COL362 Course Project: Data Suggestions

Anshul Basia 2014MT60502
Sahil Aggarwal 2014MT10607

March 14, 2017

1 Project Description

Our project is a suggestion mechanism for anyone who wants to spend some quality time and does not want to regret it later. Well, that depends on the person, but this project will reduce the probability of regretting that decision by a significant number. This project will present him with different options to explore along with different statistics and filter and then user can decide based on his interests and data given to him. This project is **highly scalable**, that is, there are hundreds of things which anyone may want to do but given the time constraint, we were able to complete few of them and we can extend this project later.

2 List of Entities and Attributes

Entity	**Attributes**
Anime	Anime_id, Name, Genre, Type, Rating, Members
Rating	User_id, Anime_id, Rating
Matches	Id, Season, City, Date, Team1, Team2, Toss_winner, Toss_decision, Result, dl_applied, Winner, Win_by_runs, Win_by_wickets, Player_of_match, Venue, Umpire1, Umpire2, Umpire3
Deliveries	Match_id, Inning, Batting_team, Bowling_team, over, ball, batsman, non_striker, bowler, is_super_over, wide_runs, bye_runs, legbye_runs, noball_runs, penalty_runs , batsman_runs, extra_runs, total_runs, player_dismissed, dismissal_kind, fielder
Movies	Color, Director_name, num_critic_for_reviews, Duration, Director_facebook_likes, Actor3_facebook_likes, Actor1_name, Actor1_facebook_likes, Gross, Genres, Actor2_name, Movie_title, num_of_voted_users, cast_total_facebook_likes, actor3_name, Facenumber_in_priority, plot_keywords, Movie_IMDB_link, num_user_for_reviews, Language, Country, Content_Rating, Budget, Title_year, Actor2_facebook_likes, IMDB_score, Aspect_ratio, movie_facebook_likes

3 Data Sources

- Anime Recommendation Database:
  https://www.kaggle.com/CooperUnion/anime-recommendations-database
- IPL Matches database:
  https://www.kaggle.com/manasgarg/ipl
- Movie's database:
  https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset

4 Raw Data Statistics

Relation	**No._of_tuples**	**Time_to_load**	**Raw Dataset Size**
Anime	5336	122.139 ms	526 KB
Rating	7813737	12872.047 ms	111 MB
Matches	577	67.339 ms	104 KB
Deliveries	136598	725.868 ms	14 MB
Movies	5038	116.364 ms	1.8 MB

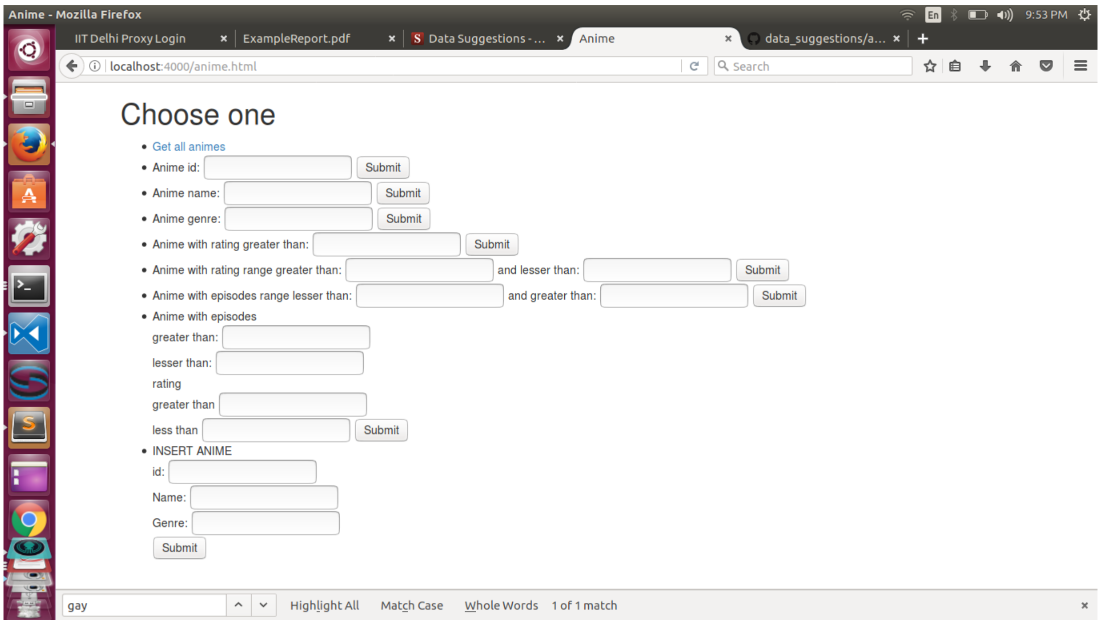
5 Functionality and Working

5.1 User's view

At the home screen, user will have three options to navigate:

- Anime
- IPL
- Movies

User can select any one of them and filter according to his preferences and interests. Some screenshots from the application:

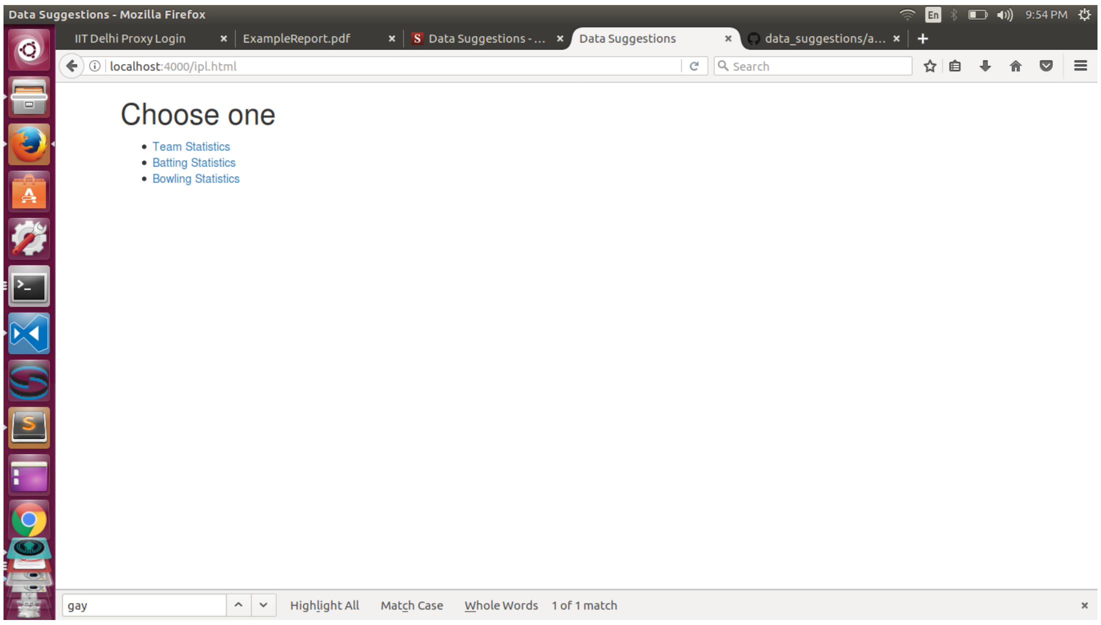


Anime - Mozilla Firefox

localhost:4000/anime.html

Choose one

- Get all animes
- Anime id: Submit
- Anime name: Submit
- Anime genre: Submit
- Anime with rating greater than: Submit
- Anime with rating range greater than: and lesser than: Submit
- Anime with episodes range lesser than: and greater than: Submit
- Anime with episodes
  greater than:
  lesser than:
  rating
  greater than
  less than Submit
- INSERT ANIME
  id:
  Name:
  Genre:
  Submit

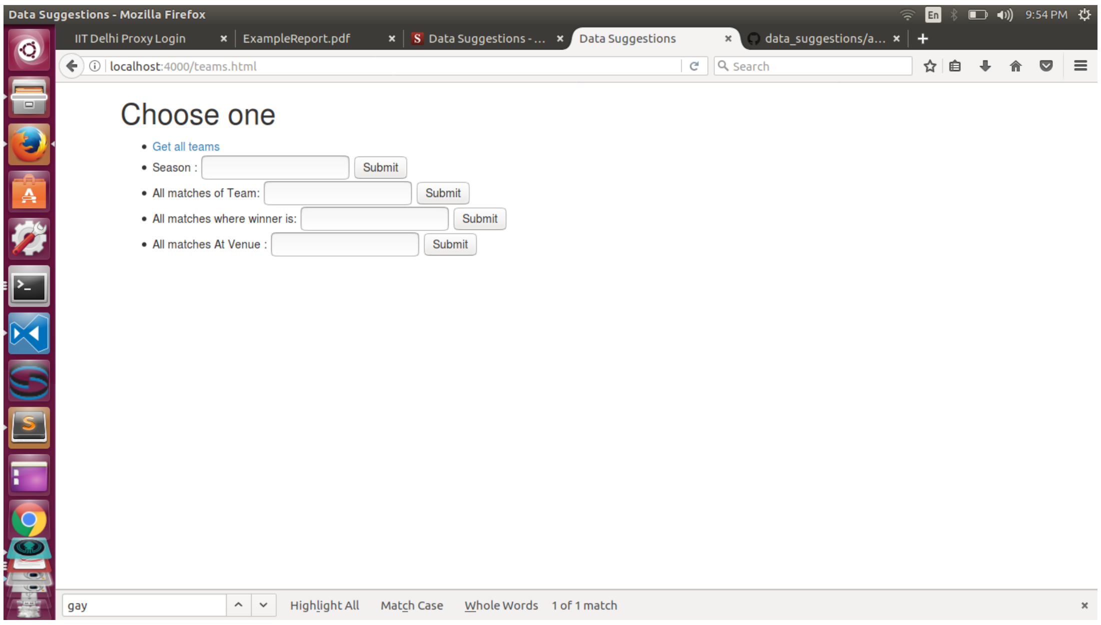


Data Suggestions - Mozilla Firefox

localhost:4000/ipl.html

Choose one

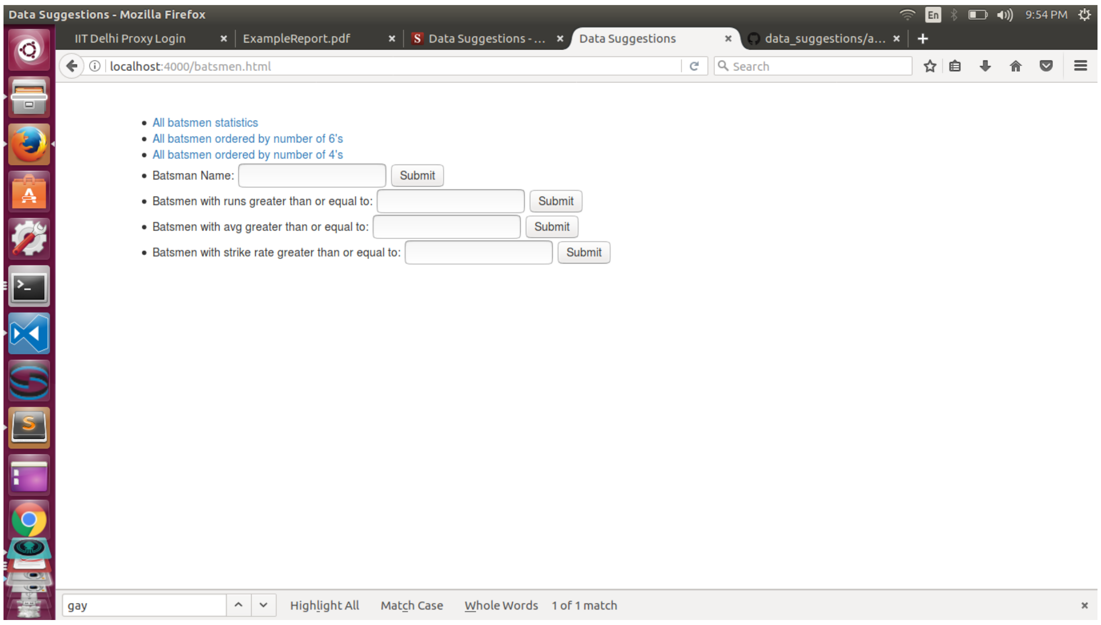
- Team Statistics
- Batting Statistics
- Bowling Statistics


Data Suggestions - Mozilla Firefox

localhost:4000/teams.html

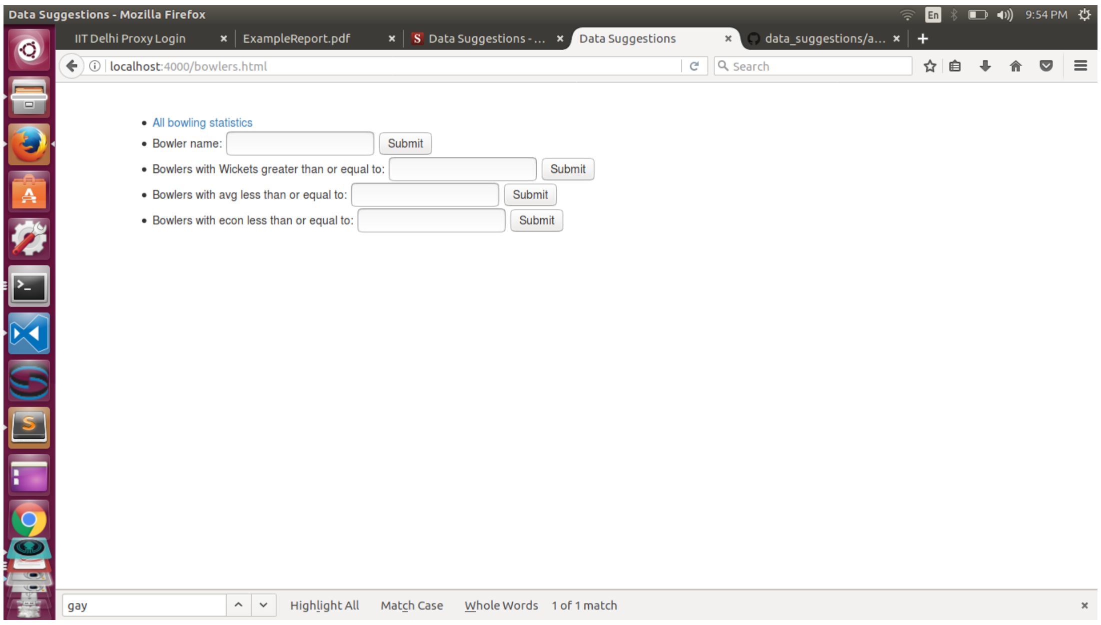
Choose one

- Get all teams
- Season : Submit
- All matches of Team: Submit
- All matches where winner is: Submit
- All matches At Venue : Submit

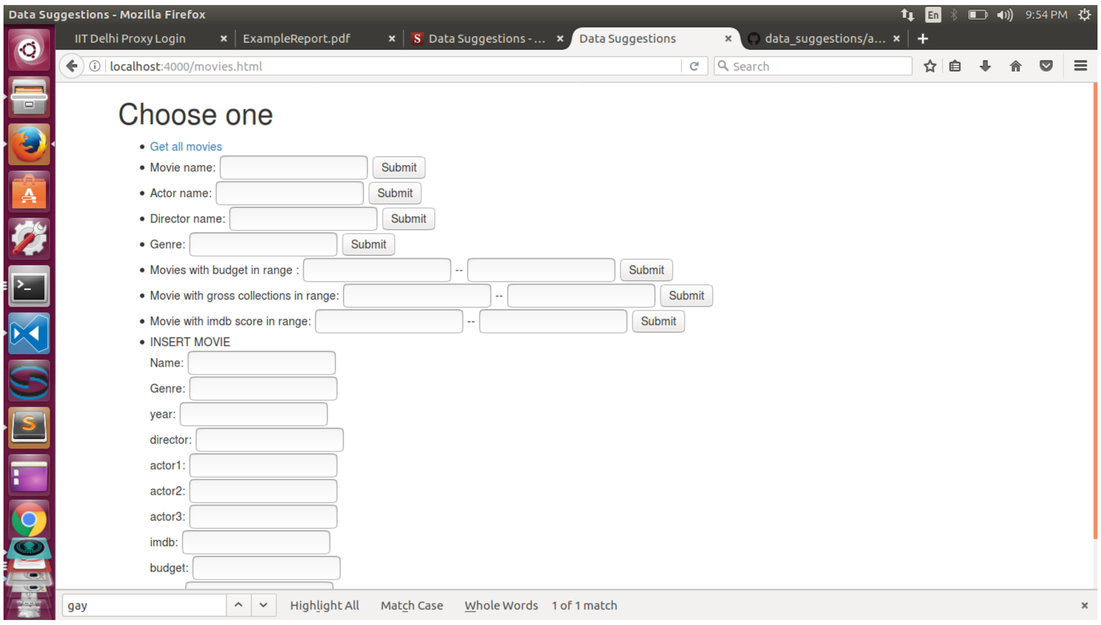

Data Suggestions - Mozilla Firefox

localhost:4000/batsmen.html

- All batsmen statistics
- All batsmen ordered by number of 6's
- All batsmen ordered by number of 4's
- Batsman Name: Submit
- Batsmen with runs greater than or equal to: Submit
- Batsmen with avg greater than or equal to: Submit
- Batsmen with strike rate greater than or equal to: Submit



- All bowling statistics
- Bowler name: Submit
- Bowlers with Wickets greater than or equal to: Submit
- Bowlers with avg less than or equal to: Submit
- Bowlers with econ less than or equal to: Submit



Choose one

- Get all movies
- Movie name: Submit
- Actor name: Submit
- Director name: Submit
- Genre: Submit
- Movies with budget in range : -- Submit
- Movie with gross collections in range: -- Submit
- Movie with imdb score in range: -- Submit
- INSERT MOVIE

  Name:

  Genre:

  year:

  director:

  actor1:

  actor2:

  actor3:

  imdb:

  budget:

5.2 Special Database features

- **Views**

  Extremely Complicated database queries have been made simple using materialized views while pre-execution of database file.

  1. Batsman

```
DROP MATERIALIZED VIEW IF EXISTS BATSMEN;
CREATE MATERIALIZED VIEW BATSMEN AS
SELECT
  batsman as name,
  count(distinct match_id) as matches_played,
  sum(batsman_runs) as runs_scored,
  count(over) as balls_faced,
  ROUND(sum(batsman_runs*100)::numeric /count(over)::
    numeric,2) as strike_rate,
  ROUND(sum(batsman_runs)::numeric/(count(player_dismissed
     = batsman)+1)::numeric,2) as avg
  from deliveries group by batsman
  order by runs_scored desc;

```

Analysis:

```
DROP MATERIALIZED VIEW
Time: 0.173 ms
SELECT 436
Time: 1421.944 ms

```

2. Bowler

```
DROP MATERIALIZED VIEW IF EXISTS BOWLERS;
CREATE MATERIALIZED VIEW BOWLERS AS
SELECT
  bowler as name,
  count(distinct match_id) as matches_played,
  sum(batsman_runs) as runs,
  count(over) as balls,
  ROUND(sum(batsman_runs*6)::numeric /count(over)::numeric
    ,2) as econ,
  count(player_dismissed = batsman AND bowler=bowler AND
    dismissal_kind != 'run out') as wickets,
  ROUND(count(over)::numeric/(count(player_dismissed =
    batsman AND bowler=bowler)+0.0001)::numeric,2) as avg
  from deliveries group by bowler
  order by wickets desc;

```

Analysis:

```
DROP MATERIALIZED VIEW
Time: 0.271 ms
SELECT 334
Time: 763.565 ms
```

3. Team Statistics

```
DROP VIEW IF EXISTS team_stats;
CREATE VIEW team_stats as
SELECT winners.name, table1.matches+table2.matches as
    matches, winners.won as won, table4.matches+table3.
    matches - winners.won as lost,table1.matches+table2.
    matches-table4.matches-table3.matches as draw from
(SELECT team1 as name,count(id) as matches from matches
    group by team1) as table1,
(SELECT team2 as name,count(id) as matches from matches
    group by team2) as table2,
(SELECT team1 as name,count(id) as matches from matches
    where result = 'normal' group by team1) as table3,
(SELECT team2 as name,count(id) as matches from matches
    where result = 'normal' group by team2) as table4,
(SELECT winner as name,count(id) as won from matches group
     by winner) as winners
WHERE winners.name = table1.name and table1.name = table2.
    name and table1.name = table3.name and table4.name =
    table1.name order by name ;

```

Analysis:

```
DROP VIEW
Time: 0.218 ms
CREATE VIEW
Time: 11.210 ms
```

4. Season Statistics

```
DROP VIEW IF EXISTS season_stats;
CREATE VIEW season_stats as
```

```
SELECT winners.name as name,winners.season as season,
    table1.matches+table2.matches as matches, winners.won
    as won, table4.matches+table3.matches - winners.won as
     lost ,table1.matches+table2.matches-table4.matches-
    table3.matches as draw from
(SELECT team1 as name, season, count(id) as matches from
    matches group by team1,season) as table1,
(SELECT team2 as name, season, count(id) as matches from
    matches group by team2,season) as table2,
(SELECT team1 as name, season, count(id) as matches from
    matches where result = 'normal' group by team1,season)
     as table3,
(SELECT team2 as name, season, count(id) as matches from
    matches where result = 'normal' group by team2,season)
     as table4,
(SELECT winner as name,season, count(id) as won from
    matches group by winner,season) as winners
WHERE winners.name = table1.name and table1.name = table2.
    name and table1.name = table3.name and table4.name =
    table1.name
 and winners.season = table1.season and table1.season =
    table2.season and table1.season = table3.season and
    table4.season = table1.season order by name ;

```

Analysis:

```
DROP VIEW
Time: 0.237 ms
CREATE VIEW
Time: 10.279 ms
```

5. Six and Four

```
CREATE VIEW SIX AS
SELECT batsman, count(over) as num_six
from deliveries where batsman_runs = 6
group by batsman order by num_six;

CREATE VIEW FOUR AS
SELECT batsman, count(over) as num_fours
from deliveries where batsman_runs = 4
group by batsman order by num_fours;

```

Analysis:

```
CREATE VIEW
Time: 11.143 ms
CREATE VIEW
Time: 77.530 ms
```

6. Batsman Matches

```
DROP MATERIALIZED VIEW IF EXISTS BATSMEN_MATCHES;
CREATE MATERIALIZED VIEW BATSMEN_MATCHES AS
SELECT match_id, batsman, sum(batsman_runs) as runs, count
    (over) as balls
FROM deliveries group by match_id,batsman order by runs;


```

Analysis:

```
DROP MATERIALIZED VIEW
Time: 0.207 ms
SELECT 8617
Time: 176.867 ms
```

7. Bowler Matches

```
DROP MATERIALIZED VIEW IF EXISTS BOWLERS_MATCHES ;
CREATE MATERIALIZED VIEW BOWLERS_MATCHES AS
SELECT match_id , bowler , sum(batsman_runs) as runs , count(
    player_dismissed = batsman AND bowler=bowler AND
    dismissal_kind != 'run out') as wickets
FROM deliveries group by match_id ,bowler order by wickets
    desc;


```

Analysis:

```
DROP MATERIALIZED VIEW
Time: 0.237 ms
SELECT 6866
Time: 187.976 ms
```

- **Indexes**
  There is a huge number of tuples in relations 'Deliveries' and 'rating', to speed up the query processing. We just used a standard B-tree index in our creation of tables. For example:

```
CREATE UNIQUE INDEX title_idx ON Matches (id);
CREATE INDEX
Time: 96.080 ms


CREATE UNIQUE INDEX anime_idx ON anime (anime_id);
CREATE INDEX
Time: 94.426 ms
```

- **Constraints**
  We have primary keys for each of the table and apart from that, several foreign keys have been used like, for relation 'rating', anime_id is the foreign key and similarly between deliveries and matches.

- **Well separated and modular front-end and backend** We used a model view controller called NodeJs for the backend and a combination of PHP and Angular for the front end. Changes are easy to implement and code is scalable along with different types of datasets.

5.3 List of APIs

```
//Anime APIs
*To get all animes:
API: ->      ('/api/anime', db.getAnime);

*To get an anime by id:
API: ->      ('/api/anime/:id', db.getAnimebyid);

*To get an anime by name:
API: ->      ('/api/anime/name/:name', db.getAnimebyname);

*To get an anime by genre:
API: ->      ('/api/anime/genre/:genre', db.getAnimebyGenre);

*To get an anime according to rating threshold:
API: ->      ('/api/anime/rating/:high', db.getAnimebyrating);

*To get an anime by range of rating:
API: ->      ('/api/anime/rating/:high/:low', db.
    getAnimebyratingrange);

*To get an anime by range of episodes:
API: ->      ('/api/anime/episodes/:high/:low', db.
    getAnimebyepisodes);

To get an anime by both rating and number of episodes' range:
API: ->      ('/api/anime/episodes_rating/:highe/:lowe/:highr/:lowr
    ', db.getAnimebyepisodes_rating);

To insert a new anime with id, name and genre:
API: ->      ('/api/insert_anime/:id/:name/:genre', db.createanime);


//Movies' APIs
*To get all the movies:
API: ->      ('/api/movies', db.getMovies);

*To get movies by name:
API: ->      ('/api/movies/name/:name', db.getMoviebyname);

*To get movies by your favourite director:
API: ->      ('/api/movies/director/:director', db.
    getMoviebyDirector);

*To get movies by your favourite actor:
API: ->      ('/api/movies/actor/:actor', db.getMoviebyActor);

*To get movies by your favourite genre:
API: ->      ('/api/movies/genre/:genre', db.getMoviesbyGenre);

*To get movies by budget:
API: ->      ('/api/movies/budget/:low/:high', db.getMoviesbyBudget)
    ;

*To get movies by gross range:
API: ->      ('/api/movies/gross/:low/:high', db.getMoviesbyGross);

*To get movies by ratings'range:
API: ->      ('/api/movies/rating/:low/:high', db.getMoviesbyRating)
    ;

```

```
API: ->     ('/api/insert_movie/:name/:genre/:year/:director/:
    actor1/:actor2/:actor3/:imdb/:budget/:gross', db.createMovie);

/*IPL api's*/

*To get all the teams:
API: ->     ('/api/ipl/teams', db.getTeams);

*To get all the batsmen:
API: ->     ('/api/ipl/batsmen', db.getBatsmen);

*To get batsmen bynumber of sixes:
API: ->     ('/api/ipl/batsmen/six', db.getBatsmenbySix);

*To get batsmen by number of fours:
API: ->     ('/api/ipl/batsmen/four', db.getBatsmenbyFour);

*To get all the bowlers:
API: ->     ('/api/ipl/bowlers', db.getBowlers);

*To get all the teams by seasson:
API: ->     ('/api/ipl/season/:season', db.getTeamsBySeason);

*To get batsman by name:
API: ->     ('/api/ipl/batsman/:name', db.getBatsman);

*To get bowler by name:
API: ->     ('/api/ipl/bowler/:name', db.getBowler);

*To get batsman by runs he scored:
API: ->     ('/api/ipl/batsman/runs/:runs', db.getBatsmenByRuns);

*To get bowler by wickets he took:
API: ->     ('/api/ipl/bowler/wickets/:wickets', db.
    getBowlersByWickets);

*To get batsman by average runs:
API: ->     ('/api/ipl/batsman/avg/:avg', db.getBatsmenByAvg);

*To get batsman by strike rates:
API: ->     ('/api/ipl/batsman/strike_rate/:strike_rate', db.
    getBatsmenByStrikeRate);

*To get bowler by avg runs:
API: ->     ('/api/ipl/bowler/avg/:avg', db.getBowlersByAvg);

*To get bowler by economy:
API: ->     ('/api/ipl/bowler/econ/:econ', db.getBowlersByEcon);

*To get matches of your favourite team:
API: ->     ('/api/ipl/teams/matches/:name', db.getMatchesByTeam);

*To get matches according to your venue:
API: ->     ('/api/ipl/venue/matches/:name', db.getMatchesByVenue);

*To get matches by winners:
API: ->     ('/api/ipl/winner/matches/:winner', db.
    getMatchesByWinners);
```

5.4 List of Queries

- **Anime**:

  1. select * from anime
  2. select * from anime where anime_id = (1)', animeID
  3. select * from anime where name = (1)", req.params.name
  4. select * from anime where **strpos**(genre,(1)) $\geq$ 0", req.params.genre
  5. select * from anime where rating $\geq$ = (1) order by rating", req.params.high
  6. select * from anime where rating $\geq$ (1) AND rating $\leq$ = (2) order by rating", [req.params.high, req.params.low]
  7. select * from anime where episodes $\geq$ (1) AND episodes $\leq$ = (2) order by episodes", [req.params.high, req.params.low]
  8. select * from anime where episodes $\geq$ (1) AND episodes $\leq$ = (2) AND rating $\geq$ = (3) AND rating $\leq$ = (4)", [req.params.highe, req.params.lowe, req.params.highr, req.params.lowr]
  9. insert into anime(anime_id, name, genre)" + "values((1),(2),(3))", [req.params.id, req.params.name, req.params.genre]

- **IPL**:

  1. select * from team_stats.
  2. select * from batsmen
  3. SELECT * FROM BATSMEN **JOIN** SIX ON batsmen.name = six.batsman order by num_six desc
  4. SELECT * FROM BATSMEN **INNER JOIN** Four ON batsmen.name = four.batsman order by num_fours desc
  5. select * from bowlers
  6. select * from batsmen where name = (1)", req.params.name
  7. select * from bowlers where name = (1)", req.params.name
  8. select * from season_stats where season = (1)", season
  9. select * from batsmen where runs_scored $\geq$= (1)", req.params.runs
  10. select * from bowlers where wickets $\geq$= (1)", req.params.wickets
  11. select * from batsmen where avg $\geq$= (1) order by avg desc", req.params.avg
  12. select * from batsmen where strike_rate $\geq$= (1) order by strike_rate desc", req.params.strike_rate
  13. select * from bowlers where avg $\leq$= (1) and wickets $\geq$= 10 order by avg", req.params.avg
  14. select * from bowlers where econ $\leq$= (1) and wickets $\geq$= 10 order by econ", req.params.econ
  15. select * from matches where team1 = (1) or team2 = (1) order by date", req.params.name

16. select * from matches where city = (1) order by date", req.params.name
17. select * from matches where winner = (1) order by date", req.params.winner

- **Movies**:

1. select * from movies order by title_year desc
2. select * from movies where **strpos**(movie_title,(1)) $\geq$ 0", req.params.name
3. select * from movies where director_name = (1)", req.params.director
4. select * from movies where actor_1_name = (1) or actor_2_name = (1) or actor_3_name = (1)", req.params.actor
5. select * from movies where **strpos**(genres,(1)) $\geq$ 0", req.params.genre)
6. select * from movies where budget $\geq$= (1) AND budget $\leq$= (2) order by budget desc", [req.params.low, req.params.high]
7. select * from movies where gross $\geq$= (1) AND gross $\leq$= (2) order by gross desc", [req.params.low, req.params.high]
8. select * from movies where imdb_score $\geq$= (1) AND imdb_score $\leq$= (2) order by imdb_score", [req.params.low, req.params.high]
9. insert into movies(movie_title, genres, title_year,director, actor_1_name, actor_2_name, actor_3_name, imdb_score, budget, gross)" + "values((1),(2),(3),(4),(5),(6),(7),(8),(9),(1)0)", [req.params.name, req.params.genre, req.params.year, req.params.director, req.params.actor1, req.params.actor2, req.params.actor3, req.params.imdb, req.params.budget, req.params.gross])

5.5 Query Times

- **Anime**

Query Number	**Average Running time**
1	88.824 ms
2	76.043 ms
3	37.145 ms
4	86.292 ms
5	107.363 ms
6	38.400 ms
7	25.815 ms
8	17.050 ms
9	48.433 ms

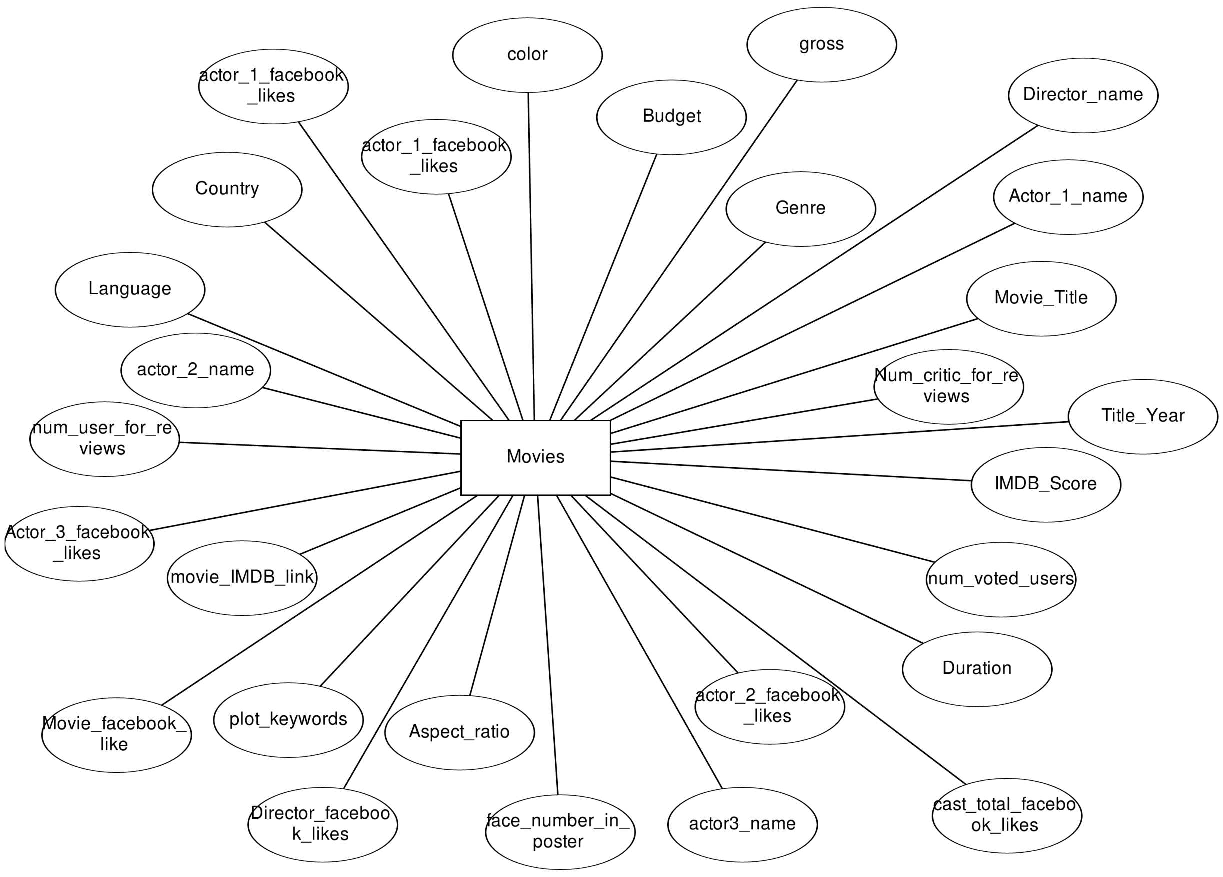
- **Movies**

Query Number	**Average Running time**
1	191.998 ms
2	39.703 ms
3	6.804 ms
4	15.422 ms
5	71.504 ms
6	38.076 ms
7	20.774 ms
8	9.457 ms
9	913.863 ms

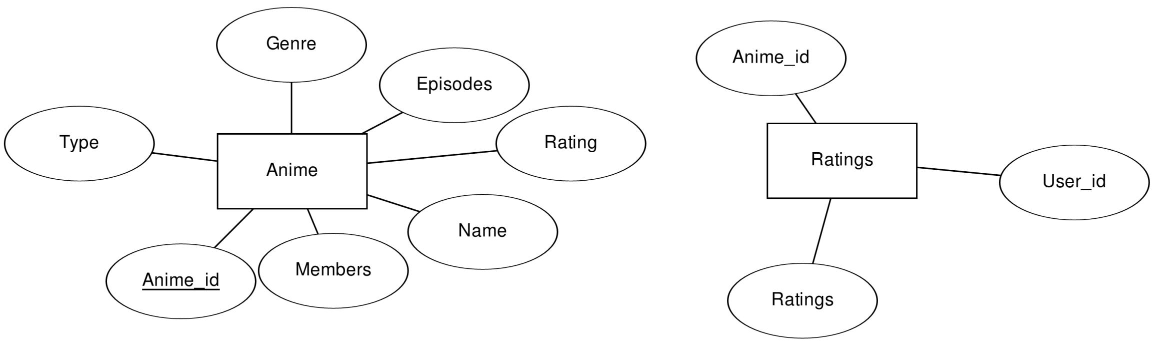
- **IPL**

Query Number	**Average Running time**
1	90.157 ms
2	23.610 ms
3	117.256 ms
4	104.003 ms
5	5.781 ms ms
6	31.650 ms
7	11.050 ms
8	20.576 ms
9	7.606 ms
10	8.076 ms
11	5.692 ms
12	7.869 ms
13	10.607 ms
14	9.002 ms
15	20.924 ms
16	6.073 ms
17	10.666 ms

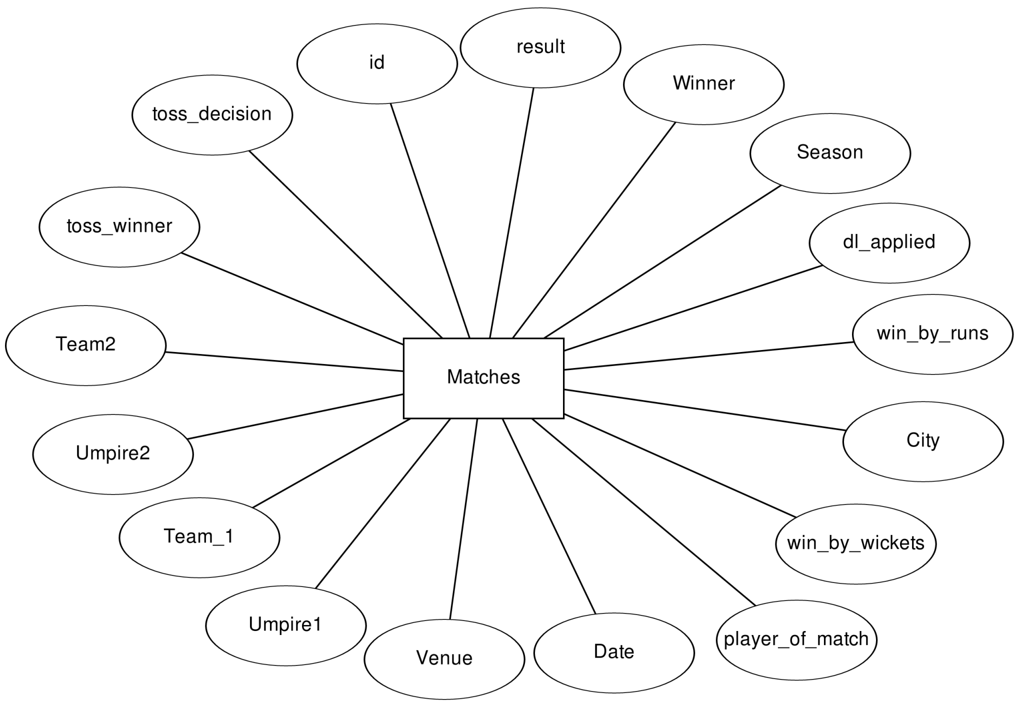
6 ERD Diagrams

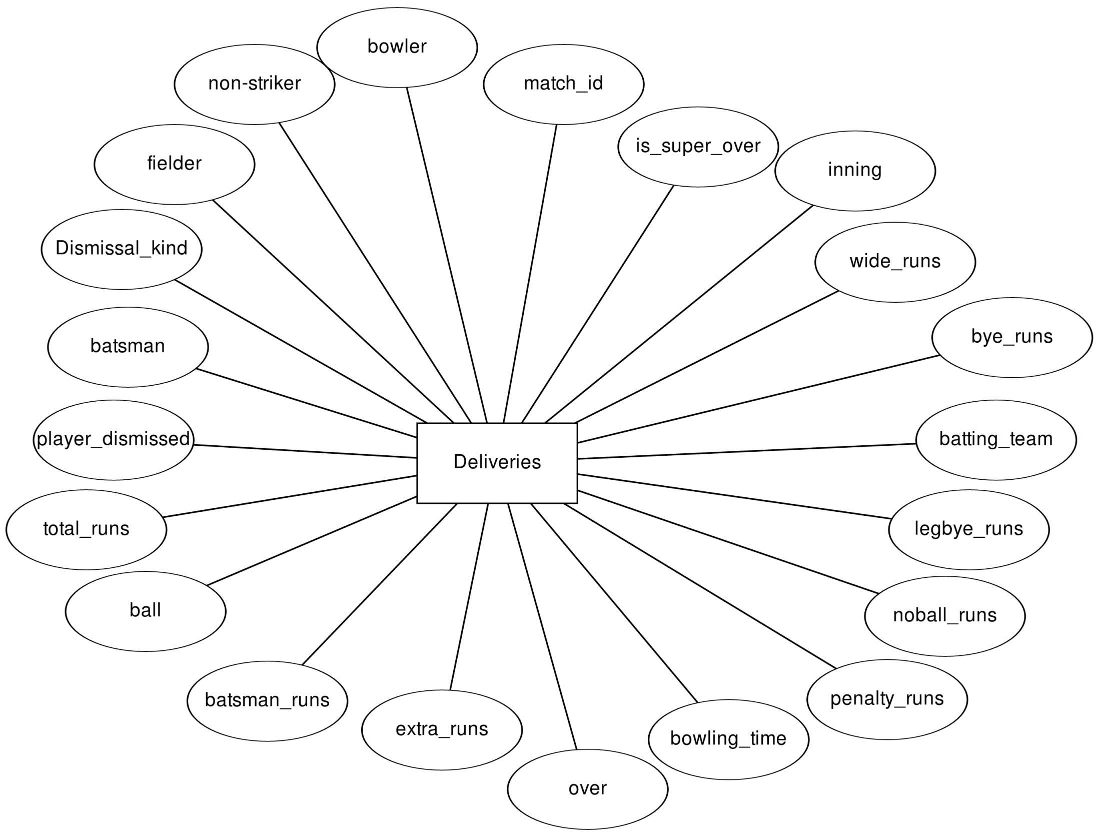


color
gross
actor_1_facebook_likes
Budget
Director_name
actor_1_facebook_likes
Country
Genre
Actor_1_name
Language
Movie_Title
actor_2_name
Num_critic_for_reviews
Title_Year
num_user_for_reviews
Movies
IMDB_Score
Actor_3_facebook_likes
movie_IMDB_link
num_voted_users
Duration
actor_2_facebook_likes
Movie_facebook_like
plot_keywords
Aspect_ratio
cast_total_facebook_likes
Director_facebook_likes
face_number_in_poster
actor3_name



Genre
Episodes
Type
Rating
Anime
Name
Anime_id
Members
Anime_id
Ratings
User_id
Ratings


Matches
id
result
Winner
Season
dl_applied
win_by_runs
City
win_by_wickets
player_of_match
Date
Venue
Umpire1
Team_1
Umpire2
Team2
toss_winner
toss_decision


Deliveries
bowler
match_id
is_super_over
inning
wide_runs
bye_runs
batting_team
legbye_runs
noball_runs
penalty_runs
bowling_time
over
extra_runs
batsman_runs
ball
total_runs
player_dismissed
batsman
Dismissal_kind
fielder
non-striker