

Name: Aniket Devidas Kale
Class: DSS640
Project: IPL Data.

Introduction:

What is IPL?

The IPL is the most-attended cricket league in the world and ranks sixth among all sports leagues. In 2010, the IPL became the first sporting event in the world to be broadcast live on YouTube. The brand value of IPL was estimated to be US\$4.5 billion in 2015 by American Appraisal, A Division of Duff & Phelps.

The Indian Premier League (IPL, officially Vivo Indian Premier League for sponsorship reasons) is a professional Twenty20 cricket league in India contested during April and May of every year by teams representing Indian cities. The league was founded by the Board of Control for Cricket in India (BCCI)

Git hub repository: <https://github.com/andy12290/Datawarehouse>

Business Use case:

I have collected data from the different resources for the past 9 seasons of IPL, in this context, I see the scopes of Project to build the data warehouse in order to access the some of the records very quickly to see the statistics of the player to analyzes the team performance and the player performance in different venue.

Once we create the Data warehouse, we can use the data to make some modeling or gave access to data scientist team or whoever need to access the files.

With the help of player performance in each season, we can use the data to auction the players in next seasons or find out the individual performance to check the best team combination or Find out the best combination with the help of cricketing knowledge.

Note: Analysis will only help in order to understand the player performance but the cricketing knowledge will also be important to find out the best combination of players.

Business Questions:

In any sports analysis, the most important part is the performance of team and player. In IPL, I am trying to the analyze the past matches data and want to come up with the season analysis and runs analysis for each venue. In cricketing field, the most important part is pitch and venue, Since the weather of that venue matters the most in the match.

So at the end, if we go with the analysis then we will come to know which venue would be good for batting first or How the previous innings build here. That will help to understand or to build the new strategy for the team.

Basically, I would try to answer the below questions for my team to analyze the season and venue.

- Season wise analysis.
- How many extra runs bowlers had given per season?
- How many runs scored in Each venue?
- Home many matches played in each venue?
- Runs scored in each season
- Runs scored per Innings
- Win margin in first innings and second innings

Data source:

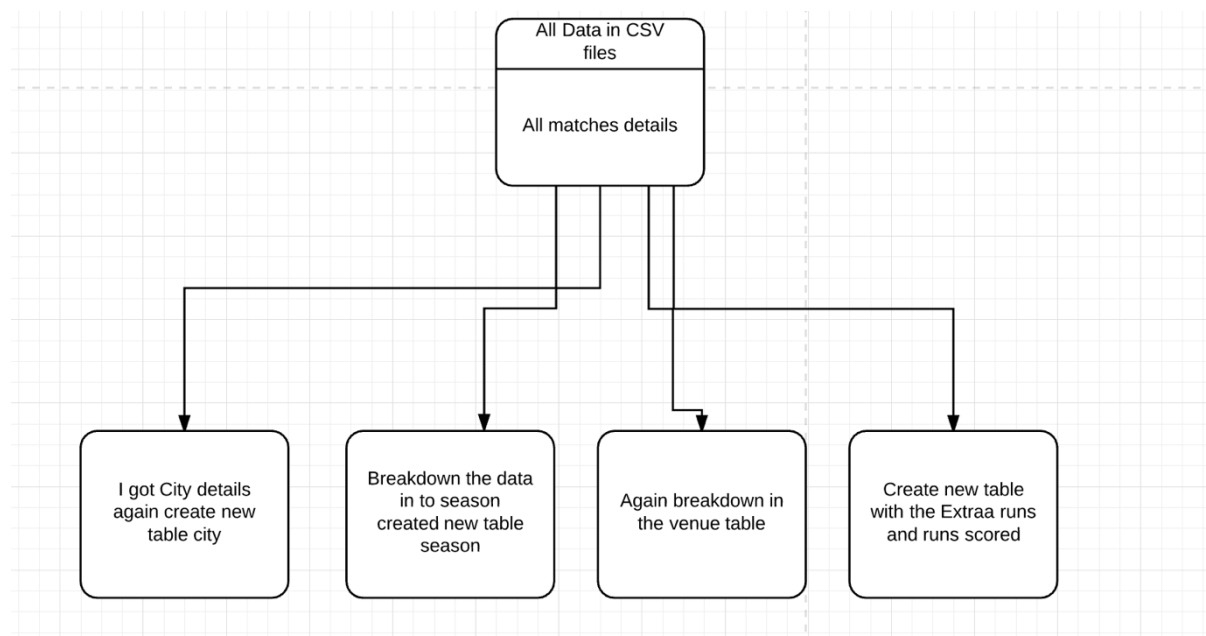
Data source: Kaggle, Ipl.com, Espncricinfo, Data world

I have collected all data in the CSV format from different resources like Kaggle, data world, Espncricinfo.

I have a lot of data in the tabular format season wise, city, player, match

Table Design and ETL Process:

First of all, I have collected data from the different resources in the tabular format.



Generated own data:

- 1) Season table
- 2) Venue table
- 3) City table

After collection of data:

- I have created the Season ID a primary key (Surrogate key) for season table
- Natural key is the season year.

The same process I repeat for the Venue table.

- Venue Id is surrogate key. Auto increment number.
- Venue name

Then same process I used for city table.

- City Name
- City Id: Increment integer

Collected data from Kaggle and data world and add some of the columns to create the following table:

4)Player table

5)Extra runs Table

6)Runs scored table

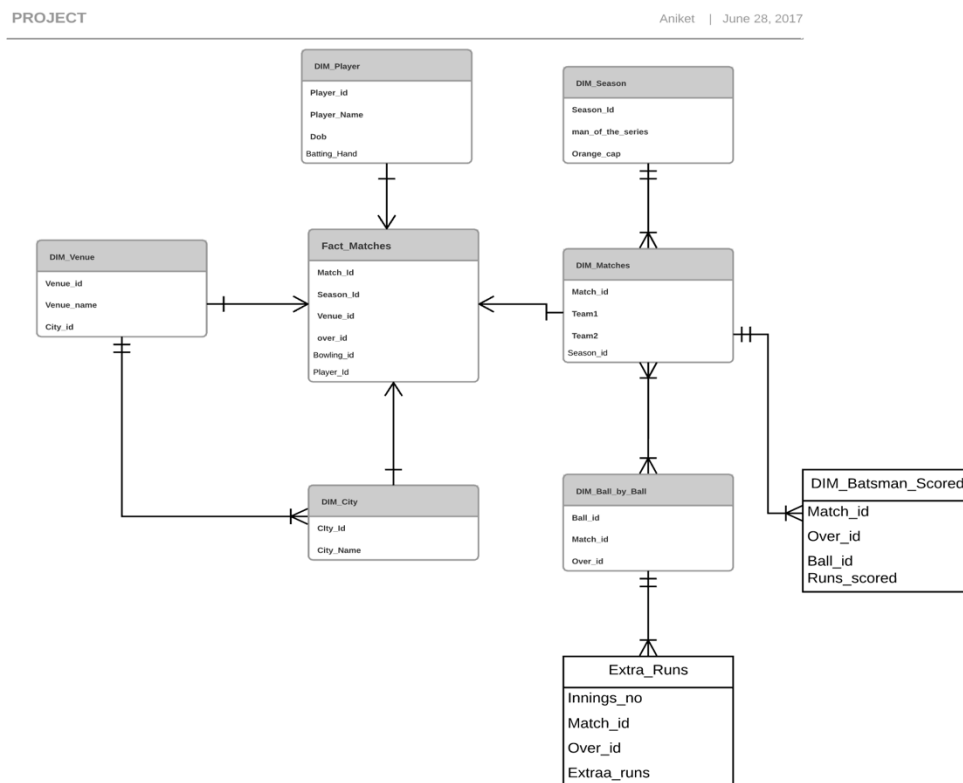
7)Ball_by_ball Table

8)match table

Logical Design (first draft):

Loaded all CSV files into the Oracle and create following tables

E-R diagrams:



E-R diagram: First draft while designing the tables and fact tables.

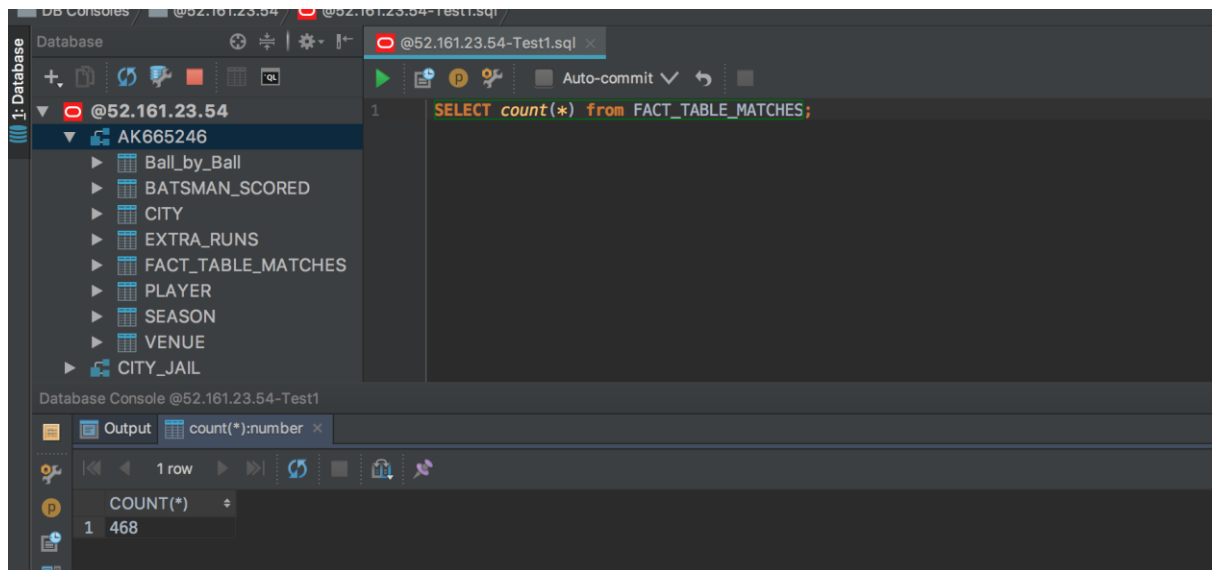
Table details:

Fact table:

1) Fact table Matches:

- Team_id = Team ID for each match PK
- match ID: Match Id for each match
- over_id: Over ID for Each match
- ball_id: Ball ID for each Innings
- innings_no: Innings Number 1 or 2
- season_id: Season ID 1 to 9
- venue_id: Venue ID
- player_id: Player ID for each Player
- player_name: Player Name
- city_id: City of each Venue.
- Player name: Name of Player
- Win Margine_runs: Count Win margin Runs for each team
- Player Runs: Player Runs for each match
- Innings Number: 1 or 2 Innings
- Team Runs: Team runs for each match

Fact Table	
Team_Id	Dimensions
Match_Id	Dimensions
Over_Id	Dimensions
Ball_Id	Dimensions
Innings_No	Dimensions
Season_Id	Dimensions
Venue_Id	Dimensions
Player_Id	Dimensions
Player_Name	Measures
City_id	Measures
Win_Margin_Runs	Measures
Player_Runs	Measures
Innings_number	Measures
Team_Runs	Measures

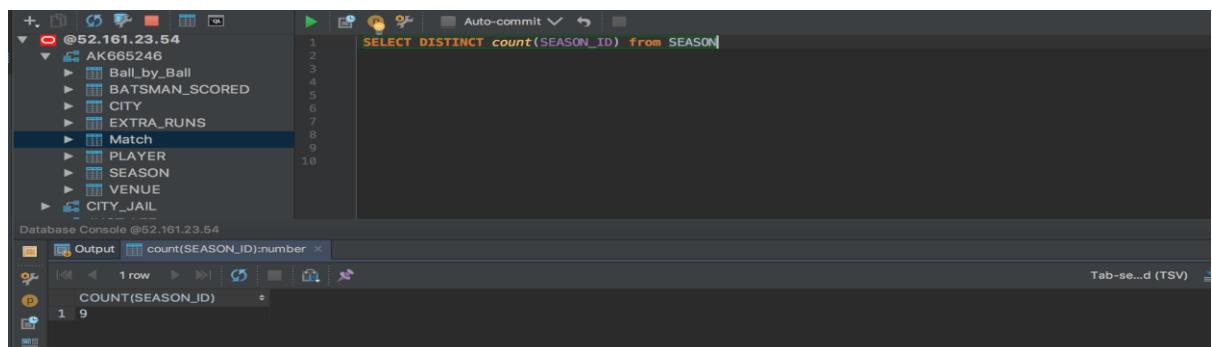


Breakdown the data in to season wise:

Season table:

- season_id: Season ID 1 to 9(PK)
- man_of_the_series: Man of the Series Number for each Player
- orange_cap: How many caps
- purple_cap: Purple cap in that season
- season_year: Year 2008 to 2015

```
CREATE TABLE AK665246.Season
(
  Season_Id INTEGER PRIMARY KEY,
  Man_of_the_Series INTEGER,
  Orange_Cap INTEGER,
  Purple_Cap INTEGER,
  Season_Year INTEGER
);
```



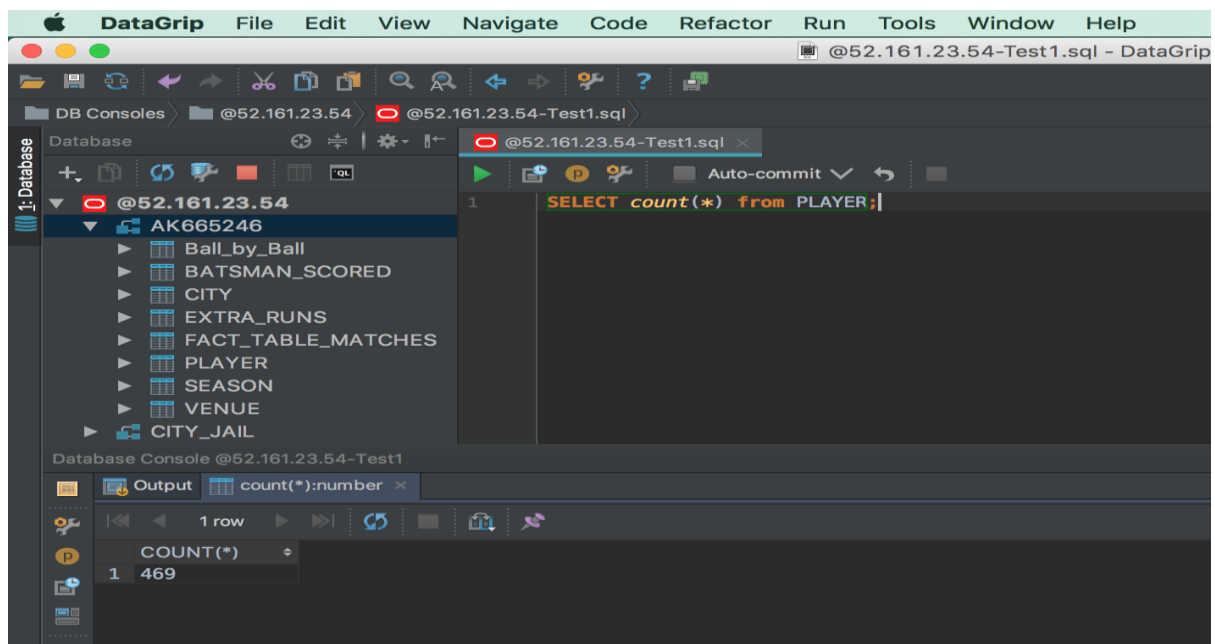
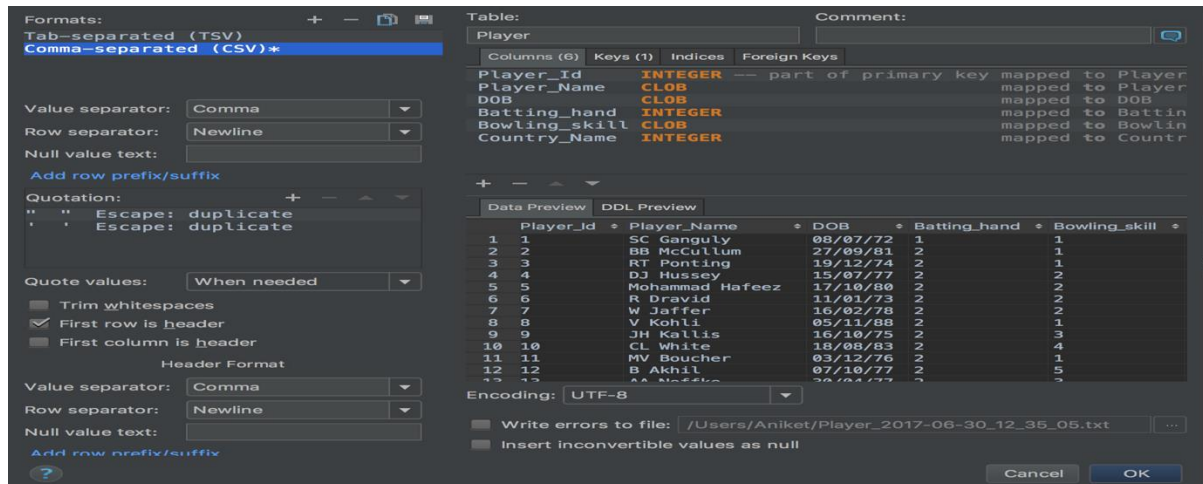
Note: I have updated all the csv files and result in the python Notebook.

Player Table:

- player_id: ID (PK)

- player_name: Name of Each player
- Dob: DOB
- batting_hand: Right or left
- bowling_skill: 1 to 4 skills
- country_name: Name

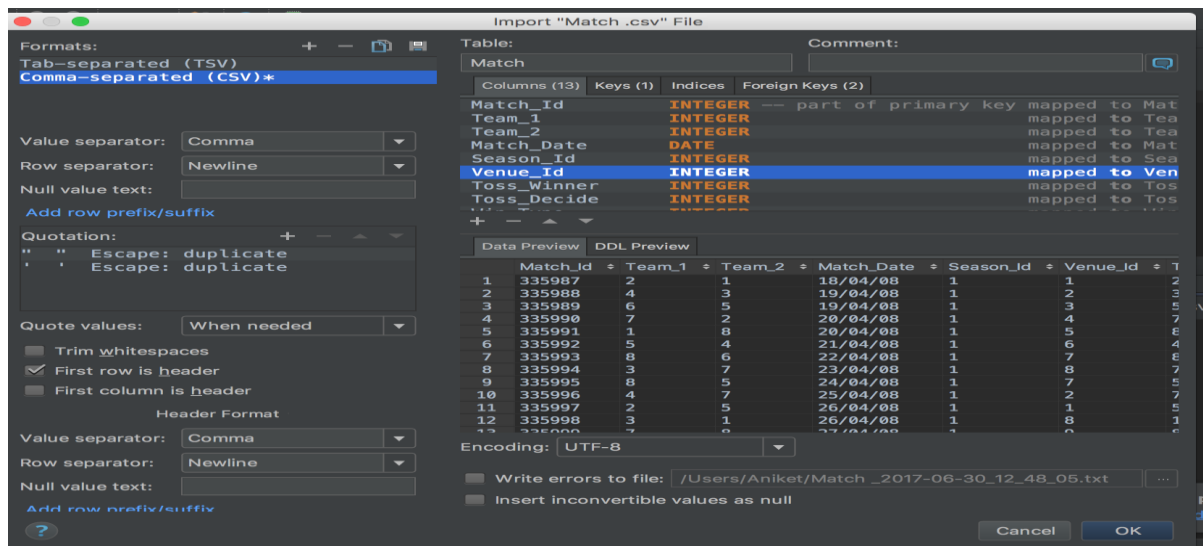
In the Player table, you will see the details of each player.



Match table:

- ID= Id for Each records, surrogate key for unique number PK
- match_id: Id for each match
- team_1: Team 1 or 2
- team_2: Team ID
- match_date: Date of match
- season_id: ID 1 to 9

- venue_id: Venue ID
- toss_winner: Id for each team
- toss_decide: 1 or 2 win or loss
- win_type: 1 Win 2 Loss
- win_margin: Runs
- outcome_type: 1 Win
- match_winner: Player id
- man_of_the_match: scores of match winner



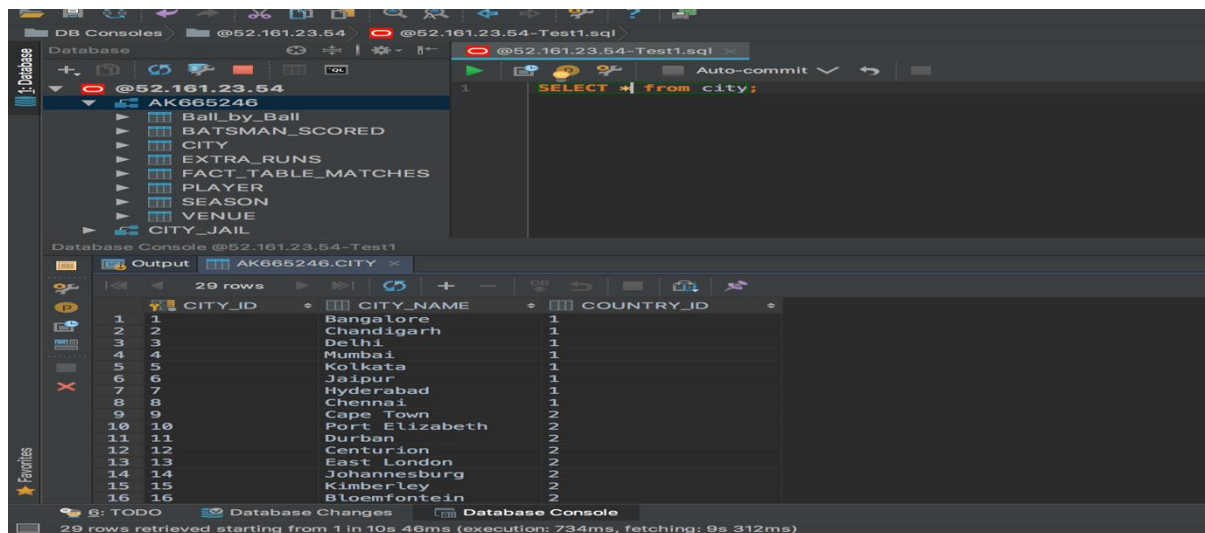
Loaded the CSV files into the Oracle:

City Table:

- city_id: ID for each city(PK)
- city_name: Name of city
- country_id: ID for country

We will see the city table

- City ID = PK
- City Name
- Country ID = FK to Country table

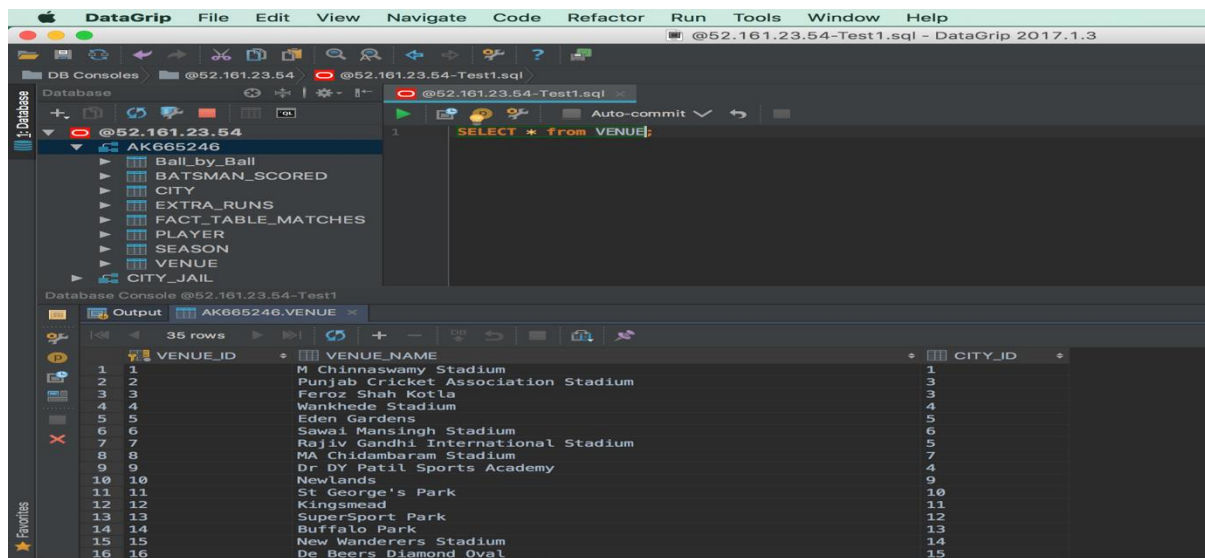


Venue table:

- venue_id: ID for each Venue
- venue_name: Name
- city_id: City if for each city

Venue ID = Primary Key

City-ID= FK and related to City table.



Extra runs:

Table to count the extra runs

- Extra_ID= ID to identify each match records PK

- match_id: ID for each match
- over_id: Id for over in that match
- ball_id: Ball of that match
- extra_type_id: Extra id for which type
- extra_runs: runs given in Match
- innings_no: Innings 1 or 2

The screenshot shows a database console interface. On the left, a tree view displays the database structure for '@52.161.23.54', including tables like 'Ball_by_Ball', 'BATSMAN_SCORED', 'CITY', 'EXTRA_RUNS', 'FACT_TABLE_MATCHES', 'PLAYER', 'SEASON', 'VENUE', and 'CITY_JAIL'. The main query editor contains the SQL statement: `SELECT count(*) from EXTRA_RUNS;`. Below the editor, the 'Output' tab shows the result: a single row with the value 7469 for the 'COUNT(*)' column.

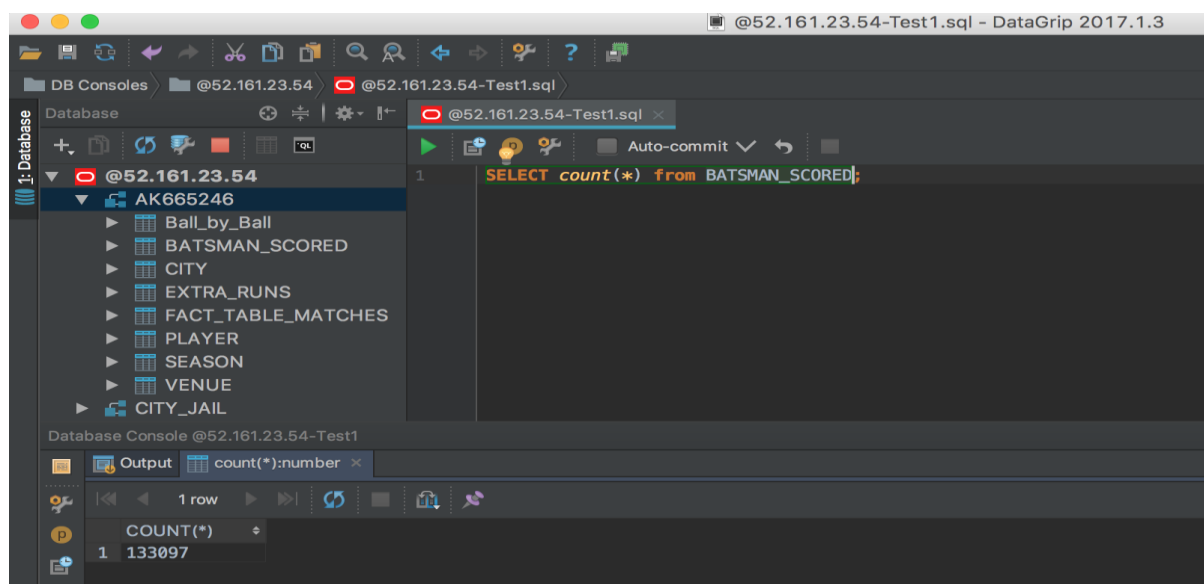
The screenshot shows a table export dialog box for the 'Extra_Runs' table. The 'Formats' section on the left has 'Comma-separated (CSV)*' selected. The 'Table' section on the right lists the columns and their data types: 'Extra_runs_ID' (INTEGER), 'Match_Id' (INTEGER), 'Over_Id' (INTEGER), 'Ball_Id' (INTEGER), 'Extra_Type_Id' (INTEGER), 'Extra_Runs' (INTEGER), and 'Innings_No' (INTEGER). The 'Data Preview' tab shows a sample of the data. The 'Encoding' is set to 'UTF-8'. The 'Write errors to file' checkbox is checked, and the file path is 'Users/Aniket/Extra_Runs_2017-06-30_12_53_11.txt'. The 'OK' button is highlighted.

	Extra_runs_ID	Match_Id	Over_Id	Ball_Id	Extra_Type_Id	Extra_Run
1	1	335987	1	1	1	1
2	2	335987	1	2	2	1
3	3	335987	1	3	2	1
4	4	335987	1	7	1	1
5	5	335987	2	3	1	4
6	6	335987	3	3	1	1
7	7	335987	3	5	2	1
8	8	335987	4	1	2	5
9	9	335987	4	3	1	1
10	10	335987	4	4	1	1
11	11	335987	7	4	2	1
12	12	335987	7	5	2	1
13	13	335987	8	2	1	1

Batsman scored table:

Count each batsman scored.

- Batsman_Id = Id to count each batsman scored(PK)
- Match_id = Each match number
- over_id: Over Id for each match
- ball_id: Ball id for each match
- runs_scored: runs scored in each match
- innings_no: Inning 1 or 2

**Ball by ball Table:**

- Bolwer_id: PK Unique key for Bowler
- match_id: Match_d for that match to identify the number
- over_id: Over id to identify the over
- ball_id: Ball id to identify the number of balls in over
- innings_no: Number of inning 1 or 2
- team_batting: first team batting or second 1 or 2
- team_bowling: 1 or 2
- striker_batting_position: striker or non-striker
- bowler: Bowler left hand or right hand.

Data warehouse Design:**Fact table:**

A fact table is used in the dimensional model in data warehouse design. A fact table is found at the center of a star schema or snowflake schema surrounded by dimension tables.

What we are going to Measure?

Star Schema:

Star schema architecture is the simplest data warehouse design. The main feature of a star schema is a table at the center, called the **fact table** and the **dimension tables** which allow browsing of specific categories, summarizing, drill-downs and specifying criteria.

Typically, most of the fact tables in a star schema are in database third normal form, while dimensional tables are de-normalized (second normal form).

Fact Table:

Matches table: Details about the Match for every season.
Included all the Ids like season ID, Match ID, Player ID

Fact Table	
Team_Id	Dimensions
Match_Id	Dimensions
Over_Id	Dimensions
Ball_Id	Dimensions
Innings_No	Dimensions
Season_Id	Dimensions
Venue_Id	Dimensions
Player_Id	Dimensions
Player_Name	Measures
City_id	Measures
Win_Margin_Runs	Measures
Player_Runs	Measures
Innings_number	Measures
Team_Runs	Measures

Measures:

- Player_Runs: Count Player Runs
- Win_Margin_Runs: Count each match win margins
- TeamRuns: Count Team runs to access quickly
- Inning_number: count of each innings
- City – count of cities

Dimension table:

- Season table
- City table
- Venue Table
- Player Table
- Ball by Ball Table
- Match Table
- Batsman scored table
- Extra runs table

Nearly all of the information in a typical fact table is also present in one or more dimension tables. The main purpose of maintaining Dimension Tables is to allow browsing the categories quickly and easily.

However, we are going to use the Snowflake schema since we have more dimension's table which are internally connected.

Why Snowflake schema?

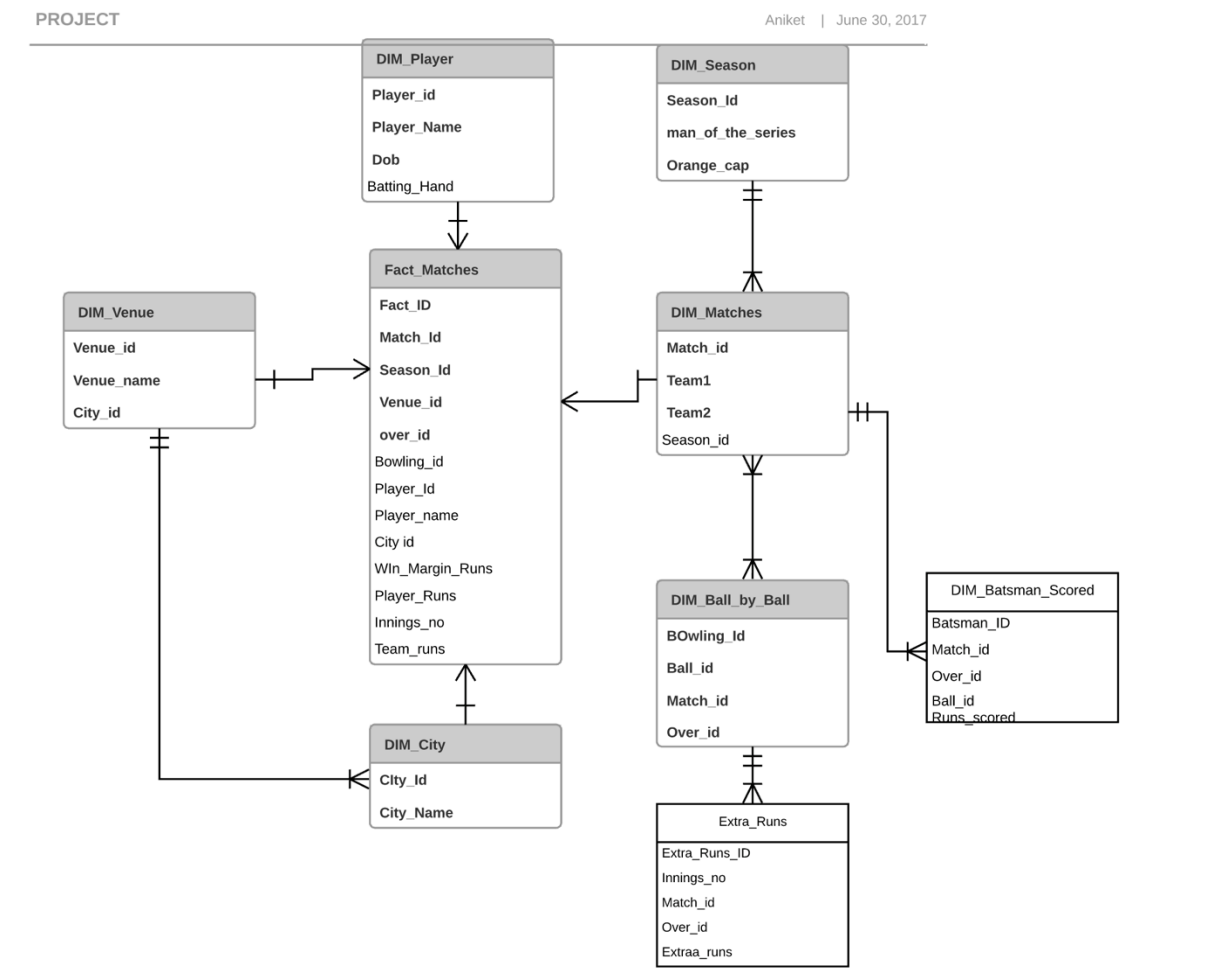
Snowflake schema consists of a fact table surrounded by multiple dimension tables which can be connected to other dimension tables via many-to-one relationship. the snowflake schema is a kind of star schema however it is more complex than a star schema in term of the data model.

We have a lot of dimensions' table, which are connected to each other.

For example:

Matches → Season → Ball_by_ball

Venue → City



Implementations:

Loaded all the tables.

SQL Scripts:

Below are the Sql scripts in order to see the answers of questions.

Script:

Number of batsman's Runs in Inning 1 and 2

```

SELECT sum(RUNS_SCORED),INNINGS_NO
FROM BATSMAN_SCORED
GROUP BY BATSMAN_SCORED.INNINGS_NO
ORDER BY INNINGS_NO;
  
```

The screenshot shows a database console interface. On the left, a tree view displays the database structure for 'AK665246', including tables 'Ball_by_Ball' and 'BATSMAN_SCORED'. The 'BATSMAN_SCORED' table structure is detailed with columns: 'BATSMAN_ID' (NUMBER), 'MATCH_ID' (NUMBER(*)), 'OVER_ID' (NUMBER(*)), 'BALL_ID' (NUMBER(*)), 'RUNS_SCORED' (NUMBER), 'INNINGS_NO' (NUMBER), and two system columns 'SYS_C0010075'. The main editor on the right contains the following SQL query:

```

1 SELECT sum(RUNS_SCORED), INNINGS_NO
2 FROM BATSMAN_SCORED
3 GROUP BY BATSMAN_SCORED.INNINGS_NO
4 ORDER BY INNINGS_NO

```

Below the editor, the 'Database Console @52.161.23.54-inde' section shows the execution results. It indicates '4 rows' and displays a table with the following data:

	SUM(RUNS_SCORED)	INNINGS_NO
1	86818	1
2	78993	2
3	80	3
4	70	4

Conclusion: Innings score 1 is more as compared to 2

Multiple Joining:

Season wise batsman scored

Result: Result uploaded in Notebooks.

```

select match.match_id, season.season_id, season.season_year,
sum(batsman_scored.runs_scored)
from batsman_scored
inner join match on match.match_id = batsman_scored.match_id
inner join season on season.season_id = match.season_id
group by season.season_id, season.season_year
order by season_year

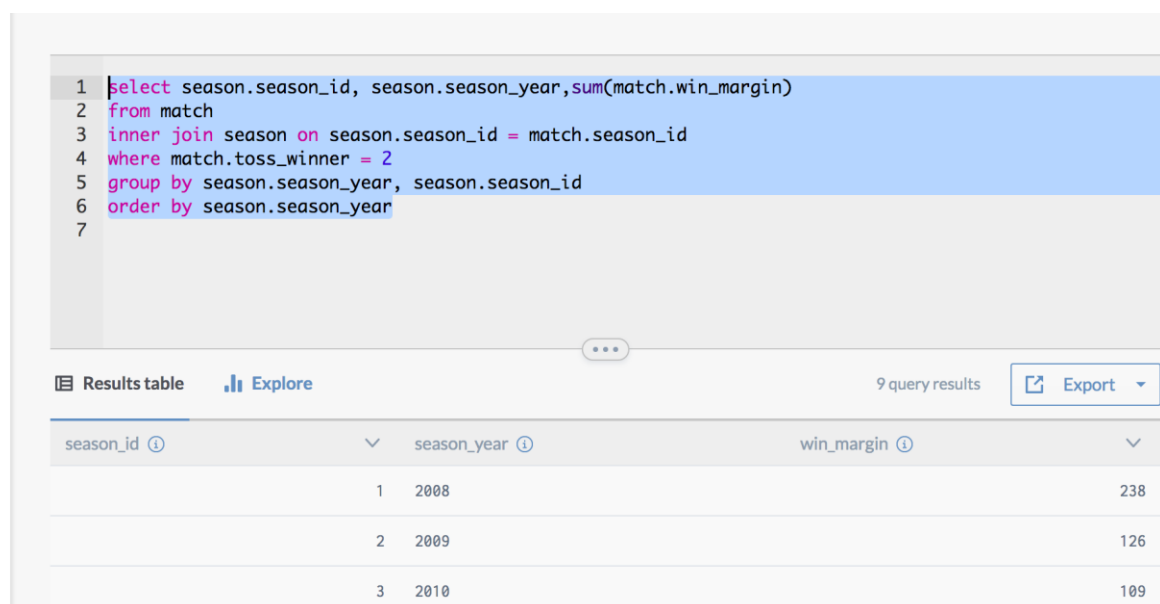
```

	match_id	season_id	season_year	runs_scored
0	335987	1	2008	16799
1	392230	2	2009	15372
2	419111	3	2010	17753
3	501203	4	2011	19916
4	548311	5	2012	21309
5	598003	6	2013	21483
6	729284	7	2014	17943
7	829710	8	2015	17427
8	980906	9	2016	17959

I have plotted graph by using the Matplotlib

Extra runs are very important for analysis to decide whether we will win or loss.

Winning Margin by season year:



The screenshot shows a SQL query editor with the following code:

```

1 select season.season_id, season.season_year, sum(match.win_margin)
2 from match
3 inner join season on season.season_id = match.season_id
4 where match.toss_winner = 2
5 group by season.season_year, season.season_id
6 order by season.season_year
7

```

Below the query editor, the results table is displayed with the following data:

season_id	season_year	win_margin
1	2008	238
2	2009	126
3	2010	109

Grouping and Joining tables:

I have uploaded all the Sql scripts and result in Notebook

Venue scores:

```

1 select venue.venue_name,
2 sum(batsman_scored.batsman_scored.runs_scored),
3 count(distinct match.match_id) match
4 from batsman_scored
5 inner join match on match.match_id = batsman_scored.match_id
6 inner join venue on venue.venue_id = match.venue_id
7 group by venue.venue_name
8 order by 2 asc;

```

Results table [Explore](#) 35 query results [Export](#)

venue_name	runs_scored	match
Punjab Cricket Association Stadium	10421	35
Rajiv Gandhi International Stadium	11748	41
Nehru Stadium	1286	5
Wankhede Stadium	14436	49
MA Chidambaram Stadium	14471	48
Saurashtra Cricket Association Stadium	1479	5

After result, I have downloaded the result table and embed in the python notebook.

Analysis of Extra runs season wise from 2008 to 2015:

```

1 select match.match_id, season.season_id, season.season_year,
2 sum(extra_runs.extra_runs)
3 from extra_runs
4 inner join match on match.match_id = extra_runs.match_id
5 inner join season on season.season_id = match.season_id
6 group by season.season_id, season.season_year
7 order by season_year

```

Results table [Explore](#) 9 query results [Export](#)

Field of aggregated query neither grouped or aggregated: (line 1, column 8) [View SQL tutorial](#)

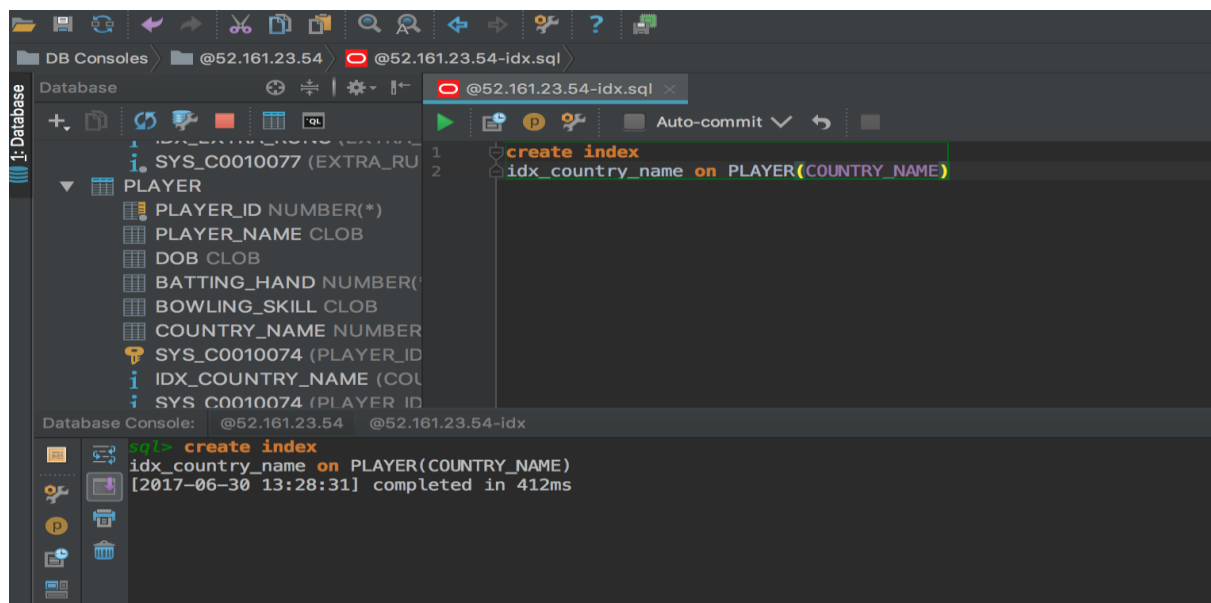
match_id	season_id	season_year	extra_runs
335987	1	2008	1128
392230	2	2009	977
419111	3	2010	1129
501203	4	2011	1226
548311	5	2012	1131

Indexing on the Player table:

```

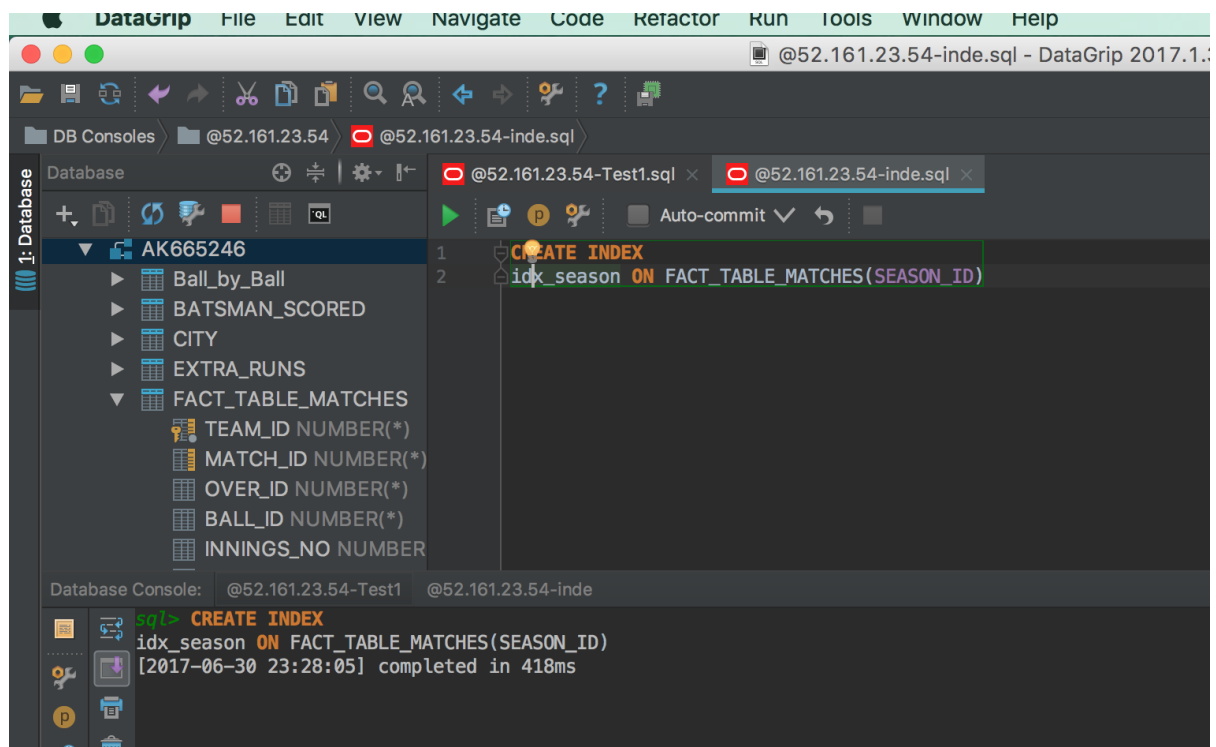
create index
idx_Player_name on fact_table_matches(player.player_name)

```

Indexing on the fact table:

We have to access the some of the records quickly from the fact tables.



Hadoop Scripts or Python scripts:

I have upload some of the CSV files in Hive to create the tables.

Then we would build the queries to understand the some of the questions.

Hive table:

Extra Runs table:

The screenshot displays a web-based interface for executing Hive queries. On the left, a sidebar lists various databases including 'default', 'avg_mileage', 'building_raw', 'buildings', 'drivermileage', 'extra_runs', 'geolocation', 'hvac', 'hvac_building', 'hvac_raw', 'hvac_temperatures', 'foodmart', and 'xademo'. The main area contains a SQL query editor with the text: `1 select * from extra_runs;`. Below the editor are buttons for 'Execute', 'Explain', 'Save as...', and 'New Worksheet'. The 'Execute' button has been clicked, resulting in a 'Query Process Results (Status: SUCCEEDED)' message. Below this, a 'Results' tab is active, showing a table with 5 columns: 'extra_runs.extra_runs_id', 'extra_runs.match_id', 'extra_runs.over_id', 'extra_runs.ball_id', and 'extra_runs.extra_type_id'. The first row of data is visible, showing values 1, 335987, 1, 1, and 1 respectively. Navigation buttons 'previous' and 'next' are present on the right side of the results table.

extra_runs.extra_runs_id	extra_runs.match_id	extra_runs.over_id	extra_runs.ball_id	extra_runs.extra_type_id
1	335987	1	1	1

Extra runs per innings 1 and 2

The image displays two screenshots of a web-based SQL interface, likely Hive or a similar data warehouse tool. Both screenshots show a query being executed to calculate the sum of extra runs for a specific innings.

Top Screenshot:

- Query:**

```
1 select sum(extra_runs.extra_runs) from extra_runs
2 where extra_runs.innings_no = 2;
```
- Buttons:** Execute, Explain, Save as...
- Query Process Results (Status: SUCCEEDED):**
 - Logs:** Filter columns... (previous)
 - Results:**

_c0
4584

Bottom Screenshot:

- Query:**

```
1 select sum(extra_runs.extra_runs) from extra_runs
2 where extra_runs.innings_no = 1;
```
- Buttons:** Execute, Explain, Save as..., New Worksheet
- Query Process Results (Status: SUCCEEDED):**
 - Logs:** Filter columns... (previous) (next)
 - Results:**

_c0
4924

The interface includes a sidebar with a search bar and a list of databases: default, avg_mileage, building_raw, buildings, drivermileage, extra_runs, geolocation, hvac, hvac_building, hvac_raw, hvac_temperatures, foodmart, and xademo. The top screenshot also shows a browser address bar with the URL 13.78.182.240:8080/#/main/views/HIVE/1.5.0/AUTO_HIVE_INSTANCE.

We can do the same thing here by using the SQL or Pig scripts.

PIG is data manipulation languages. So we can use to crunch the data and find out the insights from data.

Player analysis using pandas:

- Young Players- Players DOB > 1990
- Old players- Player DOB < 1980

We can do the same thing with python also, we can visualize the data by using different libraries:

Note: Updated all the codes in note book.

What I learned from the Project?

The most important part I learned from this project is how to design the data warehouse Project. Still I have to use some of the ETL processing to transform the data with industry standards.

I learned how to integrate the Sql scripts to python and used the pandas and Hive and Spark or Pig on top of that resulted data.

Shortcomings:

A lot of scope to add more columns and the team details. Auction table and the details of salary of each player but due to the time constraints we cannot build the data warehouse with all tables.

Conclusion:

IPL is one of the most-attended cricket league in the world and as an ardent fan of cricket. I thought this would be good idea to build the data warehouse for the IPL data. before designing the warehouse, all the data was scattered in different files. so I decided to collect data and design the warehouse. Now all data under the one platform and accessible to user and integrated well. Where anyone can use the data and find out the insights from the data.

Primary goal of the project to see the trends of matches season wise or venue wise. Each Year, IPL managing body conduct the auction so might be before going to auction we have some statistics of player or season or country. so we can use the data to make any decisions.

Next goal:

Next goal would be use some of the modeling techniques to find the particular team would win or not by using the machine learning algorithms.