NBA BASKETBALL PLAYER DATABASE MANAGEMENT SYSTEM

Hongzhi Liu (liu.hongz@husky.neu.edu)[1], *Sikai Ni (ni.si@husky.neu.edu)*[1]

[1] Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115

ABSTRACT

In this project, a practical Relational Database Management System (RDBMS) was designed and implemented for the statistics and records of NBA players. We first designed the relational model with UML diagram, then encoded it as SQL schemes. With JDBC API, we connected the database system with our user program. To carry out the review and modification functionality, a front-end GUI was developed based on HTML/CSS/JSP. With Spring/Spring MVC frameworks, each front-end operation was mapped into a controller-layer method, which connected with database by calling service-layer and DAO-layer methods. Several SQL queries were developed for Select, Insert, Update and Delete of game records. All the programming works were based on MySQL and Java EE web service.

Index Terms — Relational Database Management System, JDBC, MySQL, Java EE

1. INTRODUCTION

Game statistics are direct evaluations of player performances. Nowadays, more and more team managers are trying to explore game records for the decision of player transactions and employments. Therefore, it is significant to establish a user-friendly database management system for managers and fans to retrieve player records. Many websites provide statistics and records for this propose [1,2]. In this project, a practical NBA Basketball Player Database Management System (NBPDMS) was designed and implemented. The system includes following statistics:

- Player Basic Information (name, team, birthday, country, etc.)
- Team Basic Information (name, city, wins, founded time, etc.)
- Player Game Records (points, rebounds, etc.)
- Team Game Records (points, rebounds, etc.)
- Player Season Records (average points, average rebounds, etc.)
- Player Career Records (average points, average rebounds, etc.)
- Team Season Records (wins, championships, etc.)

2. REQUIREMENT

The NBPDMS and Users are participants in use cases. All the communications between users and NBPDMS are secure and authenticated, therefore activities such as login are not included in any use cases. The terms team, player, and statistic are used for entities that are administered by the NBPDMS. Bellows are several use-case requirements of this database system:

1. Query Teams [queryTeams]
Description: User queries the system to obtain information of relevant teams.
Step-by-step:
1. [#User] - A user requests information of teams that satisfy some specified criteria.
2. [#NBPDMS] - The system returns the information about all relevant teams.

2. Query Team Season Statistics [queryTeamSeason]
Description: User queries the system to obtain all season statistics of a team, including Wins, Loses, FG%, etc.
Step-by-step:
1. [#User] - The user submits the name of a team.
2. [#NBPDMS] - The system returns a list of season statistics of this team.

3. Query Team Statistics A Game [queryTeamAGame]
Description: User queries the system to obtain all game statistics of a team, including Home, Win, FG%, etc.
Step-by-step:
1. [#User] - The user submits the name of a team.
2. [#NBPDMS] - The system returns the list of game statistics of this team.

4. Insert Team Statistics A Game [insertTeamAGame]
Description: User gives the system a new team record of a game, inserts it into the database.
Step-by-step:
1. [#User] - The user provides new game record of a team.
2. [#NBPDMS] - The system inserts this record into database.

5. Update Team Statistics A Game [updateTeamAGame]
Description: User edits the team record of a game; system reflects corresponding updates in the database.
Step-by-step:
1. [#User] - The user provides new values of some columns of a game record of a team.
2. [#NBPDMS] - The system updates corresponding records.

6. Delete Team Statistics A Game [deleteTeamAGame]
Description: User deletes the team record of a game; system reflects corresponding affects in the database.
Step-by-step:
1. [#User] - The user asks to delete a game record of a team.
2. [#NBPDMS] - The system reflects corresponding changes.

7. Query Players [queryPlayers]
Description: User queries the system to obtain information of relevant players.
Step-by-step:
1. [#User] - A user requests information of players that play for a specific team at a specific season.
2. [#NBPDMS] - The system returns all the information about all relevant players.

8. Query Player Season/Career Statistics [queryPlayerSeasonCareer]
Description: User queries the system to obtain all season and career statistics of a player, including average Minutes Play, Win Share, FG%, etc.
Step-by-step:
1. [#User] - The user submits the name of a player.
2. [#NBPDMS] - The system returns a list of season and career statistics of this player.

9. Query Player Statistics A Game [queryPlayerAGame]
Description: User queries the system to obtain all game statistics of a player, including Minutes Play, FG%, points, etc.
Step-by-step:
1. [#User] - The user submits the name of a player.
2. [#NBPDMS] - The system returns the list of game statistics of this player.

10. Insert Player Statistics A Game [insertPlayerAGame]
Description: User gives the system a new player record of a game, inserts it into the database.
Step-by-step:
1. [#User] - The user provides new game record of a player.
2. [#NBPDMS] - The system inserts this record into database.

11. Update Player Statistics A Game
[updatePlayerAGame]
Description: User edits the player record of a game; system reflects corresponding updates in the database.
Step-by-step:
1. [#User] - The user provides new values of some columns of a game record of a player.
2. [#NBPDMS] - The system updates corresponding records.

12. Delete Player Statistics A Game [deletePlayerAGame]
Description: User deletes the player record of a game; system reflects corresponding affects in the database.
Step-by-step:
1. [#User] - The user asks to delete a game record of a player.
2. [#NBPDMS] - The system reflects corresponding changes.

3. DESIGN

In this project, there are 7 classes that play fundamental roll in both front-end pages and back-end database. *Player* contains all the basic information of a player (e.g. height, weight); *PlayerCareerStats* contains all the attributes of a player career statistics; *PlayerSeasonStats* contains all the attributes of a player season statistics; *PlayerStatisticsAGame* contains all the attributes of a player game statistics. *Team* contains all basic information of a team (e.g. city, wins); *TeamSeasonStatistics* contains all the attributes of a team season statistics; *TeamStatisticsAGame* contains all the attributes of a team game statistics.

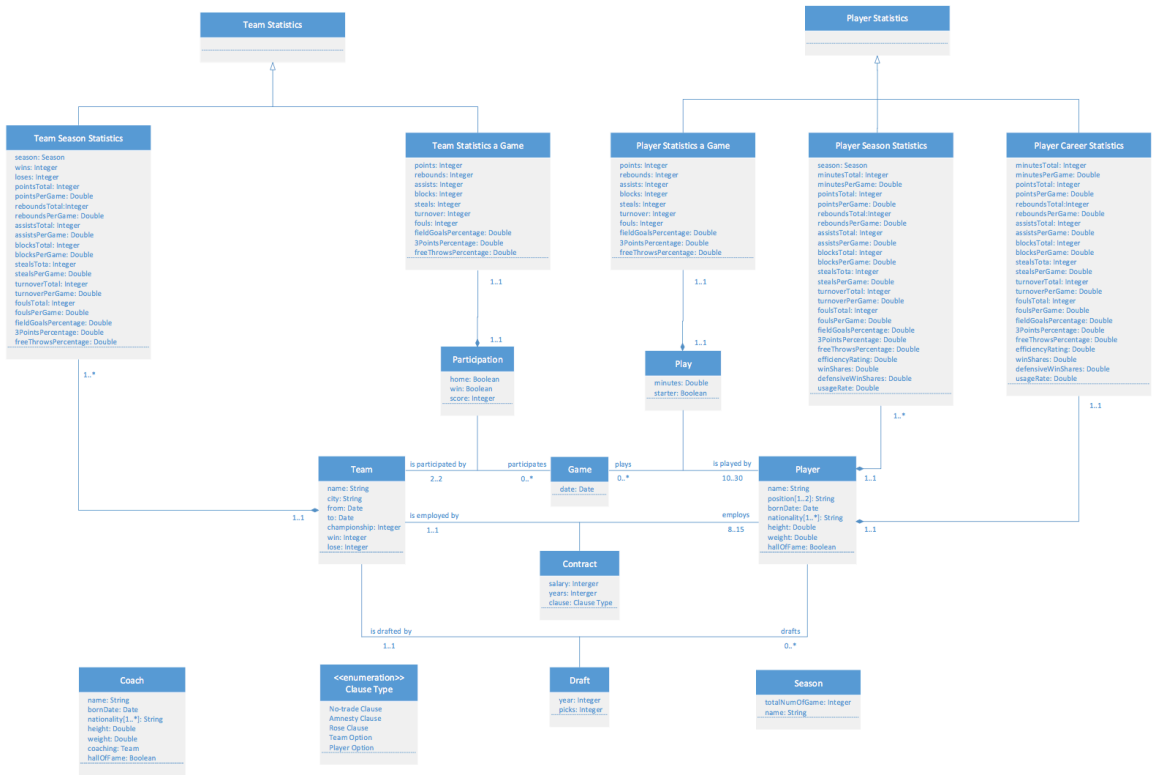


Fig. 1 UML diagram of the relational model

Detailed relational model is shown in figure 1. Average statistics (i.e. Player/Team Season Statistics, Player Career Statistics) are part of the Player/Team, therefore they are indicated using compositions. Per-game statistics (i.e. Player/Team Statistics A Game) must be specified by a game record, so it is part of the association class (i.e. Participation, Play). Team and Player are connected with three associations: Game, Contract and Draft.

4. IMPLEMENTATION

In this section, we first encoded the relational model as SQL schemes, then employed Spring/Spring MVC frameworks for system constructions. We also applied layered architecture and Inversion of control(IoC) principle to handle the bidirectional connection between front-end interface and back-end database.

4.1 Layered Architecture

From the principle of layered architecture, the system can be divided as view pages, controller layer, service layer, Data Access Object (DAO) layer, entity classes and database. This architecture promoted the flexibility and scalability of the system. It also enables us separate user interface from data-access logics. Besides, separation layer architecture guarantees a tightly-cohesion and loosely-coupling system property. It increases system security and developing independence. Changes in one layer would only have influence on itself instead of others.

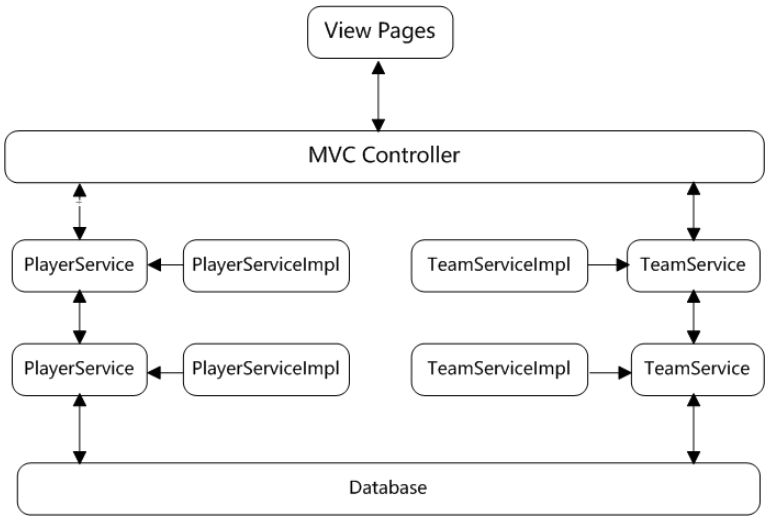


Fig. 2 Layered Architecture

In NBPDMS, a controller class is implemented in charge of service-layer methods. Service-layer consists of *TeamService* and *PlayerService* interfaces, each contains abstract methods processing business logic tasks. The implementation of those two service interfaces would call DAO-layer methods with respect to its tasks. *TeamDao* and *PlayerDao* are implemented in the DAO-layer which will map data to corresponding entities.

4.2 Spring IoC Container

The Spring framework plays a significant role of Inversion of Control (IoC) in this system. The Spring IoC container will create objects, wire/configure them together, and manage complete life cycle from creation until destruction. The Spring container uses Dependency Injection (DI) to manage the components that make up an application.

Here, *@Autowired* annotation was employed to give Spring IoC container control right of objects. This helped us get rid of XML configuration files [3]. Service objects would be auto-wired into controller object and injected by DAO objects, which would relief our programming workloads.

4.3 Model-View-Controller (MVC)

The Spring Web Model-View-Controller (MVC) framework is designed around a *DispatcherServlet* class that dispatches requests to handlers [4]. The default handler is based on the *@Controller* and *@RequestMapping* annotations, which offers a wide range of flexible handling methods. A view is responsible for rendering the HTML content. The model object is passed from controller to view template, making internal parameters accessible to the view pages.

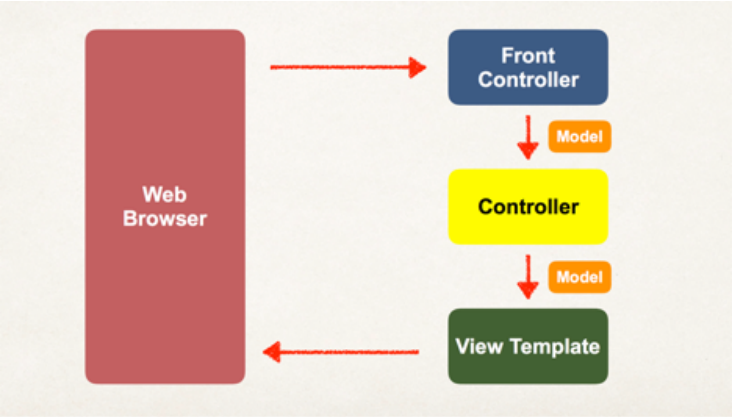


Fig. 3 MVC framework

In this project, 9 front-end pages were designed and implemented:

1. ***main-menu***: Main menu of the whole system, containing only one hyperlink *showTeams* controller method.
2. ***teams-page***: *showTeams* returns to this page, showing basic information of 30 NBA teams. It also contains 30 hyperlinks calling *showTeamSeasons* method.
3. ***team-seasons***: *showTeamSeasons* returns to this page, showing team statistics for each season. It also contains team-name hyperlinks calling *showTeamStatsAGame* and season hyperlinks calling *showTeamPlayer*.
4. ***team-stats-aGame***: *showTeamStatsAGame* returns to this page, showing game statistics for a specific team ordered by game date. It contains one *Add Stats* button calling *teamFormForAdd* method, an *update* hyperlink (per row) calling *teamFormForUpdate* method, and a *delete* hyperlink (per row) calling *deleteTeamStatsAGame*.
5. ***team-from***: *teamFormForAdd* and *teamFormForUpdate* both return to this page, showing original statistics for update method, and empty for add method. It also contains one *submit* button to call *saveForTeamStats*.
6. ***team-players***: *showTeamPlayer* method returns to this page, showing basic information of players in this team for this season. One may note that for a specific player, he may not play for this team in other seasons. This page also contains player-name hyperlinks calling *showPlayerSeasons* method.
7. ***player-seasons***: *showPlayerSeasons* method returns to this page, shows statistics per season for a player, contains team name hyperlinks *showPlayerStatsAGame* method.
8. ***player-stats-aGame***: *showPlayerStatsAGame* method returns to this page, showing all game statistics for a player ordered by game date. It also contains one *Add Stats* button to call *playerFormForAdd* method, one update hyperlink (per row) to call *playerFormForUpdate* method, and one delete hyperlink (per row) to call *deletePlayerStatsAGame* method.
9. ***player-from***: *playerFormForAdd* and *playerFormForUpdate* both return to this page, showing original statistics for update method and empty for add method. It also contains one *submit* button to call *saveForPlayerStats*.

4.4 JDBC in DAO-Layer

As we discussed above, the controller-methods called service-methods and DAO-methods serially, which implemented JDBC API by selecting, updating and deleting records from SQL tables. In DAO-layer, we employed only one *connection* for all methods. This approach significantly reduced the costs of rebuilding *connection*s. Besides, we applied *preparedStatement* to store and execute the SQL statements. For query results, we put selected fields in a result set, which maps data to entity objects with corresponding constructor. All the set and execute methods are in a try/catch block to handle the SQL exceptions.

5. DISCUSSION

In this section, we will show several examples of select, insert, update and delete operations of NBPDMS. The database implementation is based on MySQLWorkbench. The user program is based on Java EE web service with Apache Tomcat.

Team Statistics A Game

Add A Stats

Name	Date	Home	Win	FG%	3P%	FT%	TRB	AST	STL	BLK	TOV	PF	PTS	Action
Oklahoma City Thunder	2014-1-2	true	true	0.59	0.39	0.83	52	47	17	19	14	14	121	Update | Delete
Oklahoma City Thunder	2014-1-3	true	true	0.53	0.45	0.83	41	50	11	13	18	15	89	Update | Delete
Oklahoma City Thunder	2014-1-8	true	true	0.53	0.47	0.83	58	52	11	17	13	15	121	Update | Delete
Oklahoma City Thunder	2014-1-14	true	true	0.53	0.39	0.8	49	52	15	12	15	11	115	Update | Delete
Oklahoma City Thunder	2014-1-15	false	false	0.64	0.43	0.9	50	42	19	20	20	10	121	Update | Delete
Oklahoma City Thunder	2014-1-15	true	true	0.48	0.33	0.9	45	58	11	19	19	11	112	Update | Delete
Oklahoma City Thunder	2014-1-20	true	true	0.5	0.45	0.84	60	44	13	12	13	10	108	Update | Delete
Oklahoma City Thunder	2014-1-22	false	false	0.63	0.29	0.81	60	60	11	18	15	12	90	Update | Delete

Fig. 4 Query result of all game records of Oklahoma City Thunder, ordered by game date

Figure 4 shows query results of game records for a specific team. It can be noted that records are ordered by their Date. In this page, there are three hyperlinks for data modifications: *Delete* will remove the corresponding row in this table, *Update* and *Add A Stats* will return to Team Form pages.

Team Statistics A Game

Add A Stats

Name	Date	Home	Win	FG%	3P%	FT%	TRB	AST	STL	BLK	TOV	PF	PTS	Action
Oklahoma City Thunder	2014-1-3	true	true	0.53	0.45	0.83	41	50	11	13	18	15	89	Update | Delete
Oklahoma City Thunder	2014-1-8	true	true	0.53	0.47	0.83	58	52	11	17	13	15	121	Update | Delete
Oklahoma City Thunder	2014-1-14	true	true	0.53	0.39	0.8	49	52	15	12	15	11	115	Update | Delete
Oklahoma City Thunder	2014-1-15	false	false	0.64	0.43	0.9	50	42	19	20	20	10	121	Update | Delete
Oklahoma City Thunder	2014-1-15	true	true	0.48	0.33	0.9	45	58	11	19	19	11	112	Update | Delete
Oklahoma City Thunder	2014-1-20	true	true	0.5	0.45	0.84	60	44	13	12	13	10	108	Update | Delete
Oklahoma City Thunder	2014-1-22	false	false	0.63	0.29	0.81	60	60	11	18	15	12	90	Update | Delete
Oklahoma City Thunder	2014-1-25	false	false	0.68	0.27	0.88	42	55	12	16	16	19	82	Update | Delete

Fig. 5 Delete Game records.

Figure 5 shows the results of deletion operation. It is noted that the game records of 2014-1-2 are deleted, while others are maintained welly as they stand.

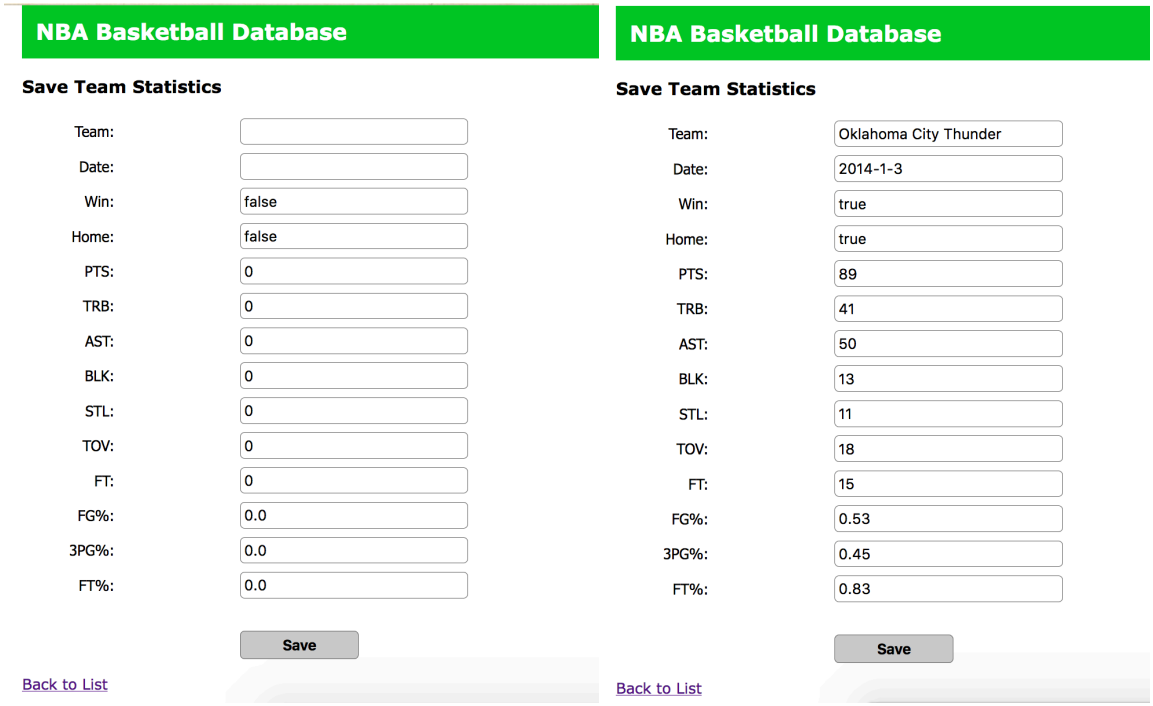
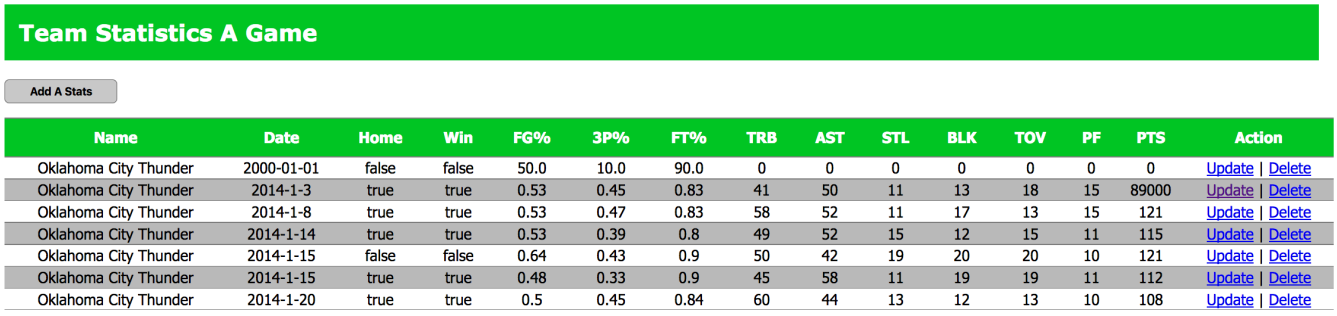


Fig. 6 Team Form page for Insert and Update game records.

Figure 6 shows the Team Form page of Insert and Update operations. For Insertion, the page is empty and ready for new records to be entered. For Update, the original statistics are inflected, one can just change the attributes he wants.



Name	Date	Home	Win	FG%	3P%	FT%	TRB	AST	STL	BLK	TOV	PF	PTS	Action
Oklahoma City Thunder	2000-01-01	false	false	50.0	10.0	90.0	0	0	0	0	0	0	0	Update | Delete
Oklahoma City Thunder	2014-1-3	true	true	0.53	0.45	0.83	41	50	11	13	18	15	89000	Update | Delete
Oklahoma City Thunder	2014-1-8	true	true	0.53	0.47	0.83	58	52	11	17	13	15	121	Update | Delete
Oklahoma City Thunder	2014-1-14	true	true	0.53	0.39	0.8	49	52	15	12	15	11	115	Update | Delete
Oklahoma City Thunder	2014-1-15	false	false	0.64	0.43	0.9	50	42	19	20	20	10	121	Update | Delete
Oklahoma City Thunder	2014-1-15	true	true	0.48	0.33	0.9	45	58	11	19	19	11	112	Update | Delete
Oklahoma City Thunder	2014-1-20	true	true	0.5	0.45	0.84	60	44	13	12	13	10	108	Update | Delete

Fig. 7 Result for Insert and Update.

Figure 7 shows the results for insert and update. We entered some exaggerated values to make the effect more recognizable. The first row is a newly-inserted team game record, and in second row, we update the original points 89 to 89000. All the examples, form figure 5 to figure 7, demonstrated the functionality we have proposed in use cases.

5. CONCLUSTION

In this project, a user-friendly database management system is designed and implemented. With front-page operations, one can easily select and modify game records of teams and players. With Spring/Spring MVC framework, the front-end operations are mapped into controller-layer methods in user program. Then the corresponding service-layer and DAO-layer methods are called to satisfy the functionality. The database system is connected with user program with JDBC API, with which we can establish a bidirectional mapping between Java entities and SQL rows. Several examples are given to demonstrate the operations of this system. With the combination of layered architecture, Spring IoC Container, and Model-View-Controller (MVC), the system is well encapsulated with loosely coupling, which reliefs our programming workloads and enforces system security.

6. REFERENCES

[1] http://www.basketball-reference.com/
[2] http://stats.nba.com/
[3] https://www.tutorialspoint.com/spring/spring_annotation_based_configuration.htm
[4] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html