Chapter 1

INTRODUCTION

Chapter 1

INTRODUCTION

Today's sporting system requires us to maintain records of all kinds of players' personal info, type of role they play in the team and the different team they play.
We aim to solve this problem for sport Cricket. Cricket Management System enables dynamic querying for searching players. It includes searching for specific players, adding new players with all their personal and Cricket related info. Support of feature that allows player to play in more than one league apart from the national team is provided.
The user friendly interface provided in this System increases the usability significantly. Especially the support of dynamic query helps user to filter unwanted players from the search result. It ensures seamless and smooth experience of user by making software more visually catching and simplicity in the application.
The player insertion feature is made robust so that invalid entries will be detected early and the player info won't be added to the database. The user is required to insert all the neccessary details like personal info, cricketing career stats, nationality etc.
There are also other features like league feature that allows the user to view different leagues that are played across different countries in the world, different teams that plays in each league and the players who play that league.
This system has plug and use capability that requires no type of prerequisites and no installation of any kind is needed. The simplicity in the application has come from the intelligent designing and thus due to this no high computing is present.
Portability is ensured as it can run in multiple Operating systems. The application size is less than 3 Megabytes. Hence no high storage space is required for storing the application itself and extra care is taken in designing the database so that all entries take minimal size.

Chapter 2

Project Requirements

Hardware

• Processor:	Pentium IV or higher
• RAM**:**	256 MB or higher
• Hard Disk**:**	10 GB or higher
• Cache Memory:	-
• Keyboard:	Microsoft Compatible 101 or higher

Software

• Operating System:	-
• Programming Language:	Java
• Front-End (Interface):	JavaFx
• Back-End (Database):	MySQL

Chapter 3

Literature Survey

3.1 Introduction to Maintenance Management

During Second World War there was a profound advance in engineering and technology that have bought the need and focused attention towards maintenance of engineering systems. Before this maintenance management was considered as a skill based functional discipline and on experience depended. There was the question frequently asked was ‘Is there a necessity for the Maintenance Management to change? Due to the increase in development towards modernization and the demand for higher productivity resulted in development and application of complex equipment. This has impacted in increased capital employed in production equipment (Waeyenbergh & Pintelon, 2002).

To bring control in the budget of the organization, downsizing was adopted to reduce the personnel availability for unscheduled work. The next is the energy costs, the largest part of any organization’s operational budget is the maintenance costs (Hansen, 2006; Lofsten, 2000; Park & Han, 2001). Therefore, it is necessary to have timely maintenance actions to minimize the incidence of failures by increasing the equipment reliability through the effective maintenance.

3.2 Maintenance Management

Maintenance is a combination of all technical, administrative, and managerial actions during the life cycle of an item to perform the required function. Before this, maintenance was supposed to be an expense account with performance measures. It was believed to track direct costs or surrogates such as the headcount of tradesmen and the total duration of forced outages during a specified period. Fortunately, nowadays maintenance is acknowledged as a major contributor to the performance and profitability of business organizations. Maintenance managers therefore explore every opportunity to improve on profitability and performance and achieve cost savings.

3.3 History of Maintenance Management

3.3.1 The First Generation of Maintenance Management

The first generation of maintenance management was before and up to the Second World War with the expectations of “Fix equipment when it breaks” with the views on equipment failure as all equipment “wears out”. The maintenance technicians were working with the basics of maintenance and the fundamental of repair skills of the equipment.

The first generation of maintenance management was “Reactive Maintenance”.

Reactive maintenance was considered as “run the equipment till it breaks” maintenance methodology. The maintenance team doesn’t do any actions or efforts to maintain the equipment as the original manufacturers are intended to ensure design life is reached.

3.3.2 The Second Generation of Maintenance Management

All equipment complies with the “Bath- tub” curve. Maintenance techniques followed are scheduled overhauls, systems for planning and controlling work, big, slow computers.

The second generation of maintenance management is “Preventive Maintenance”.

Preventive maintenance can be defined as follows: Actions performed on time- or machine-run-based schedule that detect, preclude, or mitigate degradation of a component or system with the aim of sustaining or extending its useful life through controlling degradation to an acceptable level.

3.3.3 The Third Generation of Maintenance Management

The third generation of maintenance management was after 1970s with the expectations of higher equipment availability, higher equipment reliability, greater safety, no environmental damage, better product quality, longer equipment life, greater cost effectiveness.

It is focused on six failure patterns of Nowlan and Heap.

Maintenance techniques followed are condition monitoring, design for maintainability and reliability, hazard studies, failure modes and effects analysis, small, fast computers, expert systems, team work and empowerment.

The third generation of maintenance management is “Reliability Maintenance”.

3.3.4 Generation of Maintenance

The consolidation summary of the generation of the maintenance management is shown in figure – 3.1.

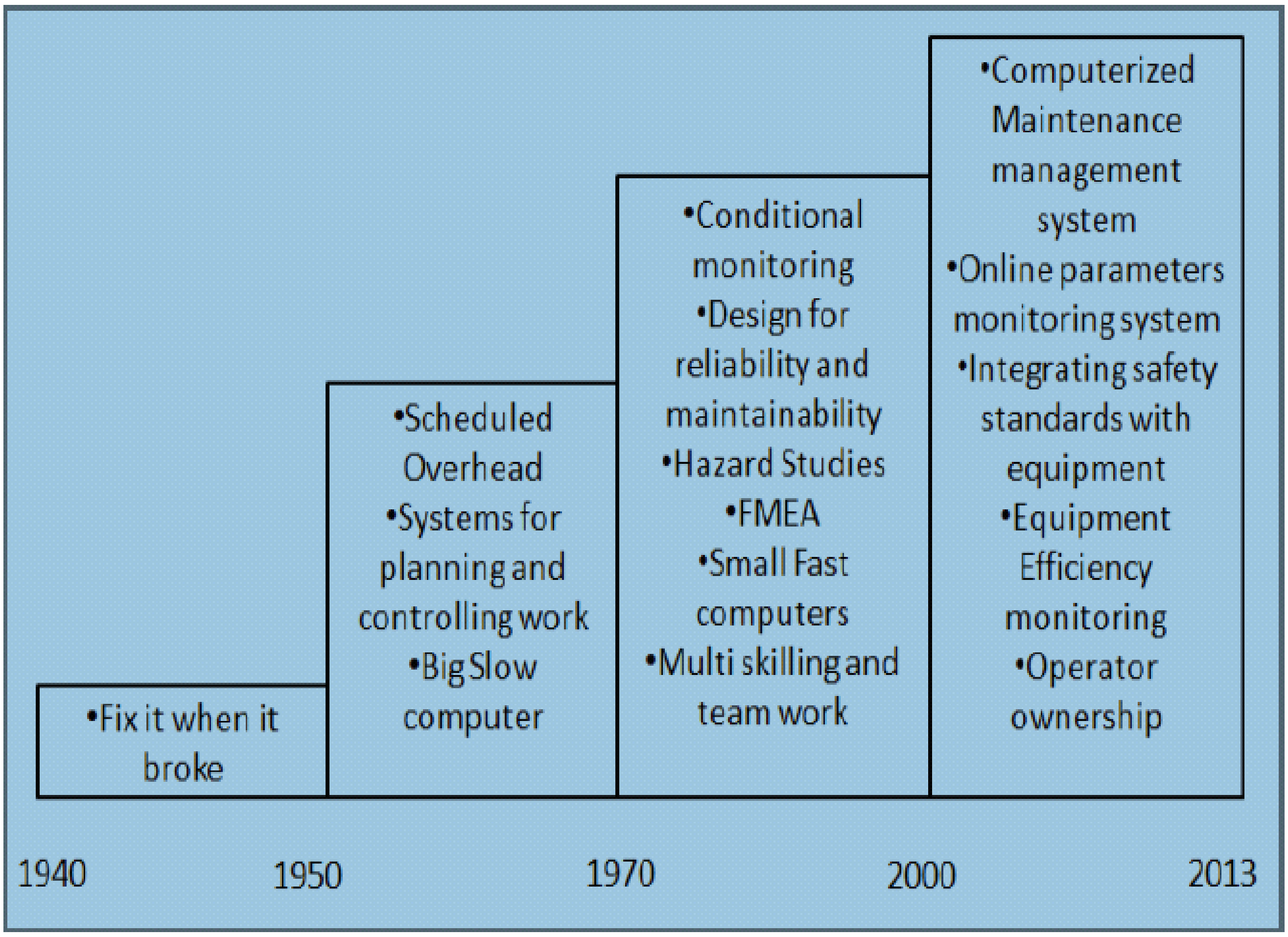

•Fix it when it broke
•Scheduled Overhead
•Systems for planning and controlling work
•Big Slow computer
•Conditional monitoring
•Design for reliability and maintainability
•Hazard Studies
•FMEA
•Small Fast computers
•Multi skilling and team work
•Computerized Maintenance management system
•Online parameters monitoring system
•Integrating safety standards with equipment
•Equipment Efficiency monitoring
•Operator ownership
1940
1950
1970
2000
2013

Fig. 3.1 Generation of Maintenance

3.4 Maintenance Organization

The Organisation structure may be Centralized or Decentralized or may be mix of two. Higher plant availability can be achieved in decentralization as it responds quickly to breakdowns. Maintenance organization concerns an optimum balance between plant availability and maintenance resource utilization. The three necessary and interdependent components are resources – men, spares and tools, administration authority and responsibility for deciding what, when and how the work should be carried out, work planning and control system- planning and scheduling the work giving feedback and directing correctly the maintenance effort towards defined objective.

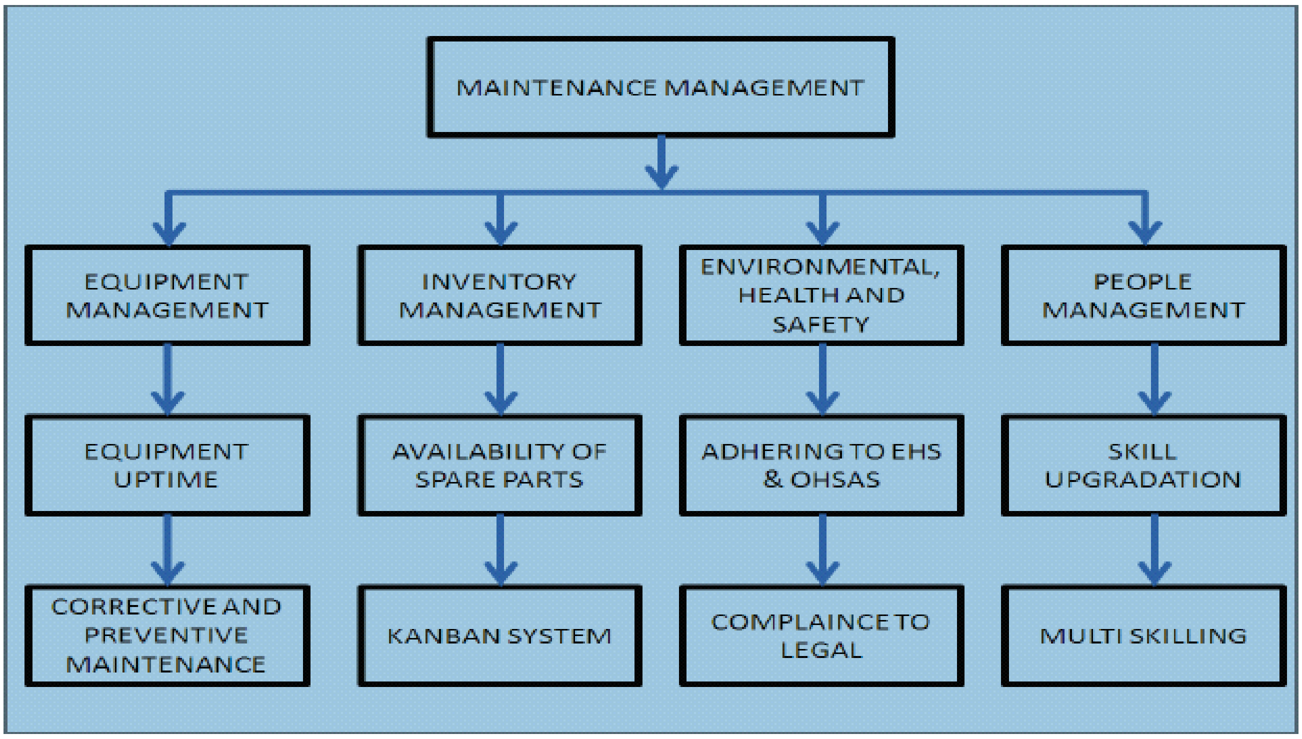


Fig. 3.2 Common Maintenance Practices

3.5 Lean Manufacturing

3.5.1 History of Lean Manufacturing

Taiichi Ohno founded Toyota Production System (TPS). In 1913, Henry Ford had invented Flow Production system, which was a real revolution of combination of constantly changing identical parts by moving conveyor belt. But the major problem faced by ford was that it was not able to produce different parts as all the machines were customized and there were no changeovers to meet the demand in the market for more models.

3.5.2 Lean Manufacturing Principles

- Specify Value
- Identify Value Stream
- Flow
- Pull System
- Continue Perfection

Chapter 4

Literature Survey and Problem Statement

As per **Lean Manufacturing**, every organization must listen to four voices that in order to formulate strategy for implementation. Those are:

- Voice of the Customer
- Voice of the Organization
- Voice of the Employee
- Voice of the Process

Some of these voices are sensitive to good management teams and others are less noticeable and require techniques to expose them.

Voices of the Customer – The business is driven by the customer and the organization focuses on the customer requirement of Quality, Cost, Delivery and Service

The voice of the internal customer will be captured with the production employees.

Through the detailed study of the literature survey we arrived at a conclusion that maintenance is important and integral part of any system and hence maintenance should be made more convenient as the time progresses. Hence, we arrive at our **problem statement** for the project:

Approach to creating a Cricket Management System as a Java application.

Chapter 5

System Design

Design is the first step in the development phase for any technique or principle. It helps in defining a device, process or system. It enables its physical realization.

After the software requirements have been analysed, the software design involves three technical activities - design, coding, implementation and testing.

The design activities are very important because it affects the success of the software implementation. A good system design helps in making the maintenance of system easy. It increases the reliability and maintainability of the system. Design can be used to translate the customer's requirements into finished software.

Through good design, quality of development can be improved. Software design is a process through which requirements can be translated into a software.

5.1 UML Diagrams

Actor:

A coherent set of roles that users of use cases play when interacting with the use cases.

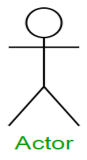


Fig. 5.1 Actor

Use case:

A description of sequence of actions, including variants, that a system performs that yields an observable result of value of an actor.

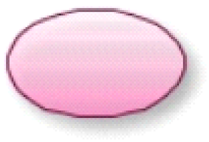


Fig. 5.2 Usecase

UML stands for Unified Modelling Language. UML is a language for specifying, visualizing and documenting the system. The goal of UML is to produce a model of the entities which are directly or indirectly involved in the project. The representation of the entities in the product being developed is needed to be designed. There are many UML diagrams that can be used to define the system e.g. Usecase Diagram.

5.1.1 Usecase Diagrams

A Use case is a description of set of sequence of actions. Graphically it is rendered as an ellipse with solid line including only its name. Use case diagram is a behavioural diagram that shows a set of use cases and actors and their relationship. It is an association between the use cases and actors. An actor represents a real-world object. Ex - Primary Actor - Sender, Secondary Actor - Receiver.

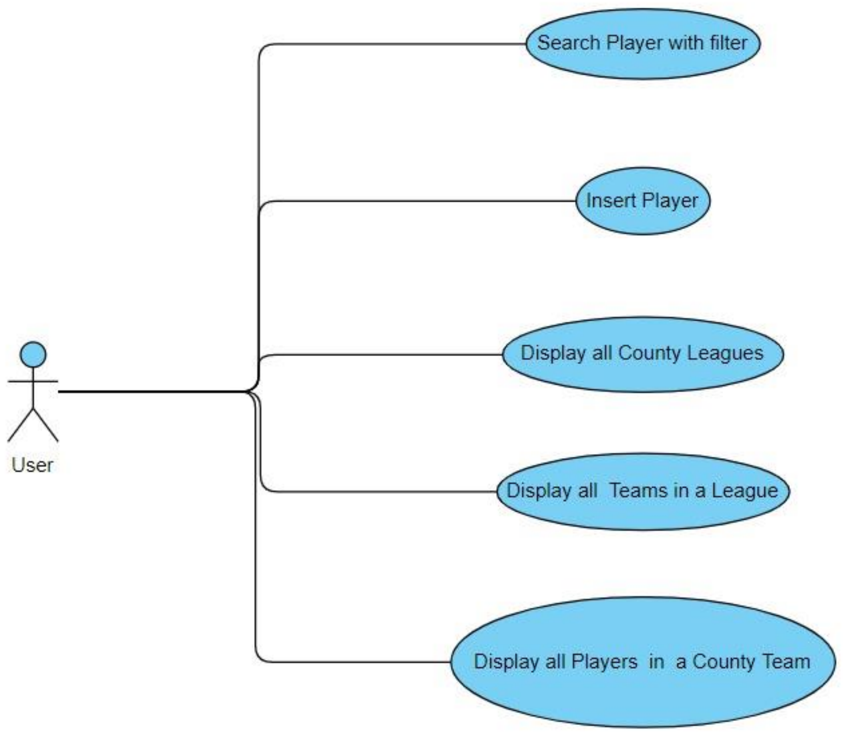

Search Player with filter
Insert Player
Display all County Leagues
Display all Teams in a League
Display all Players in a County Team
User

Fig. 5.3 Usecase Diagram

5.2 E-R Diagram

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 as a way to unify the network and relational database views. The ER model is a conceptual data model that views the real-world objects as entities and defines its relationships. A basic component of this model is the Entity-Relationship diagram. It is used to visually represents data objects. Today it is commonly used for database design by the database designer.

The utility of the ER model is:

- It maps well to the relational model.
- The constructs used in the ER model can easily be transformed into relational tables.
- It is simple and easy to understand with a minimum of training.
- The model can be used by the database designer to communicate the design to the end user.
- The model can be used as a design plan by the database developer to implement a data model in a database management software.

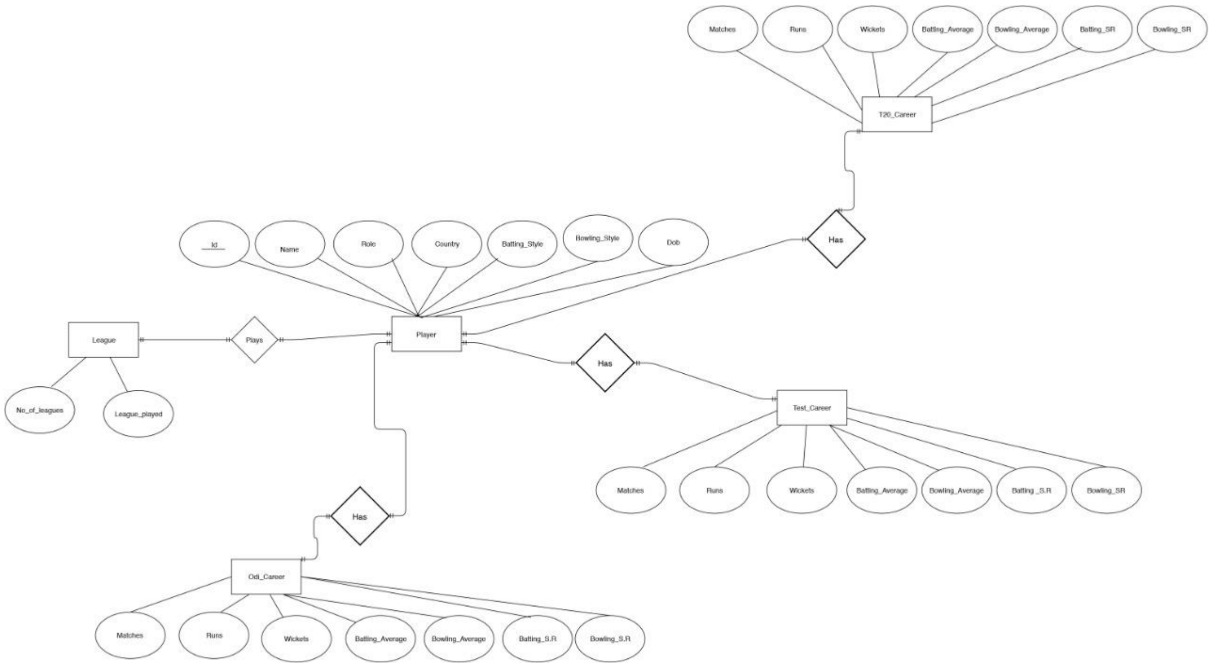


Fig. 5.4 E-R Diagram

5.3 Schema Diagram

A schema contains schema objects, which could be tables, columns, data types, views, stored procedures, relationships, primary keys, foreign keys, etc. A database schema can be represented in a visual diagram, which shows the database objects and their relationship with each other.

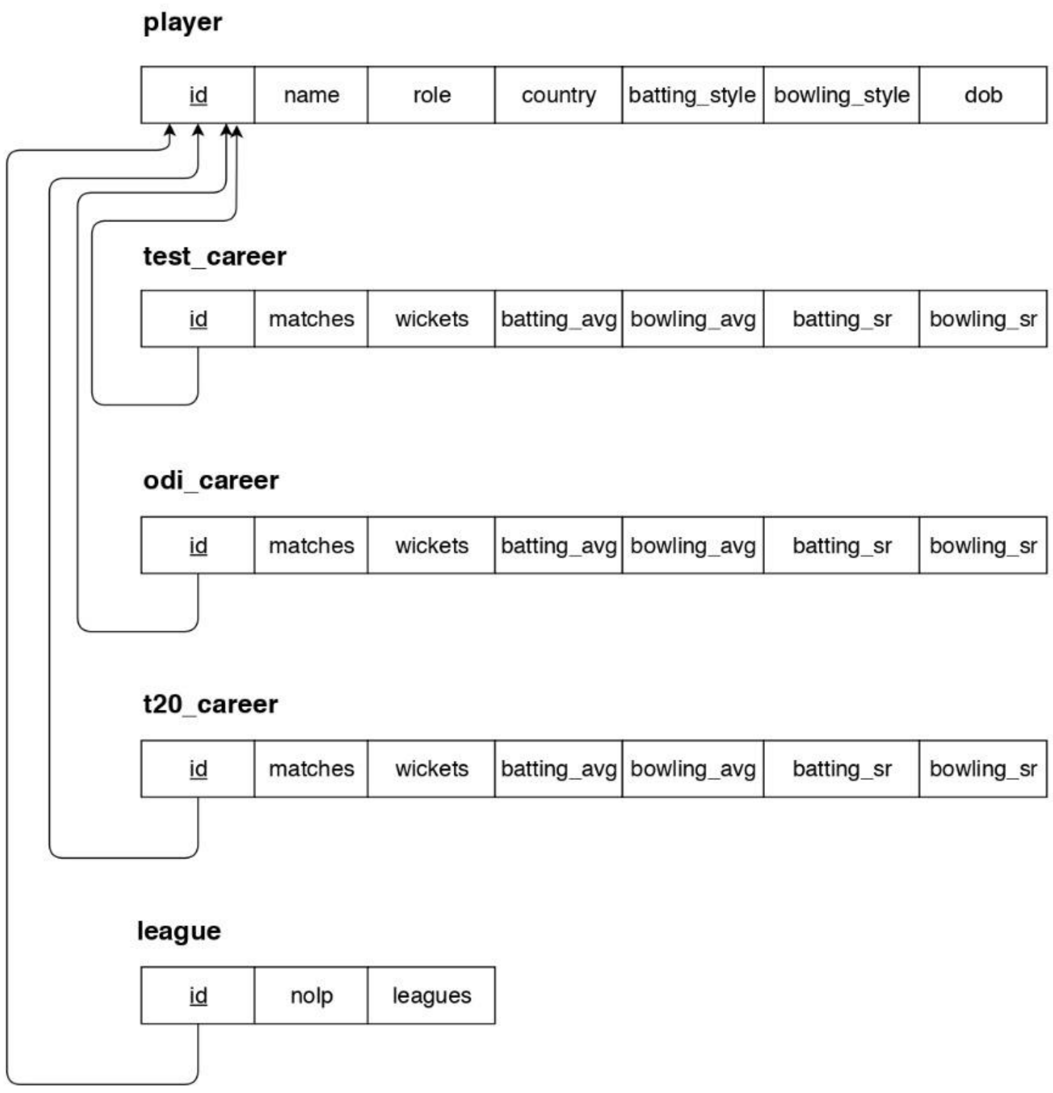


Fig. 5.5 Schema Diagram

Chapter 6

Implementation

6.1 Introduction

6.1 Front-end Technology

Java is one of the most important programming language in today's IT industries

JavaFX-JavaEX is a software platform for creating and delivering desktop applications,

as well as rich Internet applications (RIAS) that can run across a wide variety of devices

JavaFX is intended to replace Swing as the standard GUI library for Java SE,

but both will be included for the foreseeable future.

JavaBeans-This is something like Visual Basic,

a reusable software component that can be easily assemble to create some new and advanced

application.

Mobile-Besides the above technology, Java is also used in mobile devices,

many kind of games and services built-in Java.

Today, all leading mobile service provider like Nokia, Siemens, Vodafone are using Java

technology.

6.2 Database technology

MySQL

MYSQL is an open source relational database management system (RDBMS) based on

Structured Query Language (SQL). MYSQL runs on virtually all platforms,

in cluding Linux, UNIX, and Windows

6.4 CODE FOR JAVA

6.4.1 ESTABLISHING DATABASE CONNECTION

```
/* Connection to database  is established in every DAO class. The below method is
present in every DAO class to establish connection to database. */

public void connect() {

        try {

                con = DriverManager.getConnection(url, username, password);

        } catch (SQLException e) {

                System.out.println("Error in setting connection");

                e.printStackTrace();

        }

}
```

6.4.2 HOMEPAGE CODE

```
package application;


import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.stage.Stage;

import javafx.scene.Parent;

import javafx.scene.Scene;


public class Main extends Application {

        @Override

        public void start(Stage primaryStage) {

                try {

                        Parent root =
FXMLLoader.load(getClass().getResource("/application/Main.fxml"));      //  content
from Main.fxml is loaded and  the parent is set to root.

                        //Attach root object to scene
```

```
				Scene scene = new Scene(root);
				//Apply CSS to the scene

		scene.getStylesheets().add(getClass().getResource("application.css").toExternalF
orm());
				//  set scene to primary stage
				primaryStage.setScene(scene);
				//Make the primary stage visible
				primaryStage.show();
			} catch (Exception e) {
				//  If any Exception, then print the stack trace
				e.printStackTrace();
			}
		}
		public static void main(String[] args) {
			launch(args);
		}
}
```

6.4.3 ADDING NEW PLAYER

```
package application;


import java.net.URL;
import java.sql.SQLException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ResourceBundle;


import DAO.*;
import Main.*;
import javafx.beans.value.ChangeListener;

import javafx.beans.value.ObservableValue;

import javafx.collections.FXCollections;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.*;

import javafx.scene.layout.GridPane;


public class InsertPlayerController implements Initializable {

        @FXML

        private GridPane mainGrid;

        @FXML

        private Label errorLabel;

        @FXML

        private Button saveButton;

        @FXML

        private TextField nameTextField;

        @FXML

        private DatePicker dobDatePicker;

        @FXML

        private ComboBox<String> roleComboBox;

        @FXML

        private ComboBox<String> countryComboBox;

        @FXML

        private ComboBox<String> battingStyleComboBox;

        @FXML

        private ComboBox<String> bowlingStyleComboBox;
```

```
	@FXML
	private Button odiCareerButton;
	@FXML
	private Button testCareerButton;
	@FXML
	private Button t20CareerButton;
	@FXML
	private TextField matchesTextField;
	@FXML
	private TextField runsTextField;
	@FXML
	private TextField wicketsTextField;
	@FXML
	private TextField inningsTextField;
	@FXML
	private TextField bowlingAvgTextField;
	@FXML
	private TextField battingSRTextField;
	@FXML
	private TextField bowlingSRTextField;
	boolean[] valid = new boolean[16];
	String role[] = { "BATSMAN", "BOWLER", "BATTING ALLROUNDER",
"BOWLING  ALLROUNDER" };
	String country[] = { "INDIA", "AUSTRALIA", "ENGLAND", "SOUTH
AFRICA", "NEW ZEALAND", "PAKISTAN", "SRI LANKA",
				"WEST INDIES", "BANGLADESH", "AFGHANISTAN" };
	String battingStyle[] = { "RIGHT HANDED", "LEFT HANDED" };
	String bowlingStyle[] = { "RIGHT PACER", "RIGHT SPINNER", "LEFT
PACER", "LEFT SPINNER" };
```

```
	String testMatches = "", testRuns = "", testWickets = "", testInnings = "",
testBowlingAvg = "", testBattingSR = "",

				testBowlingSR = "";

	String odiMatches = "", odiRuns = "", odiWickets = "", odiInnings = "",
odiBowlingAvg = "", odiBattingSR = "",

				odiBowlingSR = "";

	String t20Matches = "", t20Runs = "", t20Wickets = "", t20Innings = "",
t20BowlingAvg = "", t20BattingSR = "",

				t20BowlingSR = "";

	boolean test = true, odi = false, t20 = false;


	@Override

	public void initialize(URL location, ResourceBundle resources) {

		testCareerButton.setStyle("-fx-background-color: #87cefa;");

		odiCareerButton.setStyle("-fx-background-color: #fffff0;");

		t20CareerButton.setStyle("-fx-background-color: #fffff0;");

		setIsValidArray();

		setComboBoxes();

		setListeners();

	}


	private void setIsValidArray() {

		for (int i = 0; i < valid.length; i++)

			valid[i] = false;

	}


	private void setListeners() {

		matchesTextField.textProperty().addListener(new
ChangeListener<String>() {
```

```
                    @Override
                    public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
                        if ((test) && (!odi) && (!t20))
                            testMatches = newValue;
                        else if ((!test) && (odi) && (!t20))
                            odiMatches = newValue;
                        else if ((!test) && (!odi) && (t20))
                            t20Matches = newValue;
                        else
                            System.out.println("Error in  matchesTextField");
                    }
                });
                runsTextField.textProperty().addListener(new ChangeListener<String>()
{

                    @Override
                    public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
                        if ((test) && (!odi) && (!t20))
                            testRuns = newValue;
                        else if ((!test) && (odi) && (!t20))
                            odiRuns = newValue;
                        else if ((!test) && (!odi) && (t20))
                            t20Runs = newValue;
                        else
                            System.out.println("Error in  runsTextField");
                    }
                });
```

```
		wicketsTextField.textProperty().addListener(new
ChangeListener<String>() {

			@Override
			public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
				if ((test) && (!odi) && (!t20))
					testWickets = newValue;
				else if ((!test) && (odi) && (!t20))
					odiWickets = newValue;
				else if ((!test) && (!odi) && (t20))
					t20Wickets = newValue;
				else
					System.out.println("Error in  wicketsTextField");
			}
		});
		inningsTextField.textProperty().addListener(new
ChangeListener<String>() {

			@Override
			public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
				if ((test) && (!odi) && (!t20))
					testInnings = newValue;
				else if ((!test) && (odi) && (!t20))
					odiInnings = newValue;
				else if ((!test) && (!odi) && (t20))
					t20Innings = newValue;
				else
					System.out.println("Error in  inningsTextField");
				}

			});

			bowlingAvgTextField.textProperty().addListener(new
ChangeListener<String>() {


				@Override

				public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {

					if ((test) && (!odi) && (!t20))

						testBowlingAvg = newValue;

					else if ((!test) && (odi) && (!t20))

						odiBowlingAvg = newValue;

					else if ((!test) && (!odi) && (t20))

						t20BowlingAvg = newValue;

					else

						System.out.println("Error in
bowlingAvgTextField");

				}

			});

			battingSRTextField.textProperty().addListener(new
ChangeListener<String>() {


				@Override

				public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {

					if ((test) && (!odi) && (!t20))

						testBattingSR = newValue;

					else if ((!test) && (odi) && (!t20))

						odiBattingSR = newValue;

					else if ((!test) && (!odi) && (t20))
```

```
                    t20BattingSR = newValue;
                else
                    System.out.println("Error in
battingSRTextField");
            }
        });
        bowlingSRTextField.textProperty().addListener(new
ChangeListener<String>() {

            @Override
            public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
                if ((test) && (!odi) && (!t20))
                    testBowlingSR = newValue;
                else if ((!test) && (odi) && (!t20))
                    odiBowlingSR = newValue;
                else if ((!test) && (!odi) && (t20))
                    t20BowlingSR = newValue;
                else
                    System.out.println("Error in
bowlingSRTextField");
            }
        });
    }

    public void careerButton(ActionEvent ae) {
        Button b = (Button) ae.getSource();
        if (testCareerButton == b) {
            testCareerButton.setStyle("-fx-background-color: #87cefa;");
            odiCareerButton.setStyle("-fx-background-color: #fffff0;");
            t20CareerButton.setStyle("-fx-background-color: #fffff0;");
            System.out.println("Test Career button clicked");
            test = true;
            odi = t20 = false;
            matchesTextField.setText(testMatches);
            runsTextField.setText(testRuns);
            wicketsTextField.setText(testWickets);
            inningsTextField.setText(testInnings);
            battingSRTextField.setText(testBattingSR);
            bowlingAvgTextField.setText(testBowlingAvg);
            bowlingSRTextField.setText(testBowlingSR);
        } else if (odiCareerButton == b) {
            testCareerButton.setStyle("-fx-background-color: #fffff0;");
            odiCareerButton.setStyle("-fx-background-color: #87cefa;");
            t20CareerButton.setStyle("-fx-background-color: #fffff0;");
            System.out.println("ODI career button clicked");
            odi = true;
            test = t20 = false;
            matchesTextField.setText(odiMatches);
            runsTextField.setText(odiRuns);
            wicketsTextField.setText(odiWickets);
            inningsTextField.setText(odiInnings);
            battingSRTextField.setText(odiBattingSR);
            bowlingAvgTextField.setText(odiBowlingAvg);
            bowlingSRTextField.setText(odiBowlingSR);
        } else if (t20CareerButton == b) {
            testCareerButton.setStyle("-fx-background-color: #fffff0;");
            odiCareerButton.setStyle("-fx-background-color: #fffff0;");
```

```
                        t20CareerButton.setStyle("-fx-background-color: #87cefa;");
                        System.out.println("T20 career button clicked");
                        t20 = true;
                        odi = test = false;
                        matchesTextField.setText(t20Matches);
                        runsTextField.setText(t20Runs);
                        wicketsTextField.setText(t20Wickets);
                        inningsTextField.setText(t20Innings);
                        battingSRTextField.setText(t20BattingSR);
                        bowlingAvgTextField.setText(t20BowlingAvg);
                        bowlingSRTextField.setText(t20BowlingSR);
                } else
                        System.out.println("Error in careerButton function!");
        }

        private boolean isAlpha(String name) {
                char[] chars = name.toCharArray();

                for (char c : chars) {
                        if (c == ' ')
                                continue;
                        if (!Character.isLetter(c)) {
                                return false;
                        }
                }

                return true;
        }
```

```
	private void setComboBoxes() {
		roleComboBox.setItems(FXCollections.observableArrayList(role));

	countryComboBox.setItems(FXCollections.observableArrayList(country));

	battingStyleComboBox.setItems(FXCollections.observableArrayList(battingStyl
e));

	bowlingStyleComboBox.setItems(FXCollections.observableArrayList(bowlingS
tyle));
	}

	public void saveClick(ActionEvent ae) {
		errorLabel.setText("Loaing");
		errorLabel.setStyle("-fx-text-fill: yellow;");
		if ((nameTextField.getText() == null) ||
(nameTextField.getText().isEmpty())
					|| !isAlpha(nameTextField.getText())) {
			errorLabel.setStyle("-fx-text-fill: red;");
			errorLabel.setText("Name field is invalid");
			return;
		}
		LocalDate ld = dobDatePicker.getValue();
		if ((ld == null) || (ld.toString() == null || ld.toString().isEmpty()) ||
(ld.isAfter(LocalDate.now()))) {
			errorLabel.setStyle("-fx-text-fill: red;");
			errorLabel.setText("Date is invalid");
			return;
		}
		if (roleComboBox.getSelectionModel().isEmpty()) {
```

```
                errorLabel.setStyle("-fx-text-fill: red;");

                errorLabel.setText("Select any role");

                return;

        }

        if (countryComboBox.getSelectionModel().isEmpty()) {

                errorLabel.setStyle("-fx-text-fill: red;");

                errorLabel.setText("Select any country");

                return;

        }

        if (battingStyleComboBox.getSelectionModel().isEmpty()) {

                errorLabel.setStyle("-fx-text-fill: red;");

                errorLabel.setText("Select any batting style");

                return;

        }

        if (bowlingStyleComboBox.getSelectionModel().isEmpty()) {

                errorLabel.setStyle("-fx-text-fill: red;");

                errorLabel.setText("Select any bowling style");

                return;

        }

        if (testMatches == null || testRuns == null || testWickets == null ||
testBattingSR == null

                        || testInnings == null || testBowlingAvg == null ||
testBowlingSR == null) {

                errorLabel.setStyle("-fx-text-fill: red;");

                errorLabel.setText("Test Career data  is invalid");

                return;

        }

        if (odiMatches == null || odiRuns == null || odiWickets == null ||
odiBattingSR == null || odiInnings == null
```

```
                        || odiBowlingAvg == null || odiBowlingSR == null) {
                errorLabel.setStyle("-fx-text-fill: red;");
                errorLabel.setText("ODI Career data  is invalid");
                return;
        }
        if (t20Matches == null || t20Runs == null || t20Wickets == null ||
t20BattingSR == null || t20Innings == null
                        || t20BowlingAvg == null || t20BowlingSR == null) {
                errorLabel.setStyle("-fx-text-fill: red;");
                errorLabel.setText("T20 Career data  is invalid");
                return;
        }
        trim();
        if (testMatches.isEmpty() || testRuns.isEmpty() || testWickets.isEmpty() ||
testInnings.isEmpty()
                        || testBattingSR.isEmpty() || testBowlingAvg.isEmpty() ||
testBowlingSR.isEmpty()) {
                errorLabel.setStyle("-fx-text-fill: red;");
                errorLabel.setText("Test Career data  is invalid");
                return;
        }
        if (odiMatches.isEmpty() || odiRuns.isEmpty() || odiWickets.isEmpty() ||
odiInnings.isEmpty()
                        || odiBattingSR.isEmpty() || odiBowlingAvg.isEmpty() ||
odiBowlingSR.isEmpty()) {
                errorLabel.setStyle("-fx-text-fill: red;");
                errorLabel.setText("ODI Career data  is invalid");
                return;
        }
        if (t20Matches.isEmpty() || t20Runs.isEmpty() || t20Wickets.isEmpty() ||
t20Innings.isEmpty()
                                        || t20BattingSR.isEmpty() || t20BowlingAvg.isEmpty() ||
t20BowlingSR.isEmpty()) {
                              errorLabel.setStyle("-fx-text-fill: red;");
                              errorLabel.setText("T20 Career data  is invalid");
                              return;
                    }
                    System.out.println("About to check test  data");
                    if (isNotNumValid(testMatches, false) || isNotNumValid(testRuns, false)
|| isNotNumValid(testWickets, false)
                                        || isNotNumValid(testInnings, false) ||
isNotNumValid(testBattingSR, true)
                                        || isNotNumValid(testBowlingAvg, true) ||
isNotNumValid(testBowlingSR, true)) {
                              errorLabel.setStyle("-fx-text-fill: red;");
                              errorLabel.setText("Test Career data  is invalid");
                              return;
                    }
                    System.out.println("About to check odi data");
                    if (isNotNumValid(odiMatches, false) || isNotNumValid(odiRuns, false)
|| isNotNumValid(odiWickets, false)
                                        || isNotNumValid(odiInnings, false) ||
isNotNumValid(odiBattingSR, true)
                                        || isNotNumValid(odiBowlingAvg, true) ||
isNotNumValid(odiBowlingSR, true)) {
                              errorLabel.setStyle("-fx-text-fill: red;");
                              errorLabel.setText("odi Career data  is invalid");
                              return;
                    }
                    System.out.println("About to check t20  data");
                    if (isNotNumValid(t20Matches, false) || isNotNumValid(t20Runs, false)
|| isNotNumValid(t20Wickets, false)
                                        || isNotNumValid(t20Innings, false) ||
isNotNumValid(t20BattingSR, true)
                                        || isNotNumValid(t20BowlingAvg, true) ||
isNotNumValid(t20BowlingSR, true)) {
                              errorLabel.setStyle("-fx-text-fill: red;");
                              errorLabel.setText("t20 Career data  is invalid");
                              return;
                    }
                    errorLabel.setStyle("-fx-text-fill: green;");
                    errorLabel.setText("Player successfully saved.");
                    int id = savePlayer();
                    saveTestCareer(id);
                    saveODICareer(id);
                    saveT20Career(id);
          }


          private void saveODICareer(int id) {
                    int matches = Integer.parseInt(odiMatches);
                    int runs = Integer.parseInt(odiRuns);
                    int wickets = Integer.parseInt(odiWickets);
                    int innings = Integer.parseInt(odiInnings);
                    double bowlAvg = Double.parseDouble(odiBowlingAvg);
                    double batSR = Double.parseDouble(odiBattingSR);
                    double bowlSR = Double.parseDouble(odiBowlingSR);
                    ODICareerDAO odiDAO = new ODICareerDAO();
                    odiDAO.connect();
                    ODICareer oc = new ODICareer(matches, runs, wickets, innings,
bowlAvg, batSR, bowlSR);
```

```
        oc.setId(id);
        try {
            odiDAO.insertODICareer(oc);
            odiDAO.UpdateAvg(id);
        } catch (SQLException e) {
            System.out.println("Error in inserting odi career");
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        odiDAO.close();
    }

    private void saveT20Career(int id) {
        int matches = Integer.parseInt(t20Matches);
        int runs = Integer.parseInt(t20Runs);
        int wickets = Integer.parseInt(t20Wickets);
        int innings = Integer.parseInt(t20Innings);
        double bowlAvg = Double.parseDouble(t20BowlingAvg);
        double batSR = Double.parseDouble(t20BattingSR);
        double bowlSR = Double.parseDouble(t20BowlingSR);
        T20CareerDAO t20DAO = new T20CareerDAO();
        t20DAO.connect();
        T20Career t20c = new T20Career(matches, runs, wickets, innings,
bowlAvg, batSR, bowlSR);
        t20c.setId(id);
        try {
            t20DAO.insertT20Career(t20c);
				t20DAO.UpdateAvg(id);
			} catch (SQLException e) {
				System.out.println("Error in inserting t20 career");
				e.printStackTrace();
			} catch (Exception e) {
				e.printStackTrace();
			}
			t20DAO.close();
		}

		private void saveTestCareer(int id) {
			int matches = Integer.parseInt(testMatches);
			int runs = Integer.parseInt(testRuns);
			int wickets = Integer.parseInt(testWickets);
			int innings = Integer.parseInt(testInnings);
			double bowlAvg = Double.parseDouble(testBowlingAvg);
			double batSR = Double.parseDouble(testBattingSR);
			double bowlSR = Double.parseDouble(testBowlingSR);
			TestCareerDAO tdao = new TestCareerDAO();
			tdao.connect();
			TestCareer tc = new TestCareer(matches, runs, wickets, innings,
bowlAvg, batSR, bowlSR);
			tc.setId(id);
			try {
				tdao.insertTestCareer(tc);
				tdao.UpdateAvg(id);
			} catch (SQLException e) {
				System.out.println("Error in inserting test career");
				e.printStackTrace();
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			tdao.close();
		}

		private int savePlayer() {
			PlayerDAO pdao = new PlayerDAO();
			pdao.connect();
			String name = nameTextField.getText();
			int role = roleComboBox.getSelectionModel().getSelectedIndex() + 1;
			int country = countryComboBox.getSelectionModel().getSelectedIndex()
+ 1;
			int batt_s =
battingStyleComboBox.getSelectionModel().getSelectedIndex() + 1;
			int bowl_s =
bowlingStyleComboBox.getSelectionModel().getSelectedIndex() + 1;
			LocalDate ld = dobDatePicker.getValue();
			String dob = ld.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
			System.out.println("///dob :: " + dob);
			Player player = new Player(name, role, country, batt_s, bowl_s, dob);
			int id = -1;
			try {
				id = pdao.insertPlayer(player);
			} catch (SQLException e) {
				System.out.println("Error in inserting player");
				e.printStackTrace();
```

```
        }
        pdao.close();
        return id;
    }

    private void trim() {
        testMatches.trim();
        testRuns.trim();
        testWickets.trim();
        testInnings.trim();
        testBattingSR.trim();
        testBowlingAvg.trim();
        testBowlingSR.trim();

        odiMatches.trim();
        odiRuns.trim();
        odiWickets.trim();
        odiInnings.trim();
        odiBattingSR.trim();
        odiBowlingAvg.trim();
        odiBowlingSR.trim();

        t20Matches.trim();
        t20Runs.trim();
        t20Wickets.trim();
        t20Innings.trim();
        t20BattingSR.trim();
        t20BowlingAvg.trim();
```

```
		t20BowlingSR.trim();
	}

	private boolean isNotNumValid(String s, boolean hasDot) {
		if (s.matches("[0-9]+")) {
			if (Integer.parseInt(s) >= 0) {
				System.out.println(Integer.parseInt(s));
				return false;
			}
		}
		int count = 0;
		for (int i = 0; i < s.length(); i++) {
			char c = s.charAt(i);
			if (c == '.')
				count++;
		}
		if ((count == 0 && !hasDot))
			return false;
		else if ((count == 0 || count == 1) && hasDot)
			return false;
		else
			return true;
	}
}
```

6.4.4 SEARCHING FOR PLAYERS IN CATALOG

```
package application;

import java.io.IOException;
import java.net.URL;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.ResourceBundle;

import DAO.*;
import Main.Player;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.geometry.Insets;
import javafx.scene.control.Button;

import javafx.scene.control.TextField;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.layout.GridPane;

public class CatalogController implements Initializable {
	@FXML
	private VBox mainVBox;
	@FXML

	private GridPane gridPane1;

	@FXML

	private TextField nameTextField;

	@FXML

	private ComboBox<String> roleComboBox;

	@FXML

	private ComboBox<String> countryComboBox;

	@FXML

	private GridPane gridPane2;

	@FXML

	private ComboBox<String> battingStyleComboBox;

	@FXML

	private ComboBox<String> bowlingStyleComboBox;

	@FXML

	private Button searchButton;

	@FXML

	private ScrollPane scroll;

	@FXML

	private VBox contentVBox;

	@FXML

	private GridPane contentGrid;

	PlayerDAO pdao;

	String role[] = { "ANYTHING", "BATSMAN", "BOWLER", "BATTING
ALLROUNDER", "BOWLING  ALLROUNDER" };

	String country[] = { "ANYTHING", "INDIA", "AUSTRALIA", "ENGLAND",
"SOUTH AFRICA", "NEW ZEALAND", "PAKISTAN",

				"SRI LANKA", "WEST INDIES", "BANGLADESH",
"AFGHANISTAN" };
```

```
	String battingStyle[] = { "ANYTHING", "RIGHT HANDED", "LEFT
HANDED" };

	String bowlingStyle[] = { "ANYTHING", "RIGHT PACER", "RIGHT
SPINNER", "LEFT PACER", "LEFT SPINNER" };


	@Override

	public void initialize(URL location, ResourceBundle resources) {

		// searchButton.setStyle("-fx-background-color: #fffafa;");

		setComboBoxes();

		setInitialData();

		contentVBox.prefWidthProperty().bind(scroll.widthProperty());

		scroll.prefWidthProperty().bind(contentGrid.widthProperty());

	}


	public void setInitialData() {

		pdao = new PlayerDAO();

		pdao.connect();

		ArrayList<Player> players = null;

		try {

			players = pdao.findPlayer("$#$0$0$0$0");

			System.out.println(players);

		} catch (IllegalArgumentException | SQLException e) {

			System.out.println("Error");

			Log.print();

		}

		addContent(players);

		pdao.close();

	}
	public void setComboBoxes() {
		roleComboBox.setItems(FXCollections.observableArrayList(role));

	countryComboBox.setItems(FXCollections.observableArrayList(country));

	battingStyleComboBox.setItems(FXCollections.observableArrayList(battingStyl
e));

	bowlingStyleComboBox.setItems(FXCollections.observableArrayList(bowlingS
tyle));
		roleComboBox.getSelectionModel().selectFirst();
		countryComboBox.getSelectionModel().selectFirst();
		battingStyleComboBox.getSelectionModel().selectFirst();
		bowlingStyleComboBox.getSelectionModel().selectFirst();
	}

	public GridPane getPlayerContent() throws IOException {
		GridPane gp = null;
		try {
			gp =
FXMLLoader.load(getClass().getResource("/application/PlayerContent.fxml"));
		} catch (IOException e) {
			System.out.println("Error getting PlayerContent");
			throw e;
		}
		return gp;
	}

	public void search(ActionEvent ae) {
		String name = nameTextField.getText().trim();
```

```
                if (name == null || name.isEmpty())
                        name = "#";
                int role = roleComboBox.getSelectionModel().getSelectedIndex();
                int country =
countryComboBox.getSelectionModel().getSelectedIndex();
                int battingStyle =
battingStyleComboBox.getSelectionModel().getSelectedIndex();
                int bowlingStyle =
bowlingStyleComboBox.getSelectionModel().getSelectedIndex();
                pdao.connect();
                String query = "$" + name + "$" + role + "$" + country + "$" +
battingStyle + "$" + bowlingStyle;
                System.out.println(query);
                ArrayList<Player> players = null;
                try {
                        players = pdao.findPlayer(query);
                } catch (IllegalArgumentException | SQLException e) {
                        Log.print();
                        e.printStackTrace();
                }
                contentVBox.getChildren().clear();
                addContent(players);
        }


        public void addContent(ArrayList<Player> players) {
                GridPane gp = null;
                for (int i = 0; i < players.size(); i++) {
                        try {
                                gp = getPlayerContent();
                        } catch (IOException e) {
```

```
                e.printStackTrace();
        }
        gp.getChildren().set(0, new Label(players.get(i).name));
        Label l = (Label) gp.getChildren().get(0);
        l.setPadding(new Insets(5, 0, 5, 10));
        l.setStyle("-fx-background-color:#ffffff;");
        l.setFont(new Font(25));
        l.setText(l.getText().toUpperCase());
        l.prefWidthProperty().bind(contentVBox.widthProperty());
        l.setId("$" + players.get(i).id + "");
        System.out.println(l.getId());

        l.setOnMouseClicked(new EventHandler<Event>() {

                @Override
                public void handle(Event event) {
                        CurrentPlayer.setID(l.getId());
                        System.out.println("Clicked on " + l.getText());
                        mainVBox.getChildren().clear();
                        GridPane playerGP = null;
                        try {
                                playerGP =
FXMLLoader.load(getClass().getResource("/application/Player.fxml"));
                        } catch (IOException e) {
                                System.out.println("Failed to load player
GridPane");
                                e.printStackTrace();
                        }
playerGP.prefHeightProperty().bind(mainVBox.heightProperty());

playerGP.prefWidthProperty().bind(mainVBox.widthProperty());
                                    mainVBox.getChildren().add(playerGP);
                              }
                        });
                        contentVBox.getChildren().add(gp);
                  }
            }

}
```

6.5 DATABASE

6.5.1 Creation and Insertion in Tables

create table player(id int primary key, name varchar(50), role int(20),

country varchar(20), batting_style int(20),bowling_style int(20),dob date);

insert into player values(1,'virat kohli',1,'india',1,1,'1988-11-5');

create table test_career(id int primary key,matches int,runs int,wickets int,batting_avg double,bowling_avg double,batting_sr double,bowling_sr double);

insert into test_career values(1,79,6749,0,53.14,0,57.01,0);

create table odi_career(id int primary key,matches int,runs int,wickets int,batting_avg double,bowling_avg double,batting_sr double,bowling_sr double);

insert into odi_career values(1,239,11520,4,60.31,166.25,93.21,160.25);

create table t20_career(id int primary key,matches int,runs int,wickets int,batting_avg double,bowling_avg double,batting_sr double,bowling_sr double);

insert into t20_career values(1,72,2450,4,50.0,49.5,135.28,36.5);

create table league(lid int primary key, lname varchar(50),lshortname varchar(20));

create table league_players(lid int,tid int,pid int);

insert into league_players values(1,3,2);

create table team(tid int,lid int, tname varchar(50),tshortname varchar(20),primary key(tid,lid));

```
insert into team values(3,1,'Mumbai Indians','MI');
```

6.5.2 Stored Procedures

```
USE `cricket`;

DROP procedure IF EXISTS `updateAvg`;


DELIMITER $$

USE `cricket`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `updateAvg`(pid int, c int)

BEGIN

        declare inn int;

   declare run int;

   declare aveg double;

   if c=1 then

                select innings into inn from test_career where id = pid;

      select runs into run from test_career where id = pid;

      set aveg = run / inn;

      update test_career set batting_avg = aveg where id = pid;

         end if;

   if c=2 then

                select innings into inn from odi_career where id = pid;

      select runs into run from odi_career where id = pid;

      set aveg = run / inn;

      update odi_career set batting_avg = aveg where id = pid;

         end if;

   if c=3 then

                select innings into inn from t20_career where id = pid;

      select runs into run from t20_career where id = pid;

      set aveg = run / inn;
```

```
    update t20_career set batting_avg = aveg where id = pid;
      end if;
END$$

DELIMITER ;
```

6.5.3 Triggers

```
DROP TRIGGER IF EXISTS `cricket`.`player_AFTER_INSERT`;

DELIMITER $$
USE `cricket`$$
CREATE DEFINER=`root`@`localhost` TRIGGER `player_AFTER_INSERT` AFTER
INSERT ON `player` FOR EACH ROW BEGIN
      insert into test_career values (NEW.id,0,0,0,0,0,0,0,0);
  insert into odi_career values (NEW.id,0,0,0,0,0,0,0,0,0);
  insert into t20_career values (NEW.id,0,0,0,0,0,0,0,0,0);
END$$
DELIMITER ;
```

6.4 MySQL

6.4.1 Introduction

To work with data in a database, we must use a set of commands and statements (language) defined by the DBMS software. There are several different languages that can be used with relational databases; the most common is MySQL. The MySQL software delivers a very fast, multithreaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software.

6.4.2 MySQL Server Features

MySQL Server supports a set of features that result in the following benefits:

Ease of installation, deployment, and use:

MySQL Server includes a set of administrative and development tools that improve your ability to install, deploy, manage, and use MySQL Server across several sites.

Scalability:

The same database engine can be used across platforms ranging from laptop computers running Microsoft Windows XP to large, multiprocessor servers running Microsoft Windows 10.

Data Warehousing:

MySQL Server includes tools for extracting and analysing summary data for online analytical processing (OLAP). SQL Server also includes tools for visually designing databases and analysing data using English-based questions.

System integration with another server software:

MySQL Server integrates with e-mail, the Internet, Windows and other software.

6.4.3 Database

A database in MySQL Server consists of a collection of tables that contain data, and other objects, such as views, indexes, stored procedures, and triggers, defined to support activities performed with the data. E.g. The data stored in a database is usually related to a particular subject or process, such as inventory information for a manufacturing warehouse.

MySQL Server can support many databases, and each database can store either interrelated data or data unrelated to that in the other databases. Before we create a database, it is important to understand the parts of a database and how to design these parts to ensure that the database performs well after it is implemented.

6.4.4 Normalization Theory

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. It divides larger tables to smaller tables and links them using relationships. A relation is said to be in a particular form if it satisfies certain specified constraints.

To decide a suitable logical structure for given database design the concept of normalization, which are briefly described below.

- **1st Normal Form (1 NF):** A relation is said to be in 1 NF if and only if all unaligned domains contain one value only. That is the fields of an n-set should have no group items and no repeating groups.
- **2nd Normal Form (2 NF):** A relation is said to be in 2 NF if and only if it is in 1 NF and every non key attribute is fully dependent on primary key. This normal takes care of functional dependencies on non-key attributes.
- **3rd Normal Form (3 NF):** A relation is said to be in 3 NF if and only if it is in 2 NF and every non key attribute is non transitively dependent on the primary key. This normal form avoids the transitive dependencies on the primary key.
- **Boyce code Normal Form (BCNF):** This is a stronger definition than that of NF. A relation is said to be in BCNF if and only if every determinant is a Candidate key.
- **4th Normal Form (4 NF):** A relation is said to be in 4 NF if and only if whenever there exists a multi valued dependency in a relation say A- >>B then all of the relation is also functionally dependent on A (i.e. A- >X for all attributes x of the relation.).
- **5th Normal Form (5 NF) OR Projection Join Normal Form (PJNF):** A relation R is in 5 NF if and only if every join dependency in R is implied by the candidate key on R . A relation can't be non-loss split into two tables but can be split into three tables. This is called Join Dependency.

This feature is incorporated into our database. All the tables in the database are normalized till 3 NF as mentioned below:

Table 1.Player

Id	Name	Role	Country	BattingStyle	BowlingStyle	Dob
1	Virat Kohli	Batsman	India	Right handed	Right pacer	1988-11-5
2	Rohit Sharma	Batsman	India	Right handed	Right spinner	1987-04-30

Table 2.Test Career

Id	Matches	Runs	Wickets	BattingAvg	BowlingAvg	BattingSR	BowlingSR	Innings
1	79	6749	0	53.14	0	57.01	0	130
2	27	1585	2	39.62	101.0	55.02	167.0	55

Table 3.League

Lid	Lname	Lshortname
1	Indian Premier League	IPL
2	Big Bash League	BBL
3	Caribbean Premier League	CPL

Table 4.Team

Tid	Lid	Tname	Tshortname
1	1	Royal Challengers	RCB
2	1	Mumbai Indians	MI

Table 5.LeaguePlayer

Lid	Tid	Pid
1	1	1
1	2	2

Chapter 7

Testing

7.1 Introduction

Testing is a process of executing a program with the intent of finding an error. It is a crucial element of software quality assurance and presents best review of specifications, design and the code.

System Testing is an important phase. It discovers an unseen anomaly for the software. Thus, a series of testing are performed on the proposed system before the system is ready for user acceptance testing.

A good test case is one that has a high probability of finding an as undiscovered error. A successful test is one that uncovers an as undiscovered error.

7.2 Testing Objectives

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an undiscovered error.

7.3 Testing Principles

- All tests should be traceable to end user requirements.
- Tests should be planned long before testing begins.
- Testing should begin on a small scale and progress towards resting in large.
- Exhaustive testing is not possible.
- To be most effective testing should be conducted by an independent third party.

The primary objective for test case design is to derive a set of tests than has the highest livelihood for uncovering defects in software. To accomplish this objective, two different categories of test case design techniques are used. They are:

- White box Testing
- Black box Testing

White-box Testing:

White box testing focus on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed.

Black-box Testing:

Black box testing is designed to validate functional requirements without regard to the internal workings of a program. Black box testing mainly focuses on the information domain of the software, deriving test cases by partitioning input and output in a manner that provides through test coverage. Incorrect and missing functions, interface errors. errors in data structures, error in functional logic are the errors falling in this category.

7.4 Testing Strategies

A strategy for software testing must accommodate low-level tests to verify all small source code segment as well as high-level tests that validate major system functions against customer requirements. Test Strategy is also known as test approach which defines how testing would be carried out.

Test approach has two techniques:

Proactive - An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.

Reactive - An approach in which the testing is not started until after design and coding are completed.

7.5 Testing Fundamentals

Testing is a process of executing program with the intent of furling error. A good test case is one that has high probability of finding an undiscovered error. If testing is conducted successfully it uncovers the errors in the software. Testing cannot show the absence of errors, it can only show the presence of it.

7.6 Testing Information Flow

Information flow for testing flows the pattern. Two class of input are provided to test the process. The software configuration includes a software requirements specification, design specification and source code.

Test configuration includes test plan, test cases and test tools. Tests are conducted and all the results are evaluated. Then test results are compared with expected results. When erroneous data are uncovered, an error is implied and debugging commences.

7.6.1 Unit Testing

UNIT TESTING is a level of software testing where individual units/components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

7.6.2 Integration Testing

INTEGRATION TESTING is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

7.6.3 System Testing

SYSTEM TESTING is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. According to ISTQB, it is the process of testing an integrated system to verify that it meets specified requirements.

7.6.4 Acceptance Testing

ACCEPTANCE TESTING is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

According to ISTQB, it is formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

7.7 Test Cases

Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed. Using White-Box testing method, the software engineer can drive test cases that:

- Guarantee the logical decisions on their true and false side.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structure to assure their validity.

The test case specification for system testing has to be submitted for review before system testing commences.

7.8 Project Testing

Under the project, the following testing has been performed:-

Unit Testing:

Unit Testing has been implemented in the front-end and back-end of the system. Each component is found to fully functional and working as per requirements.

Integration Testing:

Integration Testing has been implemented by combining the front-end and back-end with PHP. The integrated model is found to fully functional and working as per requirements.

System Testing:

System Testing has been implemented after the completion of the whole system and is found to be working properly. The requirement set are met completely.

Acceptance Testing:

Acceptance Testing has been implemented after providing the system to different users for testing. The system is found to be working properly on the user systems without any errors/faults.

Chapter 8

Results

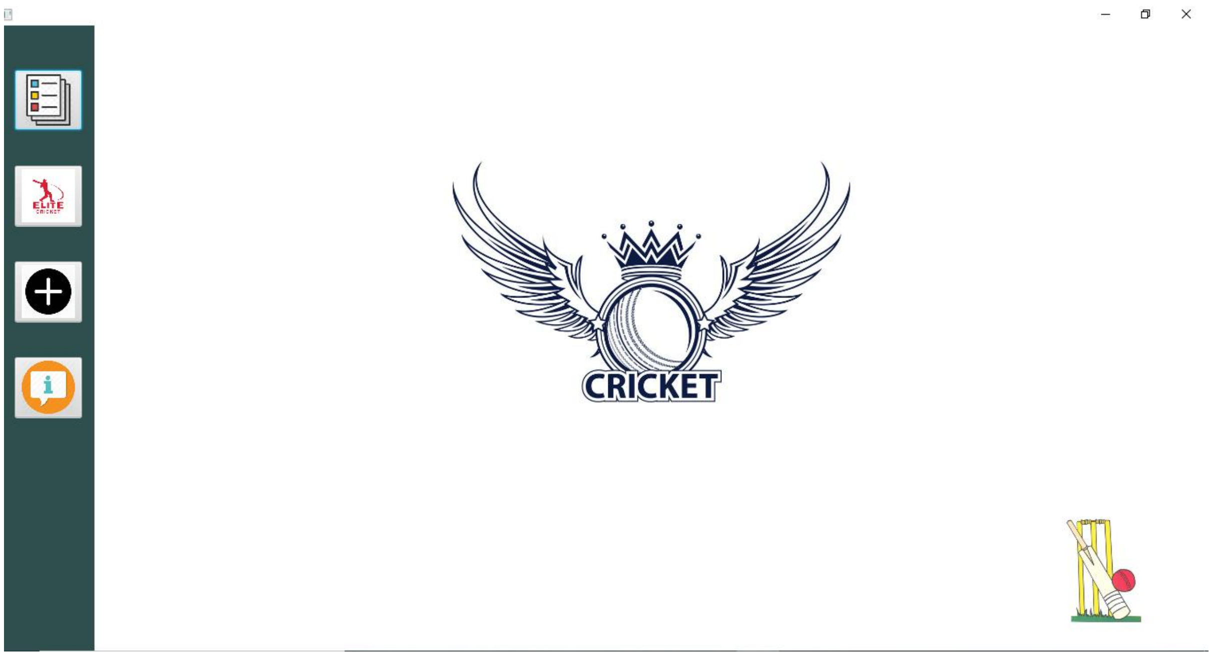
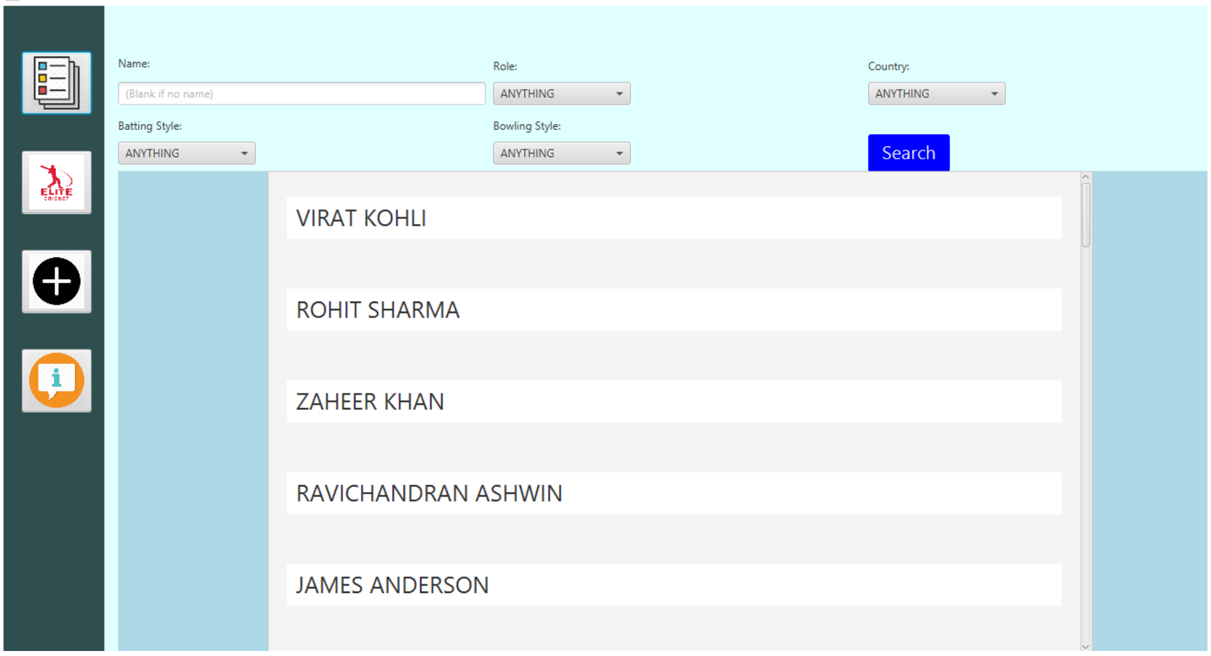


Fig. 8.1 Screenshot1



Name:

(Blank if no name)

Role: ANYTHING

Country: ANYTHING

Batting Style: ANYTHING

Bowling Style: ANYTHING

Search

VIRAT KOHLI

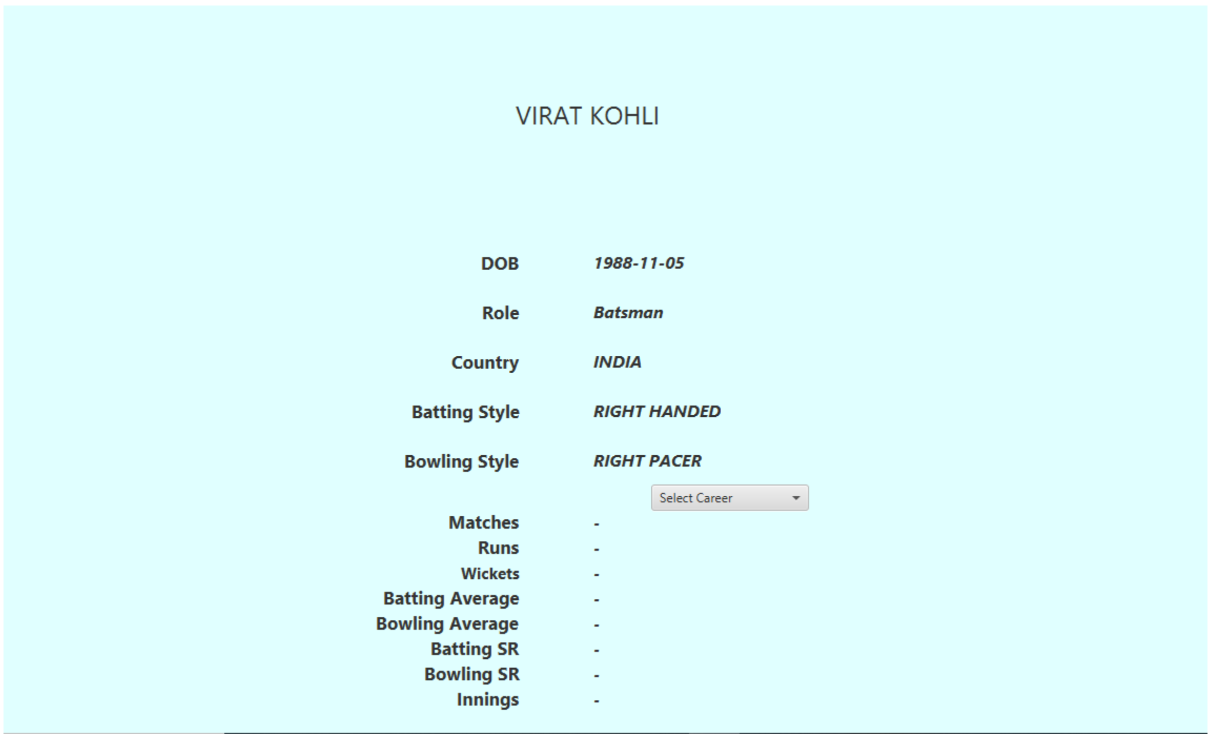
ROHIT SHARMA

ZAHEER KHAN

RAVICHANDRAN ASHWIN

JAMES ANDERSON

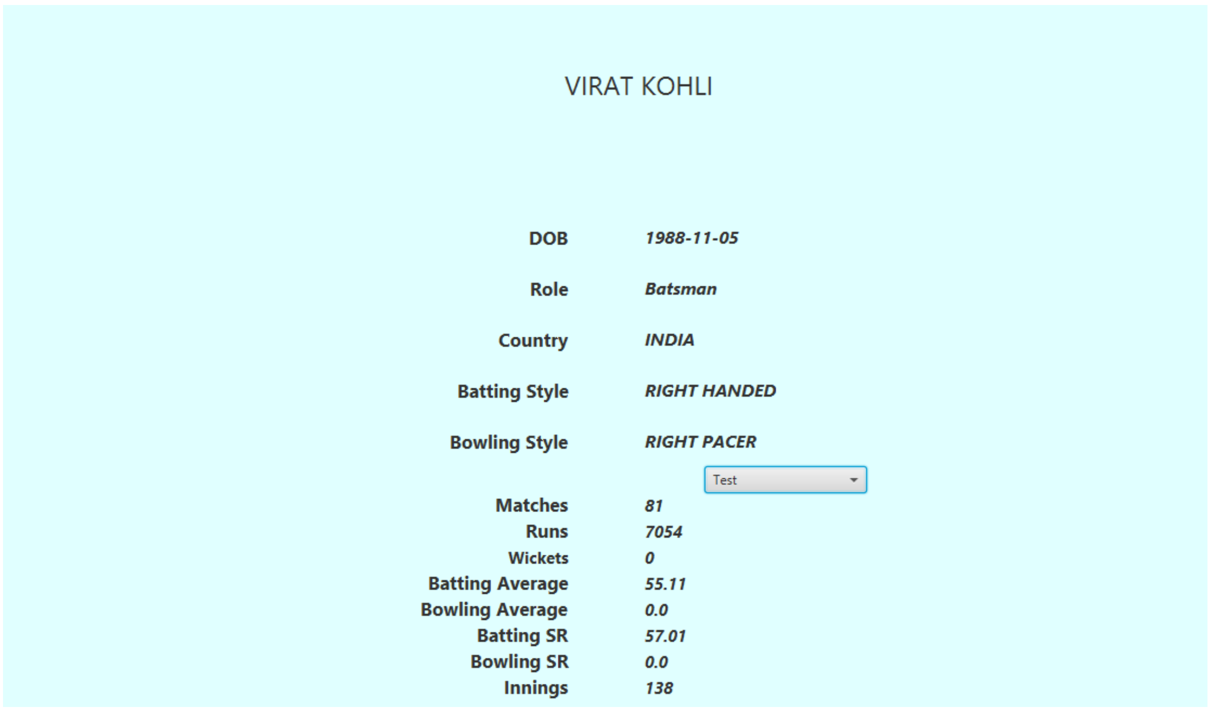
Fig. 8.2 Screenshot2



VIRAT KOHLI

DOB	1988-11-05
Role	Batsman
Country	INDIA
Batting Style	RIGHT HANDED
Bowling Style	RIGHT PACER
	Select Career
Matches	-
Runs	-
Wickets	-
Batting Average	-
Bowling Average	-
Batting SR	-
Bowling SR	-
Innings	-

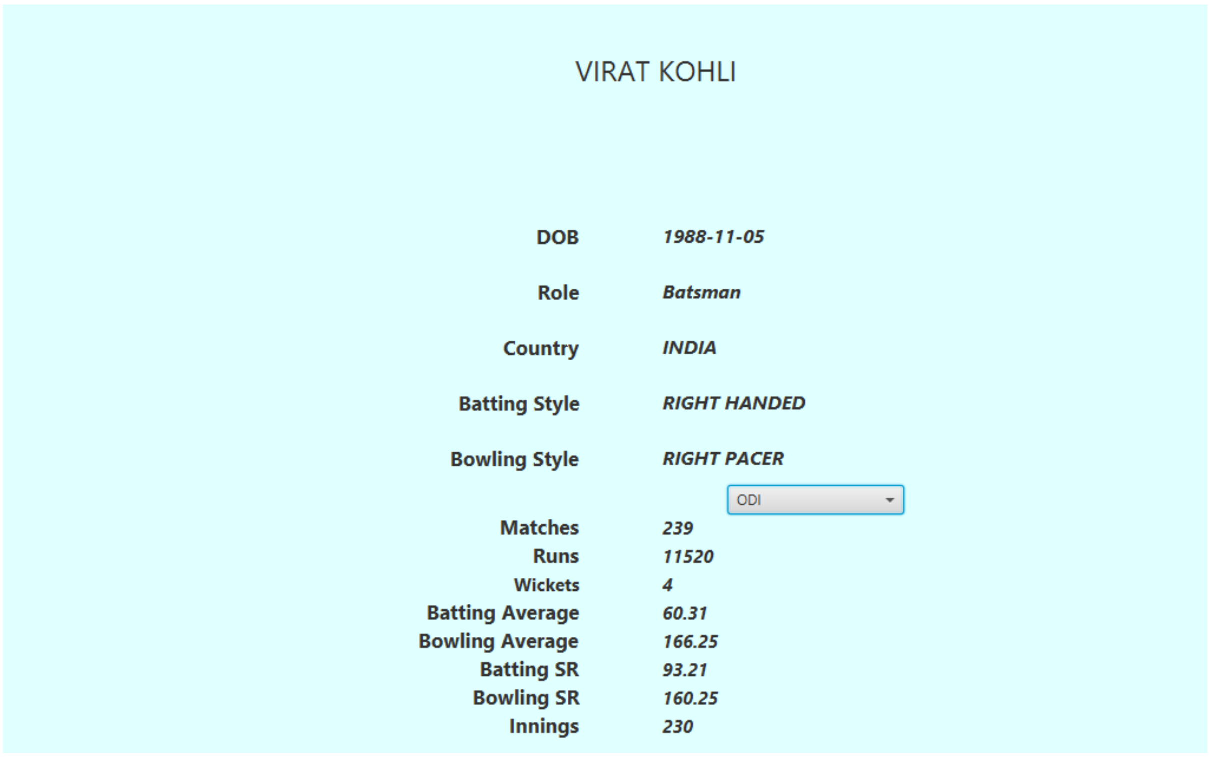
Fig. 8.3 Screenshot3



VIRAT KOHLI

DOB	1988-11-05
Role	Batsman
Country	INDIA
Batting Style	RIGHT HANDED
Bowling Style	RIGHT PACER
	Test
Matches	81
Runs	7054
Wickets	0
Batting Average	55.11
Bowling Average	0.0
Batting SR	57.01
Bowling SR	0.0
Innings	138

Fig. 8.4 Screenshot4



VIRAT KOHLI

DOB	1988-11-05
Role	Batsman
Country	INDIA
Batting Style	RIGHT HANDED
Bowling Style	RIGHT PACER
	ODI
Matches	239
Runs	11520
Wickets	4
Batting Average	60.31
Bowling Average	166.25
Batting SR	93.21
Bowling SR	160.25
Innings	230

Fig. 8.5 Screenshot5

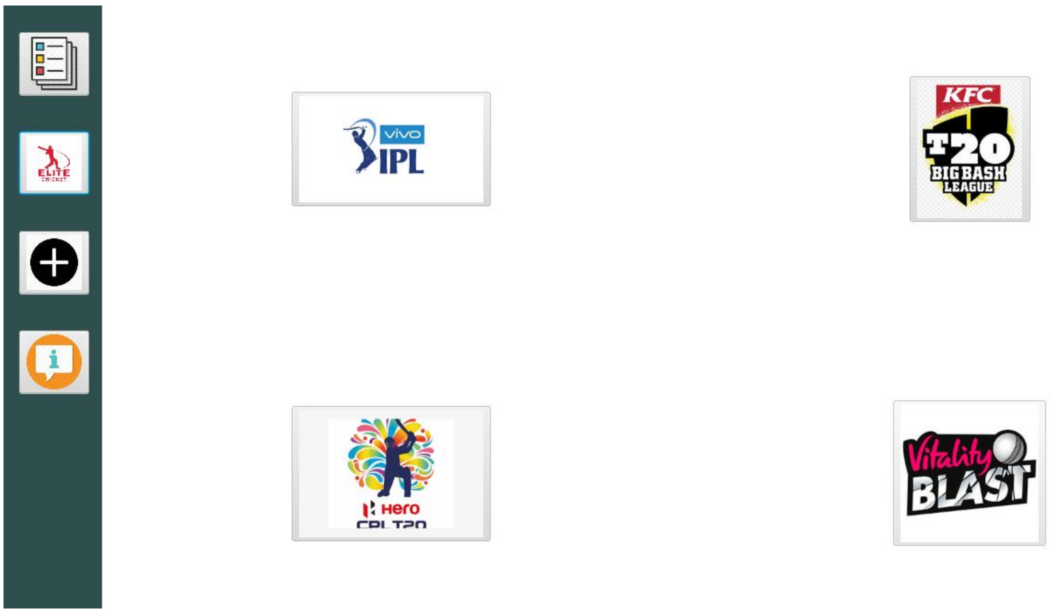


Fig. 8.6 Screenshot6

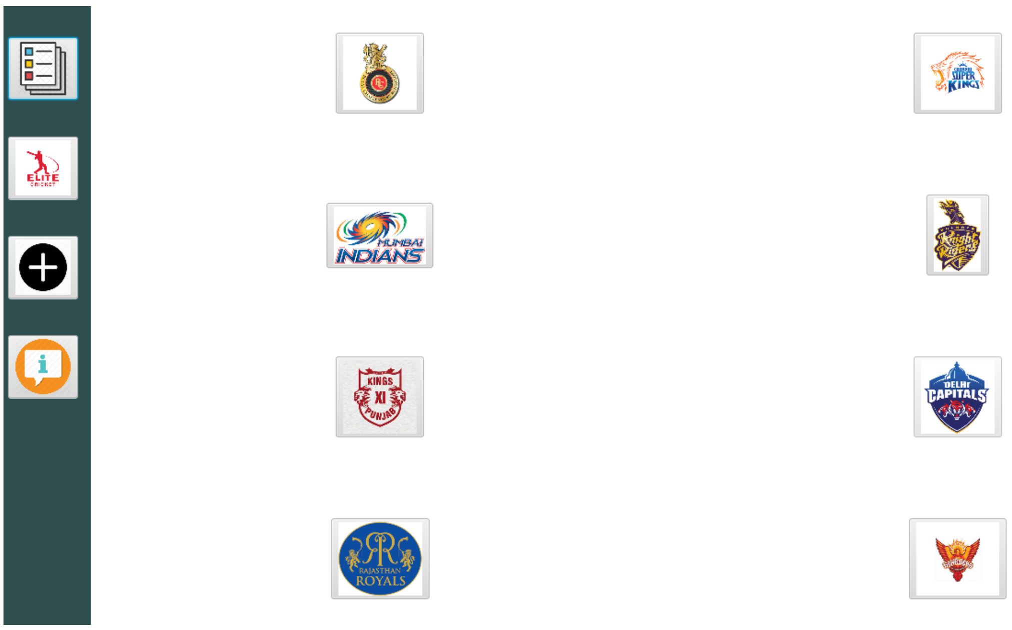
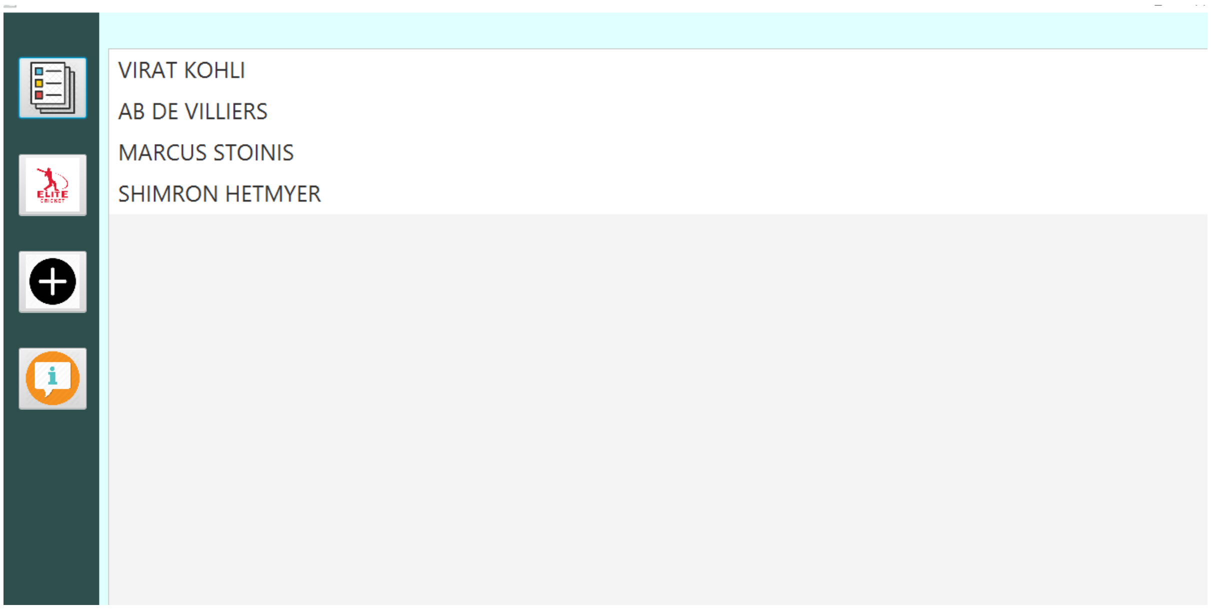


Fig. 8.7 Screenshot7



VIRAT KOHLI
AB DE VILLIERS
MARCUS STOINIS
SHIMRON HETMYER

Fig. 8.8 Screenshot8

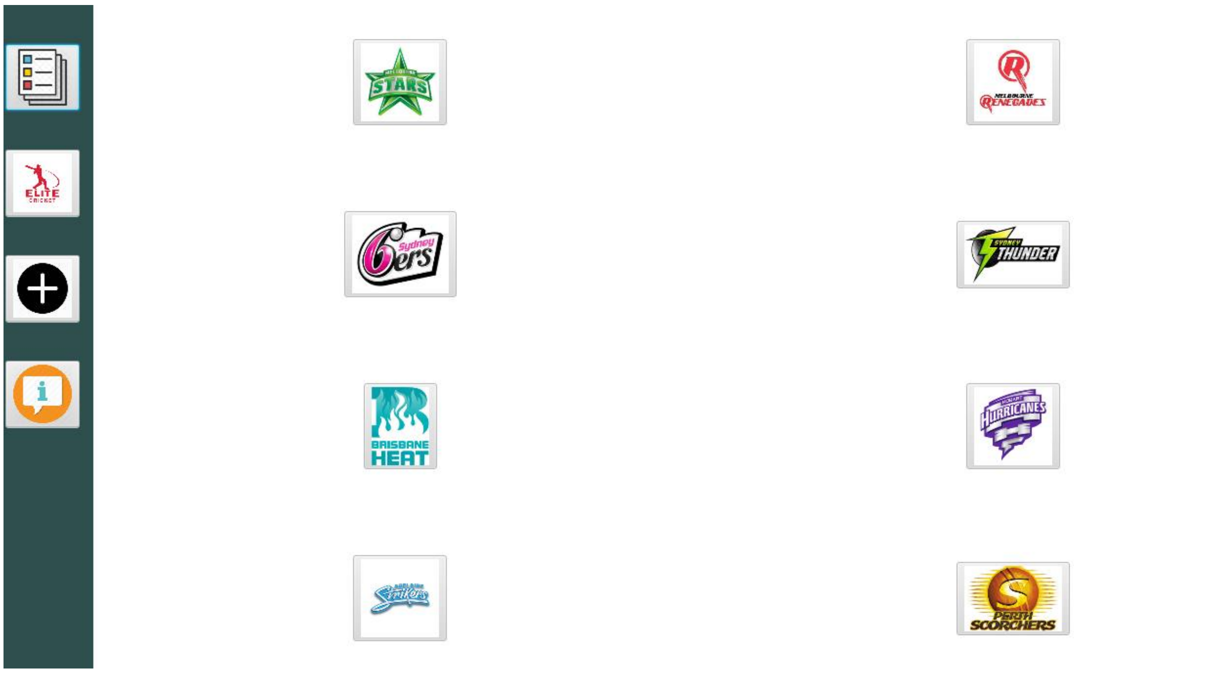


Fig. 8.9 Screenshot9

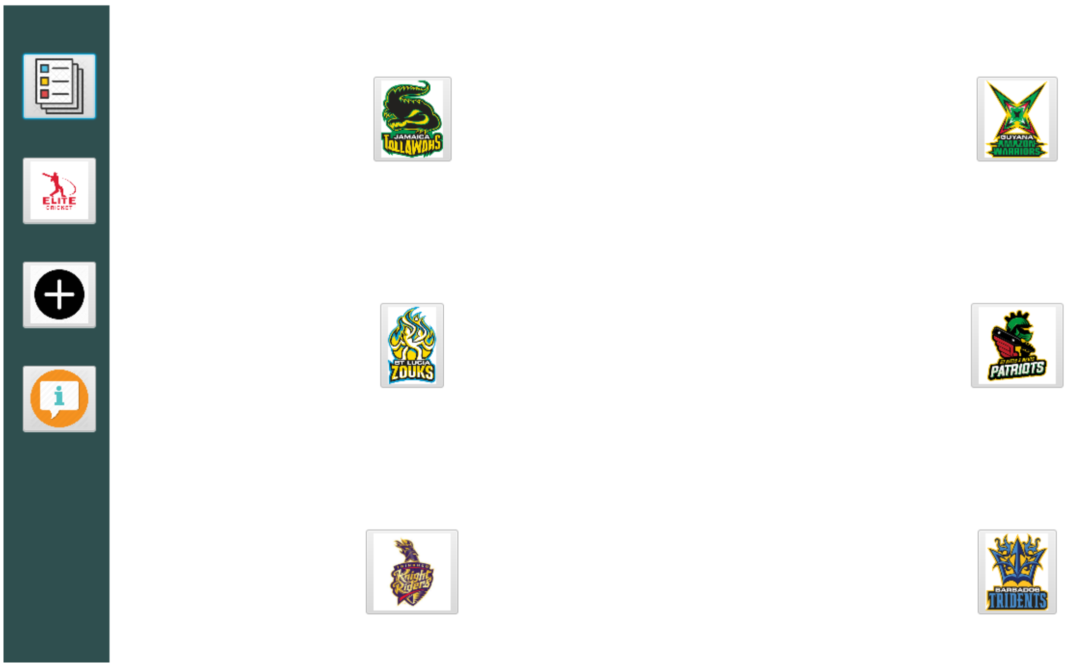
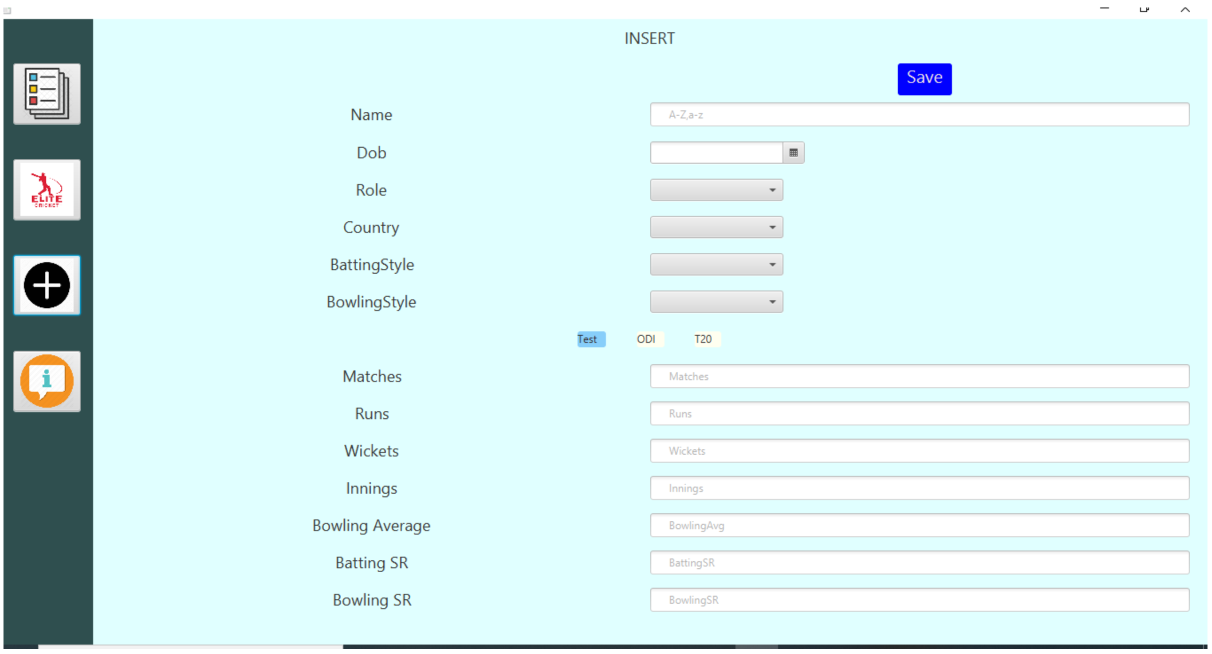


Fig.8.10 Screenshot10



INSERT

Save

Name	A-Z,a-z
Dob	
Role	
Country	
BattingStyle	
BowlingStyle	

Test ODI T20

Matches	Matches
Runs	Runs
Wickets	Wickets
Innings	Innings
Bowling Average	BowlingAvg
Batting SR	BattingSR
Bowling SR	BowlingSR

Fig.8.11 Screenshot 11

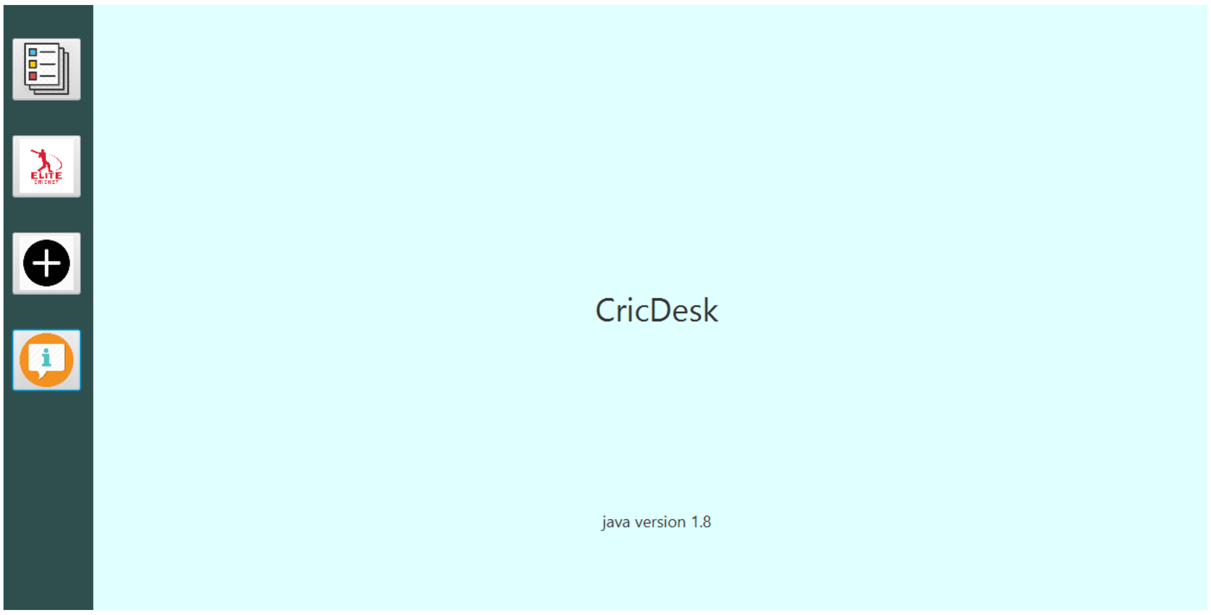
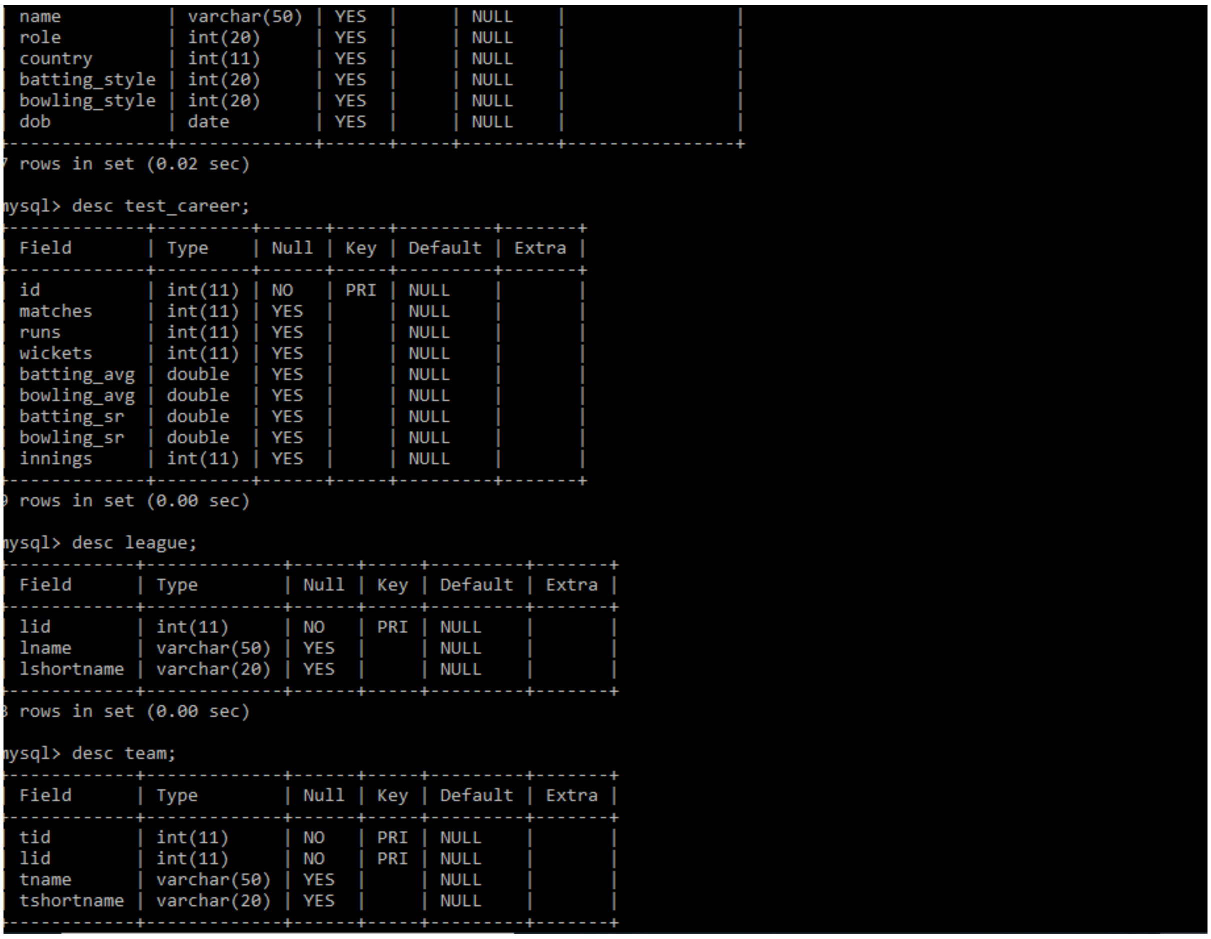


Fig.8.12 Screenshot 12



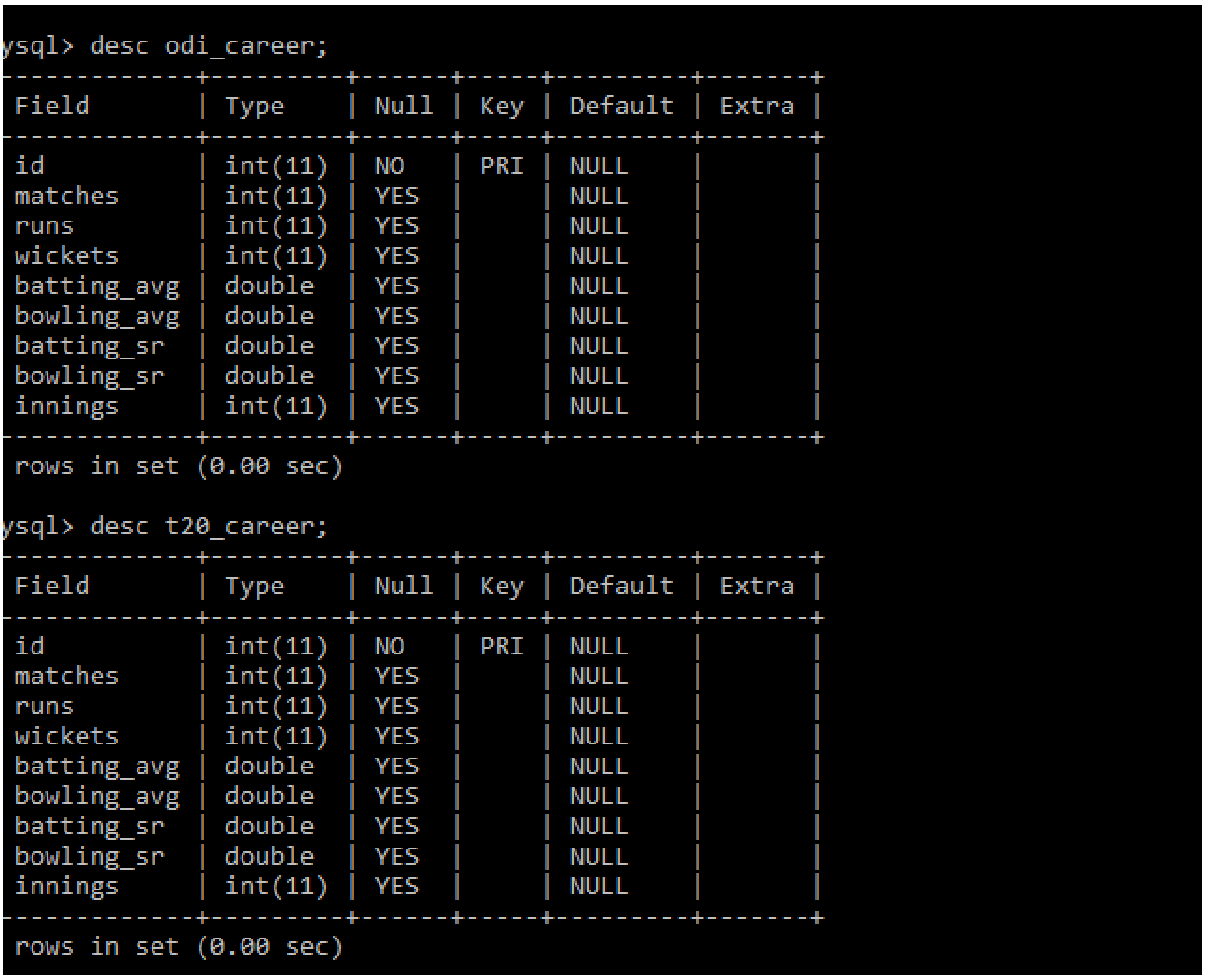
```
 name          | varchar(50) | YES  |     | NULL    |                |
 role          | int(20)     | YES  |     | NULL    |                |
 country       | int(11)     | YES  |     | NULL    |                |
 batting_style | int(20)     | YES  |     | NULL    |                |
 bowling_style | int(20)     | YES  |     | NULL    |                |
 dob           | date        | YES  |     | NULL    |                |
---------------+-------------+------+-----+---------+----------------+
7 rows in set (0.02 sec)

mysql> desc test_career;
-------------+---------+------+-----+---------+-------+
 Field       | Type    | Null | Key | Default | Extra |
-------------+---------+------+-----+---------+-------+
 id          | int(11) | NO   | PRI | NULL    |       |
 matches     | int(11) | YES  |     | NULL    |       |
 runs        | int(11) | YES  |     | NULL    |       |
 wickets     | int(11) | YES  |     | NULL    |       |
 batting_avg | double  | YES  |     | NULL    |       |
 bowling_avg | double  | YES  |     | NULL    |       |
 batting_sr  | double  | YES  |     | NULL    |       |
 bowling_sr  | double  | YES  |     | NULL    |       |
 innings     | int(11) | YES  |     | NULL    |       |
-------------+---------+------+-----+---------+-------+
9 rows in set (0.00 sec)

mysql> desc league;
------------+-------------+------+-----+---------+-------+
 Field      | Type        | Null | Key | Default | Extra |
------------+-------------+------+-----+---------+-------+
 lid        | int(11)     | NO   | PRI | NULL    |       |
 lname      | varchar(50) | YES  |     | NULL    |       |
 lshortname | varchar(20) | YES  |     | NULL    |       |
------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc team;
------------+-------------+------+-----+---------+-------+
 Field      | Type        | Null | Key | Default | Extra |
------------+-------------+------+-----+---------+-------+
 tid        | int(11)     | NO   | PRI | NULL    |       |
 lid        | int(11)     | NO   | PRI | NULL    |       |
 tname      | varchar(50) | YES  |     | NULL    |       |
 tshortname | varchar(20) | YES  |     | NULL    |       |
------------+-------------+------+-----+---------+-------+
```

Fig.8.13 Screenshot 13



```
ysql> desc odi_career;
+------------+---------+------+-----+---------+-------+
 Field       | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
 id          | int(11) | NO   | PRI | NULL    |       |
 matches     | int(11) | YES  |     | NULL    |       |
 runs        | int(11) | YES  |     | NULL    |       |
 wickets     | int(11) | YES  |     | NULL    |       |
 batting_avg | double  | YES  |     | NULL    |       |
 bowling_avg | double  | YES  |     | NULL    |       |
 batting_sr  | double  | YES  |     | NULL    |       |
 bowling_sr  | double  | YES  |     | NULL    |       |
 innings     | int(11) | YES  |     | NULL    |       |
+------------+---------+------+-----+---------+-------+
 rows in set (0.00 sec)

ysql> desc t20_career;
+------------+---------+------+-----+---------+-------+
 Field       | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
 id          | int(11) | NO   | PRI | NULL    |       |
 matches     | int(11) | YES  |     | NULL    |       |
 runs        | int(11) | YES  |     | NULL    |       |
 wickets     | int(11) | YES  |     | NULL    |       |
 batting_avg | double  | YES  |     | NULL    |       |
 bowling_avg | double  | YES  |     | NULL    |       |
 batting_sr  | double  | YES  |     | NULL    |       |
 bowling_sr  | double  | YES  |     | NULL    |       |
 innings     | int(11) | YES  |     | NULL    |       |
+------------+---------+------+-----+---------+-------+
 rows in set (0.00 sec)
```

Fig 8.14 Screenshot 14

Conclusion

This project provides an ease of management of cricketers' professional and personal Information.The scope of project can easily be increased to other sports or broaden the information that it holds on cricketers.

.

FUTURE ENHANCEMENTS:

The application has many uses. Due to time limitations we could not make it online. But since we modularized it, we can make it an interactive online application by connecting to free cricket API available. If we want to do this, then there is no stagerring changes, only FX controller should be changed. The DAO classes will remain as it is.

We can also remove database from the user and save it in different place and make it modularized, so that more than one user can have same database. We can create different accounts for different group of users.

This application can be used by cricket teams to theorize the way player plays by seeing into the stats. This strategy has been already tried and tested in football.

References

The following books and websites were referred during planning and execution phase of the project:

Books:

Java Complete Reference and JavaFX

Herbert Schildt

MySQL Complete Reference

Vikram Vaswani

Websites:

https://www.youtube.com › channel

https://www.youtube.com › user › ProgrammingKnowledge

Literature Survey -

https://shodhganga.inflibnet.ac.in/bitstream/10603/183997/7/07_chapter2.pdf

Maintenance Management: Literature Review and Directions

Amik Garg, S.G. Deshmukh

Journal of Quality in Maintenance Engineering

ISSN: 1355-2511 Publication date: 1 July 2006

Feasibility Study – by Mohammed Mohsin

https://www.scribd.com/document/209127891/Feasibility-Study

UML, ER, Usecase Definition – by MuzaFar

https://www.scribd.com/document/360219168/Project-Report

Test Strategy Definition

https://www.tutorialspoint.com/software_testing_dictionary/test_strategy.htm

Unit Testing

http://softwaretestingfundamentals.com/unit-testing/

Integration Testing

http://softwaretestingfundamentals.com/integration-testing/

System Testing

http://softwaretestingfundamentals.com/system-testing/

Acceptance Testing

http://softwaretestingfundamentals.com/acceptance-testing/