(BATSMAN_SCORED) AS RUNS,
COUNT(DISTINCT OVER_ID) AS OVERS_FACED
FROM BALL_BY_BALL
INNER JOIN MATCH USING (MATCH_ID)
INNER JOIN SEASON USING (SEASON_ID)
INNER JOIN PLAYER_MATCH USING (MATCH_ID)
INNER JOIN PLAYER USING (PLAYER_ID)
WHERE SEASON_YEAR = 2016 AND PLAYER_NAME = 'V Kohli'
GROUP BY MATCH_ID;
```



/*Aggregation with Join Query*/

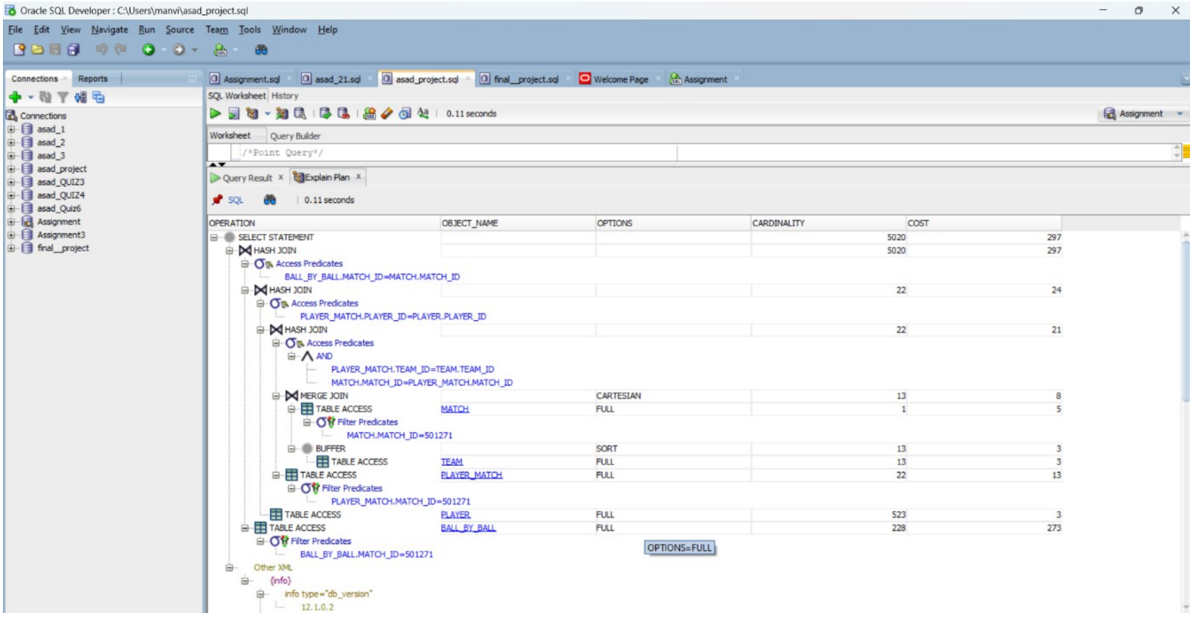
```
SELECT TEAM.TEAM_NAME, SUM(BATSMAN_SCORED) AS
TOTAL_RUNS
FROM BALL_BY_BALL
JOIN MATCH ON BALL_BY_BALL.MATCH_ID = MATCH.MATCH_ID
JOIN SEASON ON MATCH.SEASON_ID = SEASON.SEASON_ID
JOIN TEAM ON BALL_BY_BALL.TEAM_BATTING_ID =
TEAM.TEAM_ID
WHERE SEASON.SEASON_YEAR = 2015
GROUP BY TEAM.TEAM_NAME
ORDER BY TOTAL_RUNS DESC;
```


Oracle SQL Developer : C:\Users\manvi\final__project.sql
File Edit View Navigate Run Source Team Tools Window Help
Connections Reports
Assignment.sql asad_21.sql asad_project.sql final__project.sql Welcome Pag
SQL Worksheet History
Worksheet Query Builder
Connections
asad_1
asad_2
asad_3
asad_project
asad_QUIZ3
asad_QUIZ4
asad_Quiz6
Assignment
Assignment3
final__project
GROUP BY MATCH_ID;
/*Aggregation with Join Query*/
SELECT TEAM.TEAM_NAME, SUM(BATSMAN_SCORED) AS TOTAL_RUNS
FROM BALL_BY_BALL
JOIN MATCH ON BALL_BY_BALL.MATCH_ID = MATCH.MATCH_ID
JOIN SEASON ON MATCH.SEASON_ID = SEASON.SEASON_ID
JOIN TEAM ON BALL_BY_BALL.TEAM_BATTING_ID = TEAM.TEAM_ID
WHERE SEASON.SEASON_YEAR = 2015
GROUP BY TEAM.TEAM_NAME
ORDER BY TOTAL_RUNS DESC;
/*Indexing Strategy*/
Query Result
SQL All Rows Fetched: 8 in 0.056 seconds
TEAM_NAME TOTAL_RUNS
1 Mumbai Indians 2611
2 Chennai Super Kings 2564
3 Royal Challengers Bangalore 2190
4 Sunrisers Hyderabad 2117
5 Delhi Daredevils 2061
6 Rajasthan Royals 2043
7 Kolkata Knight Riders 1939
8 Kings XI Punjab 1902

Performance Tuning

1)Indexing on Point Query

```
SELECT MATCH_ID, INNINGS_ID, OVER_ID, BALL_ID,
      TEAM_NAME AS BATTING_TEAM,
      PLAYER_NAME AS BATSMAN,
      BATSMAN_SCORED AS RUNS
FROM BALL_BY_BALL
INNER JOIN MATCH USING (MATCH_ID)
INNER JOIN PLAYER_MATCH USING (MATCH_ID)
INNER JOIN PLAYER USING (PLAYER_ID)
INNER JOIN TEAM USING (TEAM_ID)
WHERE MATCH_ID = 501271 ;
```



Indexing plan

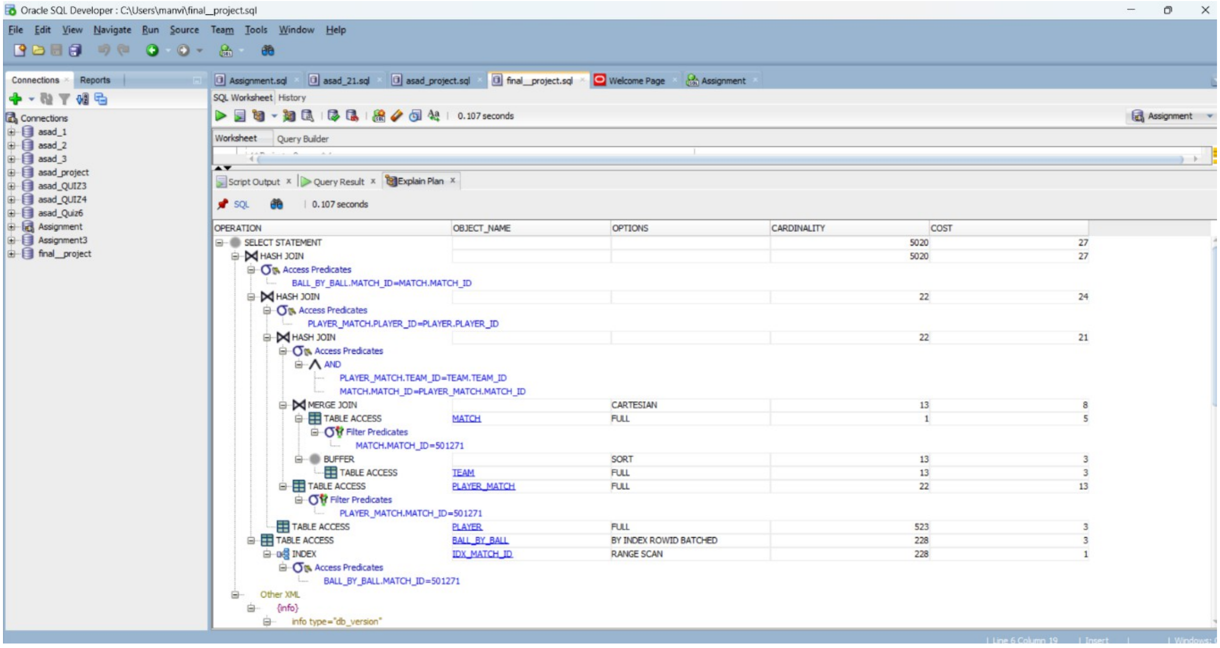
```
ALTER TABLE BALL_BY_BALL
ADD CONSTRAINT MATCH_ID PRIMARY KEY (MATCH_ID,
INNINGS_ID,OVER_ID,BALL_ID);

ALTER TABLE MATCH
ADD CONSTRAINT MATCHID PRIMARY KEY (MATCH_ID);


ALTER TABLE PLAYER
ADD CONSTRAINT PLAYERID PRIMARY KEY (PLAYER_ID);
```

Impact:

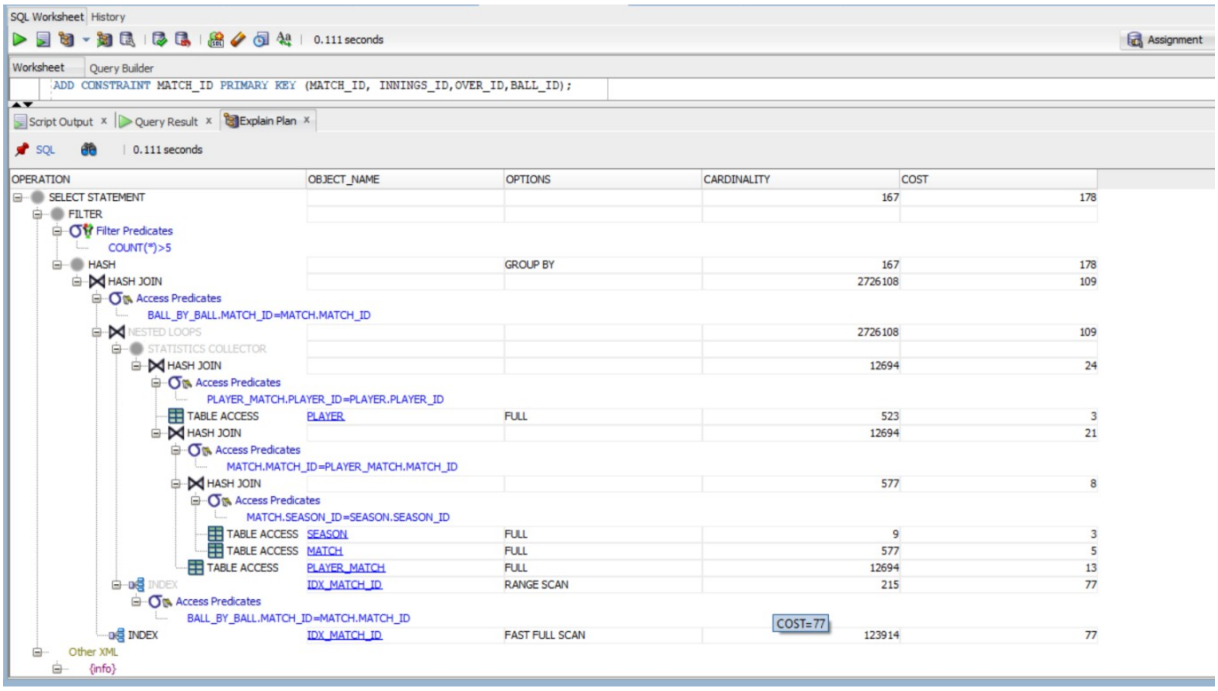
- ➢ Indexing on Match_Id column: It helps the where clause to filter matches based on match_id.
- ➢ Indexing on Ball_By_Ball column: It helps the join operator to link match_id with other columns.
- ➢ Indexing on Player column: It helps the join operator to link player_id with other columns.



➢ We can see that in the Join Operator the cost reduced from 297 to 27 because of the indexing.

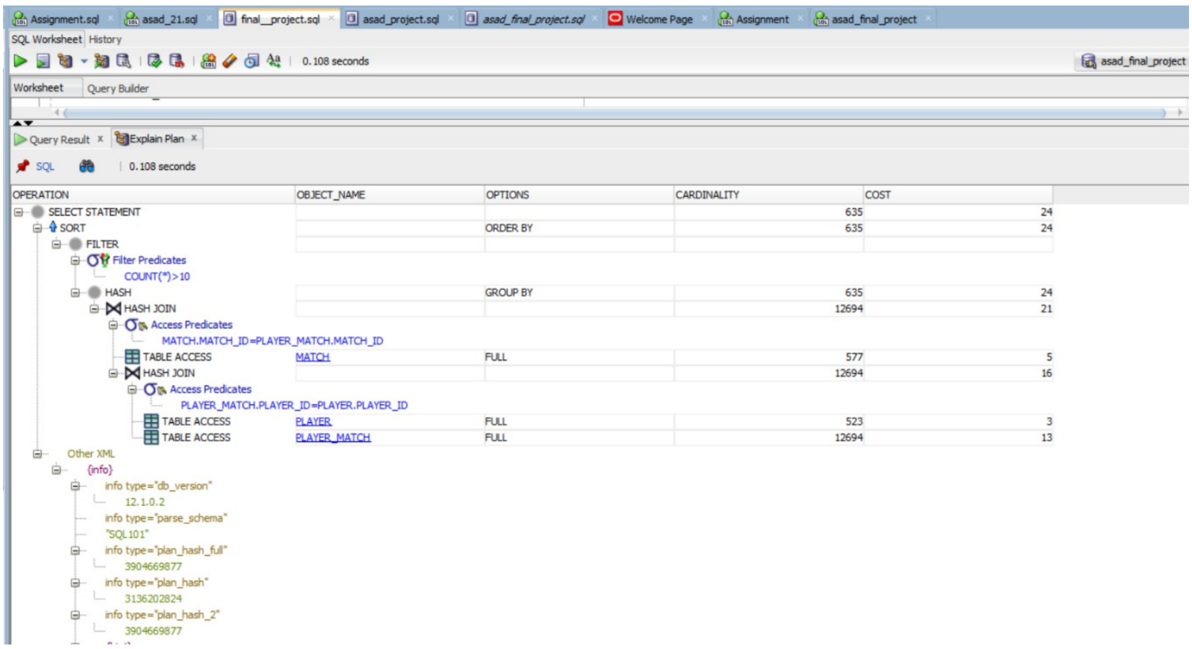
Indexing on Scan Query

```
/*Scan Query*/
SELECT PLAYER_ID, PLAYER_NAME, COUNT(*) AS
MAN_OF_THE_MATCH_COUNT
FROM MATCH
INNER JOIN PLAYER_MATCH USING (MATCH_ID) INNER
JOIN PLAYER USING (PLAYER_ID)
GROUP BY PLAYER_ID, PLAYER_NAME
HAVING COUNT(*) > 10
ORDER BY MAN_OF_THE_MATCH_COUNT DESC;
```



Indexing Plan

ALTER TABLE PLAYER_MATCH
ADD CONSTRAINT PLAYER_ID PRIMARY KEY
(MATCH_ID,PLAYER_ID);



- We can see that in the Join Operator the cost reduced from 178 to 24 because of the indexing.

Indexing on Combined Range and Scan Query

```
/*Combined Range and Scan Query*/
SELECT MATCH_ID, SUM(BATSMAN_SCORED) AS RUNS, COUNT(DISTINCT
OVER_ID) AS OVERS_FACED
FROM BALL_BY_BALL
INNER JOIN MATCH USING (MATCH_ID)
INNER JOIN SEASON USING (SEASON_ID)
INNER JOIN PLAYER_MATCH USING (MATCH_ID) INNER
JOIN PLAYER USING (PLAYER_ID)
WHERE SEASON_YEAR = 2016 AND PLAYER_NAME = 'V Kohli'
GROUP BY MATCH_ID;
```

Query Result | Explain Plan

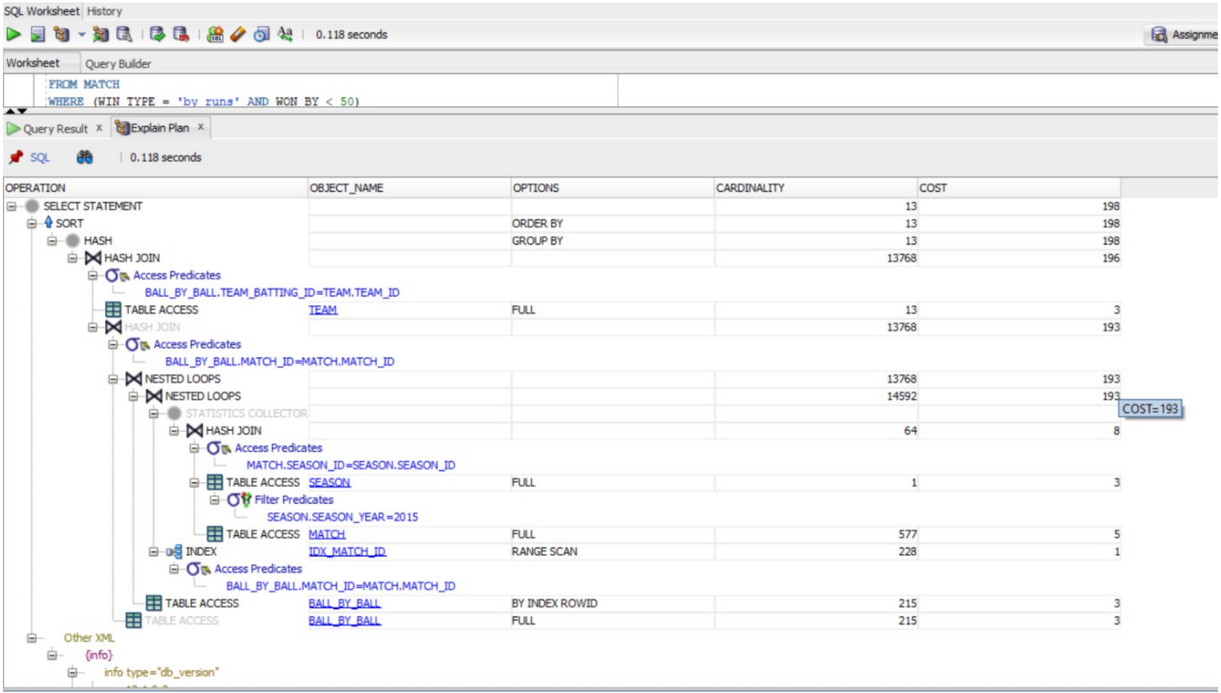
SQL | 0.12 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			543	299
HASH		GROUP BY	543	299
VIEW	SYS.VW_DAG_0		646	299
HASH		GROUP BY	646	299
HASH JOIN			646	298
Access Predicates				
BALL_BY_BALL.MATCH_ID=MATCH.MATCH_ID				
HASH JOIN			3	24
Access Predicates				
AND				
MATCH.MATCH_ID=PLAYER_MATCH.MATCH_ID				
MATCH.SEASON_ID=SEASON.SEASON_ID				
HASH JOIN			27	19
Access Predicates				
PLAYER_MATCH.PLAYER_ID=PLAYER.PLAYER_ID				
MERGE JOIN		CARTESIAN	1	6
TABLE ACCESS	PLAYER	FULL	1	3
Filter Predicates				
PLAYER.PLAYER_NAME='V Kohli'				
BUFFER		SORT	1	3
TABLE ACCESS	SEASON	FULL	1	3
Filter Predicates				
SEASON.SEASON_YEAR=2016				
TABLE ACCESS	PLAYER_MATCH	FULL	12694	13
TABLE ACCESS	MATCH	FULL	577	5
TABLE ACCESS	BALL_BY_BALL	FULL	123914	273
Other XML				
{info}				

Indexing Plan

```
ALTER TABLE SEASON
ADD CONSTRAINT SEASON_ID PRIMARY KEY (SEASON_ID);

ALTER TABLE TEAM
ADD CONSTRAINT TEAM_ID PRIMARY KEY (TEAM_ID);
```



- We can see that in the Join Operator the cost reduced from 299 to 198 because of the indexing.

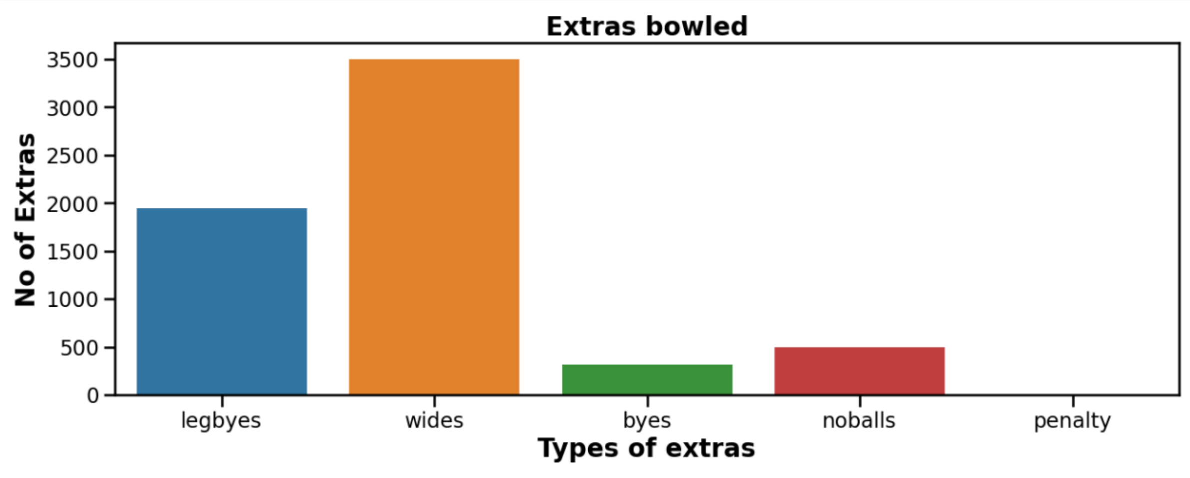
Data Visualisation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

ball_by_ball_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Ball_by_Ball.csv")
match_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Match.csv") player_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Player.csv") playermatch_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Player_Match.csv") season_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Season.csv")
Team_data=pd.read_csv("C:/Users/manvi/OneDrive/Desktop/Team.csv")
```

1) Extras bowled in the entire IPL

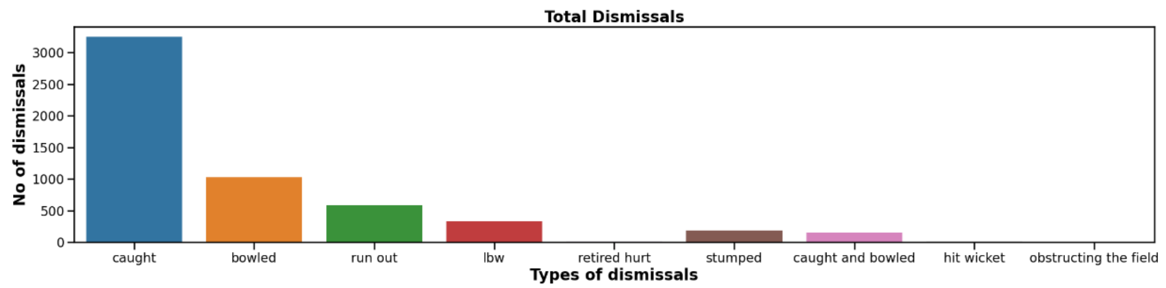
```
ball_by_ball_data['Extra_Type'].replace(' ', np.nan, inplace=True)
ball_by_ball_data['Extra_Type'].dropna()  plt.figure(figsize=(15,5))
sns.countplot(x='Extra_Type', data=ball_by_ball_data)
sns.set_context("talk")
plt.ylabel("No of Extras",fontsize = 20, weight = 'bold')
plt.xlabel("Types of extras",fontsize = 20, weight = 'bold')
plt.title("Extras bowled",fontsize = 20, weight = 'bold');
plt.show()
```



- ➢ As you can see most of the Extras are Wides which are nearly 3500.
- ➢ There are very less penalty runs.

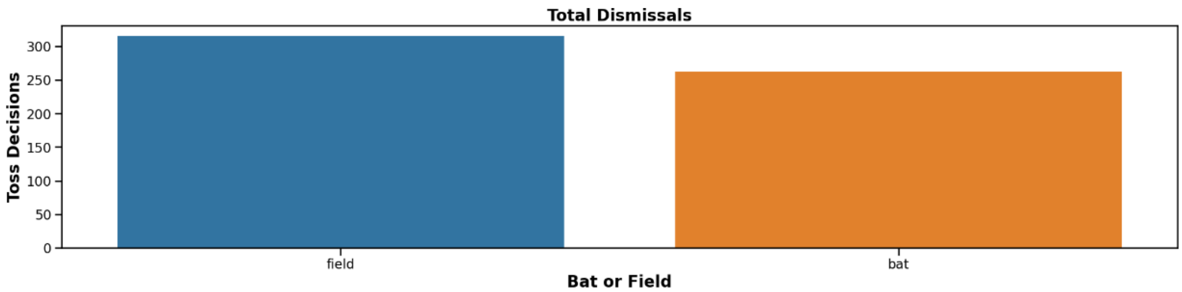
2) Different types of Dismissals

```
ball_by_ball_data['Dissimal_Type'].replace(' ', np.nan, inplace=True)
ball_by_ball_data['Dissimal_Type'].dropna()   plt.figure(figsize=(25,5))
sns.countplot(x='Dissimal_Type', data=ball_by_ball_data)
sns.set_context("talk")
plt.ylabel("No of dismissals",fontsize = 20, weight = 'bold')
plt.xlabel("Types of dismissals",fontsize = 20, weight = 'bold')
plt.title("Total Dismissals",fontsize = 20, weight = 'bold');
plt.show()
```



3) Toss Decisions

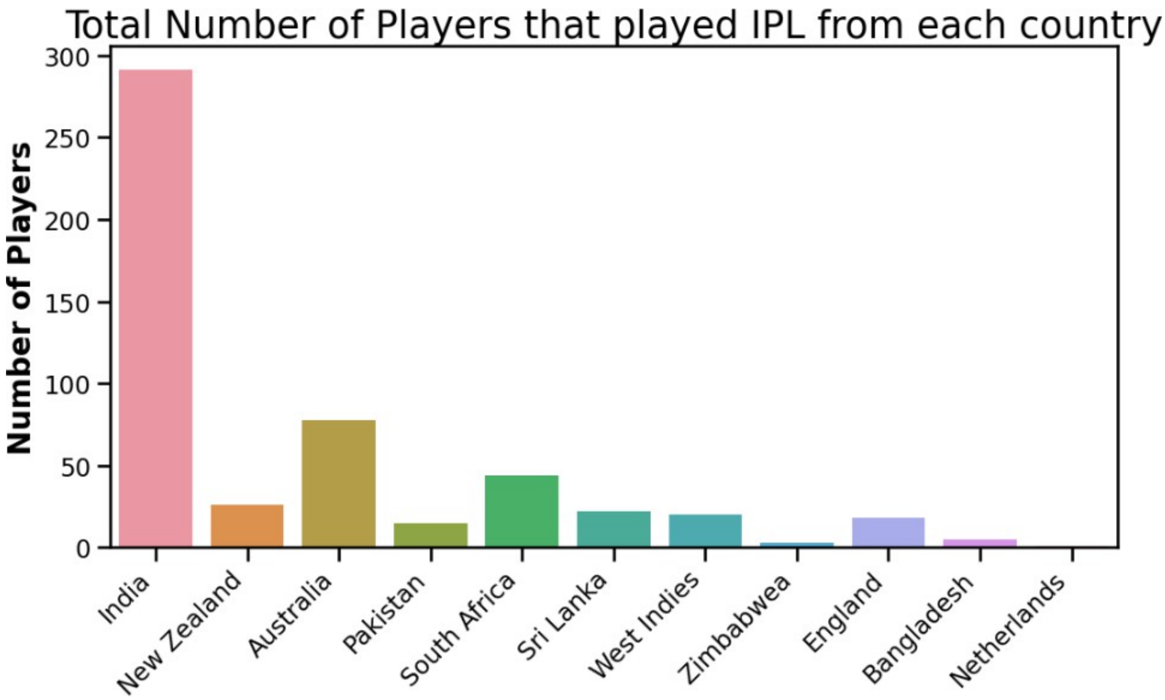
```
plt.figure(figsize=(25,5))
sns.countplot(x='Toss_Decision', data=match_data)
sns.set_context("talk")
plt.ylabel("Toss Decisions",fontsize = 20, weight = 'bold')
plt.xlabel("Bat or Field",fontsize = 20, weight = 'bold')
plt.title("Total Dismissals",fontsize = 20, weight = 'bold');
plt.show()
```



4) Number of Players from different countries

```
plt.figure(figsize=(12,6))
sns.countplot(x='Country', data=player_data)
sns.set_context('talk')
plt.xlabel("Country Names",fontsize=20,weight='bold')
plt.xticks( rotation=45, horizontalalignment='right')
plt.ylabel("Number of Players",fontsize=20,weight='bold')
plt.title("Total Number of Players that played IPL from each country",fontsize=25)
```

```
plt.show()
```



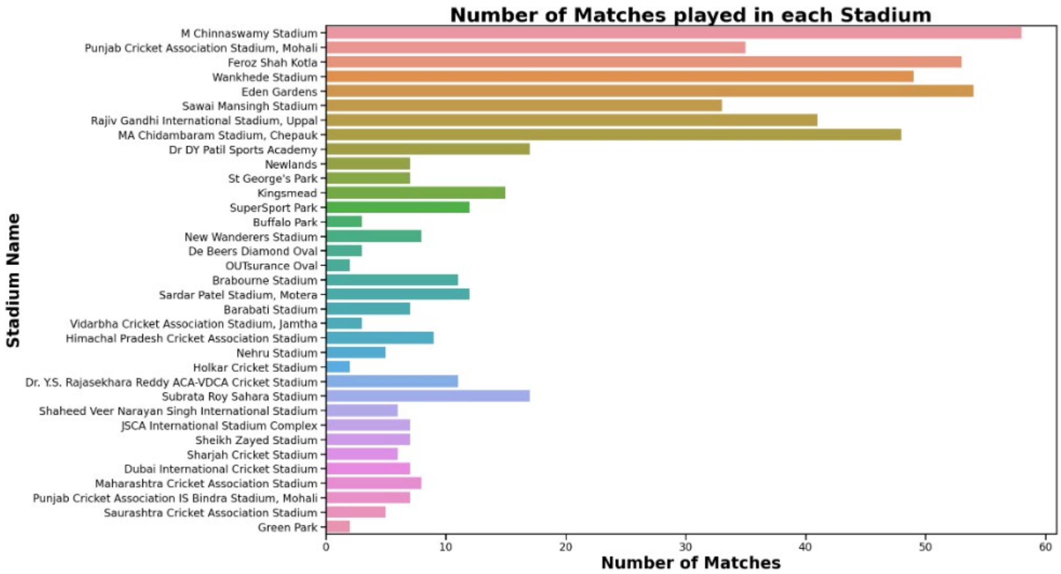
5) Host Countries of IPL

```
match_data['Host_Country'].unique()
```

```
array(['India', 'South Africa', 'U.A.E'], dtype=object)
```

6) Number of matches played in each stadium

```
plt.figure(figsize=(20,14))
sns.countplot(y='Venue_Name', data=match_data)
plt.yticks(rotation='horizontal')
plt.xlabel("Number of Matches",fontsize = 25, weight = 'bold')
plt.ylabel("Stadium Name",fontsize = 25, weight = 'bold')
plt.title("Number of Matches played in each Stadium",fontsize = 30, weight = 'bold');
plt.show()
```



Data Mining

Predicting the Result of the match based on team winning the Toss. We have considered the following columns "Toss_Decision","Match_Winner_Id","City_Name","Team_Name_Id","Opponent_Team_Id", for predicting of the result

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

data=match_data[["Toss_Decision","Match_Winner_Id","City_Name","Team_Name_Id"
,"Opponent_Team_Id"]]

# Drop rows with missing values
data = data.dropna()

# Convert categorical columns to numerical using Label Encoding
label_encoder = LabelEncoder()
data["Toss_Decision"] = label_encoder.fit_transform(data["Toss_Decision"])
data["City_Name"] = label_encoder.fit_transform(data["City_Name"])
```

```
# Separate features (X) and target variable (y)
X = data.drop("Match_Winner_Id", axis=1)
y = data["Match_Winner_Id"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize different classifiers
rf_classifier = RandomForestClassifier(random_state=42)
gb_classifier = GradientBoostingClassifier(random_state=42)
svm_classifier = SVC(random_state=42)

# Train and evaluate Random Forest classifier
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Classifier Accuracy:", rf_accuracy)
print("Classification Report:")
print(classification_report(y_test, rf_predictions))

# Train and evaluate Gradient Boosting classifier
gb_classifier.fit(X_train, y_train)
gb_predictions = gb_classifier.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_predictions)
print("\nGradient Boosting Classifier Accuracy:", gb_accuracy)
print("Classification Report:")
print(classification_report(y_test, gb_predictions))

# Train and evaluate Support Vector Machine classifier
svm_classifier.fit(X_train, y_train)
svm_predictions = svm_classifier.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("\nSupport Vector Machine Classifier Accuracy:", svm_accuracy)
print("Classification Report:")
print(classification_report(y_test, svm_predictions))

# Logistic Regression
logreg_classifier = LogisticRegression(random_state=42)
logreg_classifier.fit(X_train, y_train)
logreg_predictions = logreg_classifier.predict(X_test)
logreg_accuracy = accuracy_score(y_test, logreg_predictions)
print("\nLogistic Regression Accuracy:", logreg_accuracy)
print("Classification Report:")
print(classification_report(y_test, logreg_predictions))

# K-Nearest Neighbors
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("\nK-Nearest Neighbors Accuracy:", knn_accuracy)
print("Classification Report:")
print(classification_report(y_test, knn_predictions))
```

```
# Naive Bayes
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("\nNaive Bayes Accuracy:", nb_accuracy)
print("Classification Report:")
print(classification_report(y_test, nb_predictions))

# Decision Tree
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("\nDecision Tree Classifier Accuracy:", dt_accuracy)
print("Classification Report:")
print(classification_report(y_test, dt_predictions))

# AdaBoost
adaboost_classifier = AdaBoostClassifier(random_state=42)
adaboost_classifier.fit(X_train, y_train)
adaboost_predictions = adaboost_classifier.predict(X_test)
adaboost_accuracy = accuracy_score(y_test, adaboost_predictions)
print("\nAdaBoost Classifier Accuracy:", adaboost_accuracy)
print("Classification Report:")
print(classification_report(y_test, adaboost_predictions))
```

```
Random Forest Classifier Accuracy: 0.3652173913043478
Classification Report:
```

	precision	recall	f1-score	support
1.0	0.39	0.78	0.52	9
2.0	0.56	0.50	0.53	18
3.0	0.75	0.60	0.67	15
4.0	0.10	0.08	0.09	13
5.0	0.31	0.56	0.40	9
6.0	0.29	0.11	0.16	18
7.0	0.33	0.46	0.39	13
8.0	0.17	0.25	0.20	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.25	0.50	0.33	4
12.0	0.00	0.00	0.00	2
13.0	0.00	0.00	0.00	4
accuracy			0.37	115

macro avg	0.24	0.29	0.25	115
weighted avg	0.35	0.37	0.34	115

Gradient Boosting Classifier Accuracy: 0.4782608695652174
Classification Report:

	precision	recall	f1-score	support
1.0	0.47	0.78	0.58	9
2.0	0.55	0.33	0.41	18
3.0	0.53	0.53	0.53	15
4.0	0.18	0.15	0.17	13
5.0	0.54	0.78	0.64	9
6.0	0.53	0.50	0.51	18
7.0	0.50	0.62	0.55	13
8.0	0.50	0.50	0.50	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.50	0.75	0.60	4
12.0	1.00	0.50	0.67	2
13.0	0.67	0.50	0.57	4
accuracy			0.48	115
macro avg	0.46	0.46	0.44	115
weighted avg	0.47	0.48	0.46	115

Support Vector Machine Classifier Accuracy: 0.30434782608695654
Classification Report:

	precision	recall	f1-score	support
1.0	0.38	0.89	0.53	9
2.0	0.46	0.33	0.39	18
3.0	0.33	0.60	0.43	15
4.0	0.12	0.08	0.10	13
5.0	0.30	0.33	0.32	9
6.0	0.00	0.00	0.00	18
7.0	0.19	0.38	0.25	13
8.0	0.00	0.00	0.00	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.33	0.75	0.46	4

12.0	0.00	0.00	0.00	2
13.0	0.00	0.00	0.00	4
accuracy			0.30	115
macro avg	0.16	0.26	0.19	115
weighted avg	0.22	0.30	0.24	115

Logistic Regression Accuracy: 0.2782608695652174
Classification Report:

	precision	recall	f1-score	support
1.0	0.37	0.78	0.50	9
2.0	0.56	0.28	0.37	18
3.0	0.35	0.53	0.42	15
4.0	0.13	0.15	0.14	13
5.0	0.33	0.11	0.17	9
6.0	0.00	0.00	0.00	18
7.0	0.15	0.38	0.22	13
8.0	0.00	0.00	0.00	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.31	1.00	0.47	4
12.0	0.00	0.00	0.00	2
13.0	0.00	0.00	0.00	4
accuracy			0.28	115
macro avg	0.17	0.25	0.18	115
weighted avg	0.23	0.28	0.22	115

K-Nearest Neighbors Accuracy: 0.28695652173913044
Classification Report:

	precision	recall	f1-score	support
1.0	0.29	0.78	0.42	9
2.0	0.25	0.28	0.26	18
3.0	0.50	0.33	0.40	15
4.0	0.16	0.23	0.19	13
5.0	0.33	0.33	0.33	9
6.0	0.43	0.17	0.24	18
7.0	0.27	0.31	0.29	13
8.0	0.00	0.00	0.00	4
9.0	0.00	0.00	0.00	4
10.0	1.00	0.50	0.67	2
11.0	0.33	0.50	0.40	4
12.0	0.00	0.00	0.00	2
13.0	0.00	0.00	0.00	4
accuracy			0.29	115
macro avg	0.27	0.26	0.25	115
weighted avg	0.30	0.29	0.27	115

Naive Bayes Accuracy: 0.28695652173913044
Classification Report:

	precision	recall	f1-score	support
1.0	0.42	0.89	0.57	9
2.0	0.75	0.17	0.27	18
3.0	0.38	0.53	0.44	15
4.0	0.17	0.31	0.22	13
5.0	0.33	0.11	0.17	9
6.0	0.00	0.00	0.00	18
7.0	0.23	0.54	0.32	13
8.0	0.00	0.00	0.00	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.17	0.50	0.25	4
12.0	0.00	0.00	0.00	2

13.0	0.00	0.00	0.00	4
accuracy			0.29	115
macro avg	0.19	0.23	0.17	115
weighted avg	0.28	0.29	0.23	115

Decision Tree Classifier Accuracy: 0.40869565217391307
Classification Report:

	precision	recall	f1-score	support
1.0	0.37	0.78	0.50	9
2.0	0.58	0.39	0.47	18
3.0	0.73	0.73	0.73	15
4.0	0.08	0.08	0.08	13
5.0	0.33	0.56	0.42	9
6.0	0.44	0.22	0.30	18
7.0	0.41	0.54	0.47	13
8.0	0.33	0.25	0.29	4
9.0	0.00	0.00	0.00	4
10.0	0.20	0.50	0.29	2
11.0	0.60	0.75	0.67	4
12.0	0.00	0.00	0.00	2
13.0	0.00	0.00	0.00	4
accuracy			0.41	115
macro avg	0.31	0.37	0.32	115
weighted avg	0.40	0.41	0.39	115

AdaBoost Classifier Accuracy: 0.2608695652173913
Classification Report:

	precision	recall	f1-score	support
1.0	0.57	0.89	0.70	9
2.0	0.00	0.00	0.00	18
3.0	0.48	0.73	0.58	15
4.0	0.00	0.00	0.00	13
5.0	0.00	0.00	0.00	9
6.0	0.00	0.00	0.00	18
7.0	0.14	0.77	0.23	13
8.0	0.00	0.00	0.00	4
9.0	0.00	0.00	0.00	4
10.0	0.00	0.00	0.00	2
11.0	0.00	0.00	0.00	4

12.0	0.00	0.00	0.00	2
13.0	0.25	0.25	0.25	4
accuracy			0.26	115
macro avg	0.11	0.20	0.13	115
weighted avg	0.13	0.26	0.16	115

We have considered 7 different models for prediction, but only Gradient Boosting Classifier is giving an Accuracy of 0.5 which is best among the other models. It predicts that the team winning the toss has nearly 50% chance of winning the match.