

COL362 Course Project: Data Suggestions

Anshul Basia 2014MT60502
Sahil Aggarwal 2014MT10607

March 15, 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

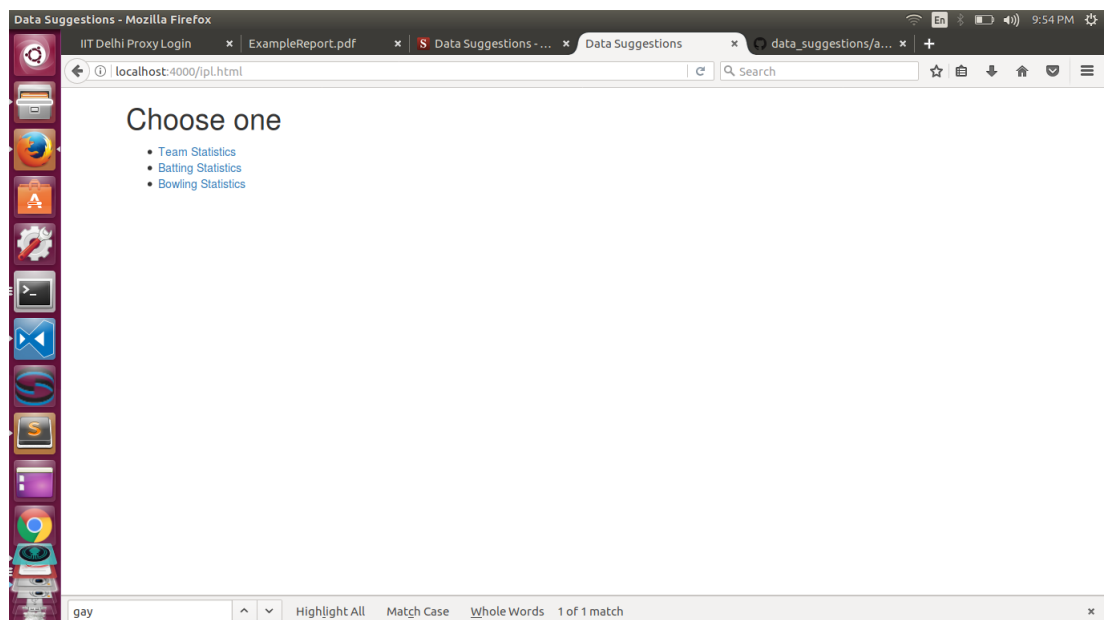
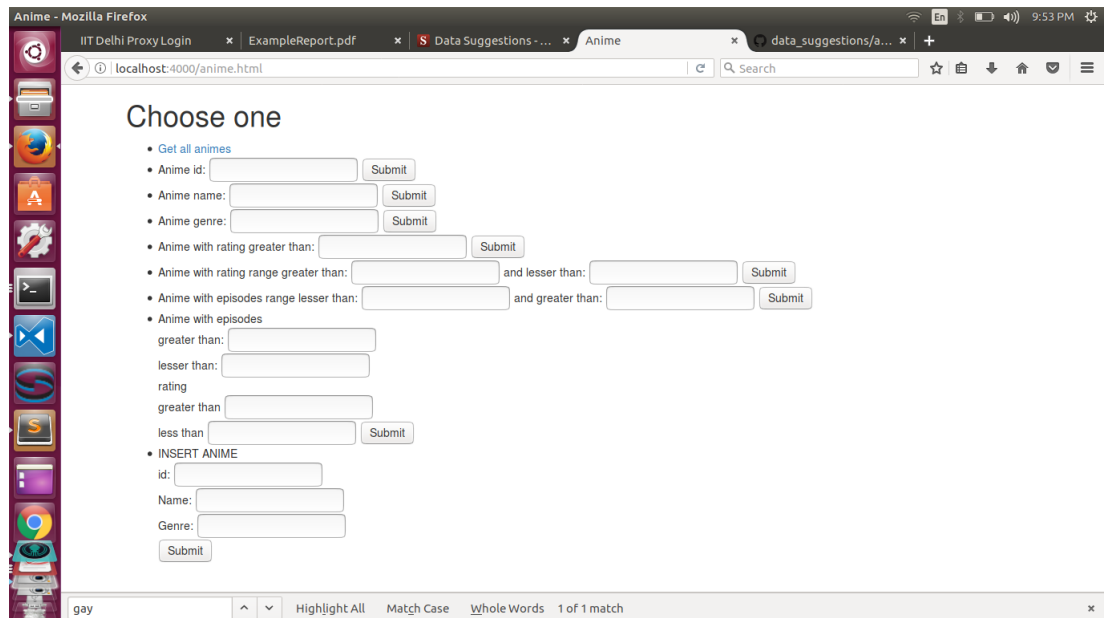
5 Functionality and Working

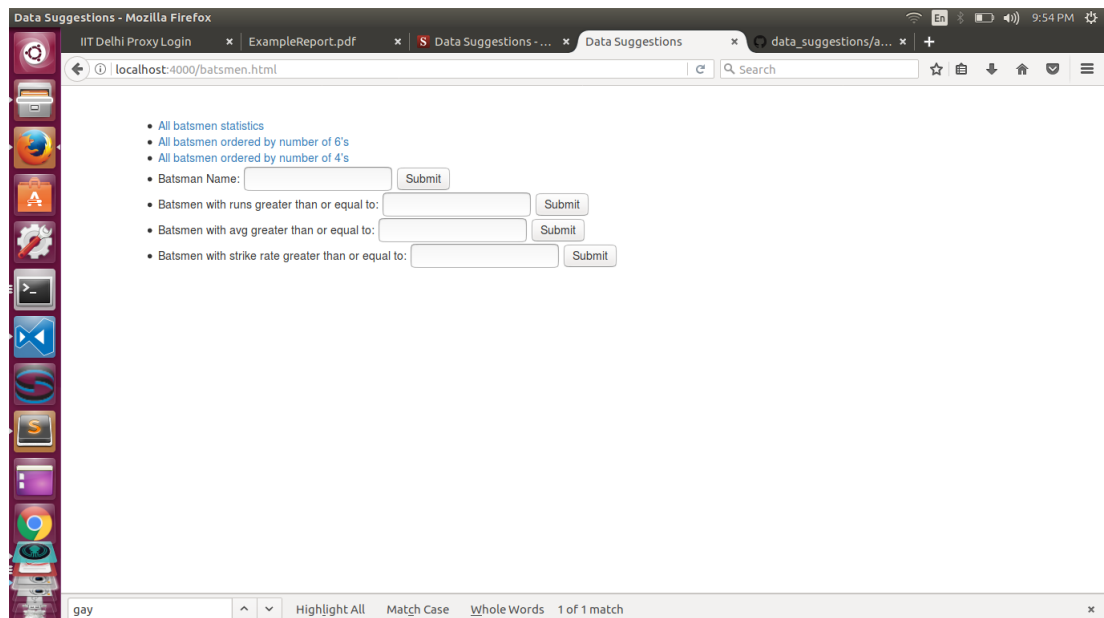
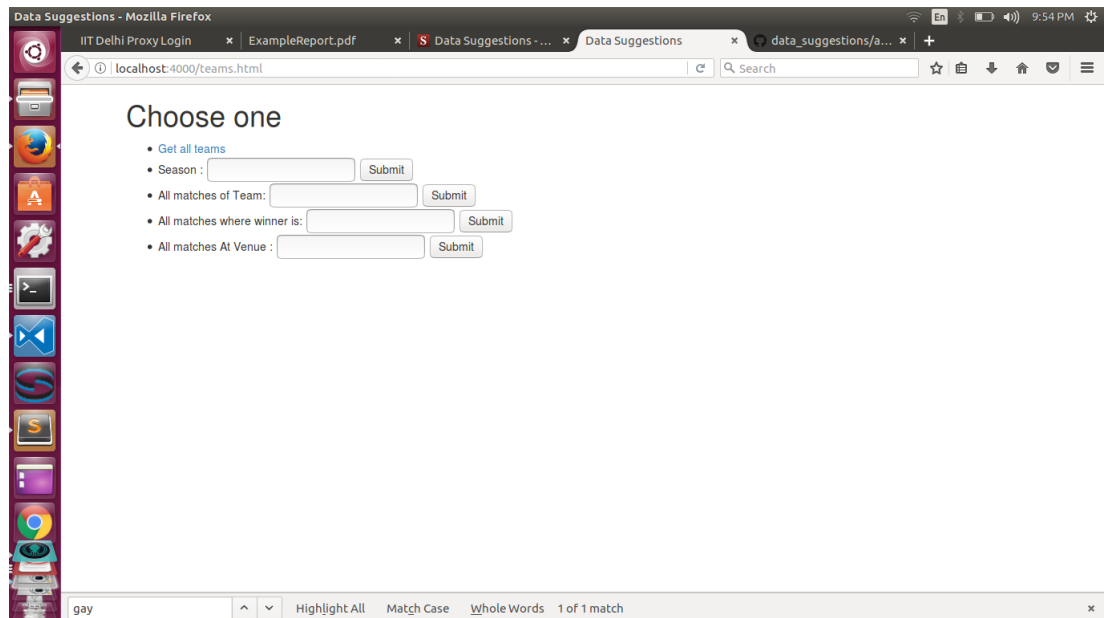
5.1 User's view

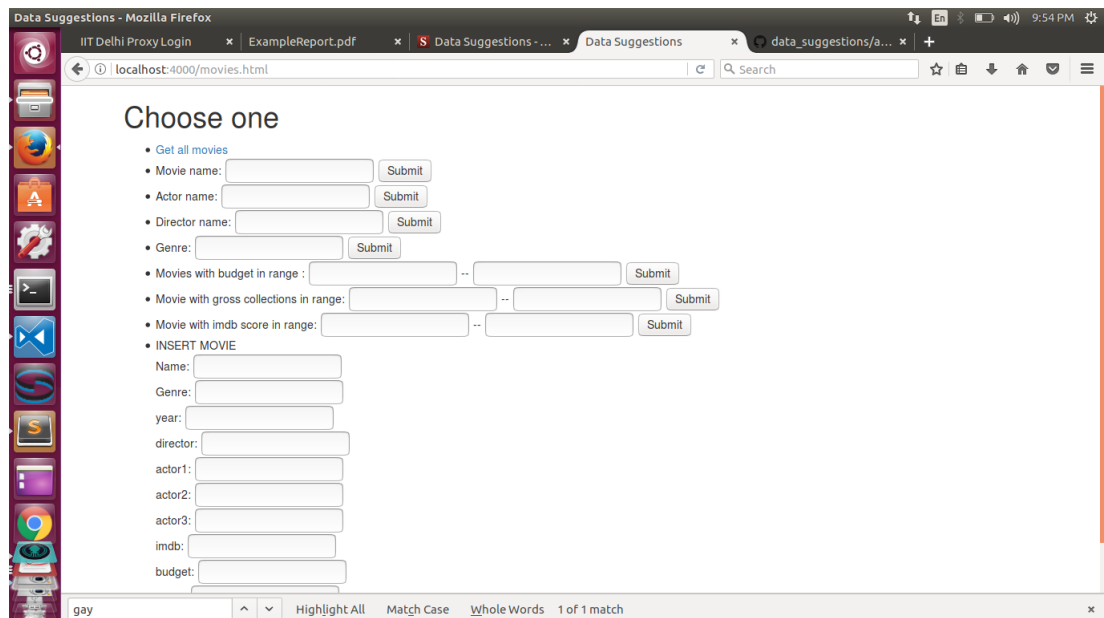
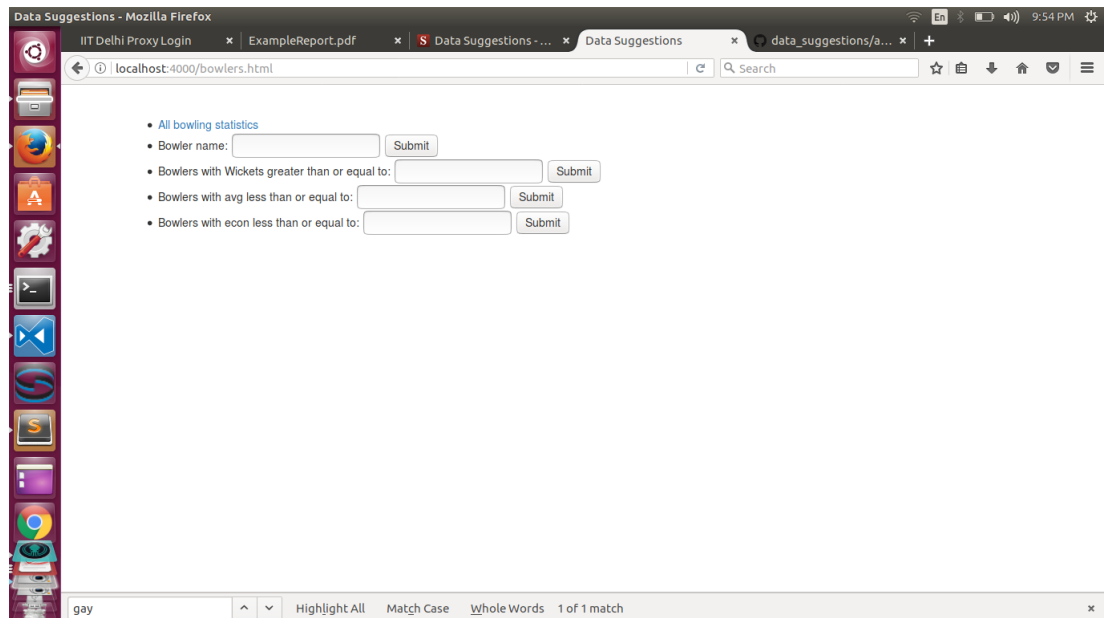
At the home screen, user will have three options to navigate:

- Anime
- IPL
- Movies

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







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

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

Analysis:

```
1 DROP MATERIALIZED VIEW
2 Time: 0.173 ms
3 SELECT 436
4 Time: 1421.944 ms
5
```

2. Bowler

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

Analysis:

```
1 DROP MATERIALIZED VIEW
2 Time: 0.271 ms
3 SELECT 334
4 Time: 763.565 ms
```

3. Team Statistics

```
1 DROP VIEW IF EXISTS team_stats;
2 CREATE VIEW team_stats as
3 SELECT winners.name, table1.matches+table2.matches as
4     matches, winners.won as won, table4.matches+table3.
5     matches - winners.won as lost ,table1.matches+table2.
6     matches-table4.matches-table3.matches as draw from
7 (SELECT team1 as name,count(id) as matches from matches
8     group by team1) as table1,
9 (SELECT team2 as name,count(id) as matches from matches
10    group by team2) as table2,
11 (SELECT team1 as name,count(id) as matches from matches
12    where result = 'normal' group by team1) as table3,
13 (SELECT team2 as name,count(id) as matches from matches
14    where result = 'normal' group by team2) as table4,
15 (SELECT winner as name,count(id) as won from matches group
16    by winner) as winners
17 WHERE winners.name = table1.name and table1.name = table2.
18     name and table1.name = table3.name and table4.name =
19     table1.name order by name ;
```

Analysis:

```
1 DROP VIEW
2 Time: 0.218 ms
3 CREATE VIEW
4 Time: 11.210 ms
```

4. Season Statistics

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

```

3 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
4 (SELECT team1 as name, season, count(id) as matches from
   matches group by team1,season) as table1 ,
5 (SELECT team2 as name, season, count(id) as matches from
   matches group by team2,season) as table2 ,
6 (SELECT team1 as name, season, count(id) as matches from
   matches where result = 'normal' group by team1,season)
   as table3 ,
7 (SELECT team2 as name, season, count(id) as matches from
   matches where result = 'normal' group by team2,season)
   as table4 ,
8 (SELECT winner as name,season, count(id) as won from
   matches group by winner,season) as winners
9 WHERE winners.name = table1.name and table1.name = table2.
   name and table1.name = table3.name and table4.name =
   table1.name
10 and winners.season = table1.season and table1.season =
   table2.season and table1.season = table3.season and
   table4.season = table1.season order by name ;
11

```

Analysis:

```

1 DROP VIEW
2 Time: 0.237 ms
3 CREATE VIEW
4 Time: 10.279 ms

```

5. Six and Four

```

1 CREATE VIEW SIX AS
2 SELECT batsman, count(over) as num_six
3 from deliveries where batsman_runs = 6
4 group by batsman order by num_six;
5
6 CREATE VIEW FOUR AS
7 SELECT batsman, count(over) as num_fours
8 from deliveries where batsman_runs = 4
9 group by batsman order by num_fours;
10

```

Analysis:

```

1 CREATE VIEW
2 Time: 11.143 ms
3 CREATE VIEW
4 Time: 77.530 ms

```

6. Batsman Matches

```

1 DROP MATERIALIZED VIEW IF EXISTS BATSMEN_MATCHES;
2 CREATE MATERIALIZED VIEW BATSMEN_MATCHES AS
3 SELECT match_id, batsman, sum(batsman_runs) as runs, count
   (over) as balls
4 FROM deliveries group by match_id, batsman order by runs;
5
6

```


Analysis:

```
1 DROP MATERIALIZED VIEW
2 Time: 0.207 ms
3 SELECT 8617
4 Time: 176.867 ms
```

7. Bowler Matches

```
1 DROP MATERIALIZED VIEW IF EXISTS BOWLERS_MATCHES ;
2 CREATE MATERIALIZED VIEW BOWLERS_MATCHES AS
3 SELECT match_id, bowler, sum(batsman_runs) as runs, count(
   player_dismissed = batsman AND bowler=bowler AND
   dismissal_kind != 'run out') as wickets
4 FROM deliveries group by match_id, bowler order by wickets
   desc;
5
6
```

Analysis:

```
1 DROP MATERIALIZED VIEW
2 Time: 0.237 ms
3 SELECT 6866
4 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:

```
1 CREATE UNIQUE INDEX title_idx ON Matches (id);
2 CREATE INDEX
3 Time: 96.080 ms
4
5
6 CREATE UNIQUE INDEX anime_idx ON anime (anime_id);
7 CREATE INDEX
8 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

```
1 //Anime APIs
2 *To get all animes:
3 API: -> ('/api/anime', db.getAnime);
4
5 *To get an anime by id:
6 API: -> ('/api/anime/:id', db.getAnimebyid);
7
8 *To get an anime by name:
9 API: -> ('/api/anime/name/:name', db.getAnimebyname);
10
11 *To get an anime by genre:
12 API: -> ('/api/anime/genre/:genre', db.getAnimebyGenre);
13
14 *To get an anime according to rating threshold:
15 API: -> ('/api/anime/rating/:high', db.getAnimebyrating);
16
17 *To get an anime by range of rating:
18 API: -> ('/api/anime/rating/:high/:low', db.
    getAnimebyratingrange);
19
20 *To get an anime by range of episodes:
21 API: -> ('/api/anime/episodes/:high/:low', db.
    getAnimebyepisodes);
22
23 To get an anime by both rating and number of episodes' range:
24 API: -> ('/api/anime/episodes_rating/:high/:low/:highr/:lowr
    ', db.getAnimebyepisodes_rating);
25
26 To insert a new anime with id, name and genre:
27 API: -> ('/api/insert_anime/:id/:name/:genre', db.createanime);
28
29
30 //Movies' APIs
31 *To get all the movies:
32 API: -> ('/api/movies', db.getMovies);
33
34 *To get movies by name:
35 API: -> ('/api/movies/name/:name', db.getMoviebyname);
36
37 *To get movies by your favourite director:
38 API: -> ('/api/movies/director/:director', db.
    getMoviebyDirector);
39
40 *To get movies by your favourite actor:
41 API: -> ('/api/movies/actor/:actor', db.getMoviebyActor);
42
43 *To get movies by your favourite genre:
44 API: -> ('/api/movies/genre/:genre', db.getMoviesbyGenre);
45
46 *To get movies by budget:
47 API: -> ('/api/movies/budget/:low/:high', db.getMoviesbyBudget)
    ;
48
49 *To get movies by gross range:
50 API: -> ('/api/movies/gross/:low/:high', db.getMoviesbyGross);
51
52 *To get movies by ratings' range:
53 API: -> ('/api/movies/rating/:low/:high', db.getMoviesbyRating)
    ;
54
```

```

55 API: -> ('/api/insert_movie/:name/:genre/:year/:director/:
        actor1/:actor2/:actor3/:imdb/:budget/:gross', db.createMovie);
56
57 /*IPL api's*/
58
59 *To get all the teams:
60 API: -> ('/api/ipl/teams', db.getTeams);
61
62 *To get all the batsmen:
63 API: -> ('/api/ipl/batsmen', db.getBatsmen);
64
65 *To get batsmen bynumber of sixes:
66 API: -> ('/api/ipl/batsmen/six', db.getBatsmenbySix);
67
68 *To get batsmen by number of fours:
69 API: -> ('/api/ipl/batsmen/four', db.getBatsmenbyFour);
70
71 *To get all the bowlers:
72 API: -> ('/api/ipl/bowlers', db.getBowlers);
73
74 *To get all the teams by season:
75 API: -> ('/api/ipl/season/:season', db.getTeamsBySeason);
76
77 *To get batsman by name:
78 API: -> ('/api/ipl/batsman/:name', db.getBatsman);
79
80 *To get bowler by name:
81 API: -> ('/api/ipl/bowler/:name', db.getBowler);
82
83 *To get batsman by runs he scored:
84 API: -> ('/api/ipl/batsman/runs/:runs', db.getBatsmenByRuns);
85
86 *To get bowler by wickets he took:
87 API: -> ('/api/ipl/bowler/wickets/:wickets', db.
        getBowlersByWickets);
88
89 *To get batsman by average runs:
90 API: -> ('/api/ipl/batsman/avg/:avg', db.getBatsmenByAvg);
91
92 *To get batsman by strike rates:
93 API: -> ('/api/ipl/batsman/strike_rate/:strike_rate', db.
        getBatsmenByStrikeRate);
94
95 *To get bowler by avg runs:
96 API: -> ('/api/ipl/bowler/avg/:avg', db.getBowlersByAvg);
97
98 *To get bowler by economy:
99 API: -> ('/api/ipl/bowler/econ/:econ', db.getBowlersByEcon);
100
101 *To get matches of your favourite team:
102 API: -> ('/api/ipl/teams/matches/:name', db.getMatchesByTeam);
103
104 *To get matches according to your venue:
105 API: -> ('/api/ipl/venue/matches/:name', db.getMatchesByVenue);
106
107 *To get matches by winners:
108 API: -> ('/api/ipl/winner/matches/:winner', db.
        getMatchesByWinners);
109
110 * To get matches of your favourite batsman:
111 API: -> ('/api/ipl/favbatsmen/:name', db.getfavBatsmen);
112

```

```

113 To get matches of your favourite bowler:
114 API: -> ('/api/ipl/favbatsmen/:name', db.getfavbowler);

```

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
18. select * from batsmen_matches where batsman = (1) order by runs desc", req.params.name
19. select * from BOWLERS_MATCHES where bowler = (1) order by wickets desc", req.params.name

- **Movies:**

1. select * from movies order by title_year desc
2. select * from movies where **strpos**(movie_title,(1)) ≥ 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)) ≥ 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),(10)", [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
10	17.574 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
18	15.729 ms

6 ERD Diagrams

