# **CASE STUDY: BUS TICKET BOOKING SYSTEM**

## **The Problem**

ABC Travel Corporation plans to automate its ticket booking system. Currently ABC Travel Corporation operates in 3 routes with 3 different buses. Anyhow, the travel manager should be able to do the following in the application.

1. Add as many numbers of buses along with routes, seating capacity and ticket price.

2. View all buses with details

When passenger calls in to check the availability, Travel manager gets the source and destination from the passenger and checks the availability of bus. If available, passenger’s name and age are collected. Bus and seat number are allocated to the passenger. If all the seats are allocated, passenger will not be given a ticket.

Travel manager should be able to view the passengers for a given bus number.

## **Objectives**

The principle functionalities of the Bus Ticket Booking system are as follows:

1. Passengers will be able to find whether seats are available on a bus plying on a specific route on particular day/time.
2. Depending on the availability of seats passengers will be able to reserve in advance seats on the requisite bus.
3. Details about all the passengers will be maintained by the travelling company.
4. Seat reservation details are maintained in the database when seat booking is successful.
5. A report of all the passengers travelling on a particular bus number can be generated.
6. New buses can be added to the existing routes.
7. Details of all the existing buses can be viewed.

## **Preliminary Ideas**

In our bus ticket booking system we have taken some assumptions which are as follows:

1. Suppose a bus B1 is going from a place R1 to R2, we have assumed that there will always be a different bus B2 which will travel from R2 to R1.
2. A passenger will always contiguous seats in the bus, in other words multiple seats will be allocated to a passenger in a serial fashion only.
3. A passenger cannot select his/her own seats; he/she will be allocated seats as per next seat available.
4. There is no provision for cancellation of the reservation.
5. The travel manager can only see the number of seats available and not the individual seat numbers in a bus.
6. The travel manager can see the list of buses in the company and its associated routes, time of departure.

## **The Feasibility Study**

As the initial investigation is completed, it leads to a more detailed investigation of the system. The conclusions of the initial study become the input for the detailed study. We can also refer to the feasibility study as the detailed study or detailed investigation. Feasibility study is called like because as in the first phase, we just check briefly about the problems related to the old system, & the need of the new system. So, in this phase, that initial survey is further expanded to a more detailed feasibility study.

The tasks performed during this phase were as follows:

(a) The users demonstrable needs are fulfilled.

(b) The availability of resources was being checked.

(c) The estimation required for the resources were achieved.

(d) The impact of the system on the organization has been check by placing a model in it.

## **Requirement Analysis and Specification**

**Objective to be fulfilled:**

(a) Development of software in the given time.

(b) To create an effective and efficient application.

**User Requirements:**

(a) The application should provide a user friendly environment.

(b) The application should be easily understandable and reliable.

(c) The application should fulfill all essential facilities.

(d) The software being built must provide platform independent application.

**Requirement Determination Techniques & System Analysis Method:**

(a) User communication

(b) Team discussion

(c) Analysis of existing system

(d) Study of old projects and records.

## Entity Relationship Diagram

Conceptual modeling is a very important part in designing a proper database application. Here the term database application refers to a software product that has a central database containing data and some associated programs that operate on the data in the database to give desired results of our queries. In the Bus Ticket Booking problem we are representing conceptual model of the problem by the help of ER data modeling. An entity-relationship (ER) diagram is a specialized graphical representation that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

For our database application, consider a BUS SYSTEM database which keeps track of the buses and their corresponding routes, a finite number of source(s) and destination(s) that make up the various routes, the number of seats that are reserved in each bus, conversely the number of seats which are still available in each bus. The database also keeps track of the passengers name and address who have reserved seats in one or more buses plying on different routes.

In our BUS SYSTEM database we have three entities: BUS, ROUTE and RESERVATION.

1. BUS: For each bus, the database maintains information on the Route on which the bus plies , total number of seats in the bus and ticket price of each seat of the bus. Thus the specific attributes of the BUS entity are: [BUS\_ID], [ROUTE], [TOTAL\_SEATS], [TICKET\_PRICE].
2. ROUTE: This entity has attributes [ROUTE\_ID],[SOURCE],[DESTINATION]. The database maintains all the source-destination pairs on which the travel company provides their bus service.
3. RESERVATION: Our database maintains the details like name, age of each and every passenger who has reserved seat(s) in any one or more buses. In other words the database maintains a record of the name and age of all the passengers who has travelled in buses under this company. The specific attributes of this entity are [RS\_ID], [BUS], [ROUTE], [NAME], [AGE], [SEAT\_NO].

Now these three entities are related to each other by two different relationships: RUNS\_ON, BOOKING.

1. RUNS\_ON: The entities BUS and ROUTE are related by the RUNS\_ON relationship. We know that only one bus can ply on only one route but in the same time there can be multiple different buses plying on the same route, thus the relationship RUNS\_ON is N:1 relationship, which can be seen from the ER diagram. Again the from the ER diagram we can see a **Total Participation** of entity BUS on relationship RUNS\_ON, i.e. if the policy of the travel company states that *every* bus must ply on some route , then a bus can only exist if it participates in at least one RUNS\_ON relationship instance. Thus the participation of BUS in RUNS\_ON is called total participation, meaning every entity in the total set of BUS entities must be related to a ROUTE entity via RUNS\_ON.
2. BOOKING: This is a 1:N relationship that exists between RESERVATION and ROUTE entities. As we have assumed that one passenger can only travel in only one route at a given time , thus each tuple in the reservation relation can have only one route description, on the other hand the there can be multiple reservations on the same route at any instant of time. Thus there can be N number of reservations on a single route and there can be only one route in a particular reservation.

The ER diagram is as follows:

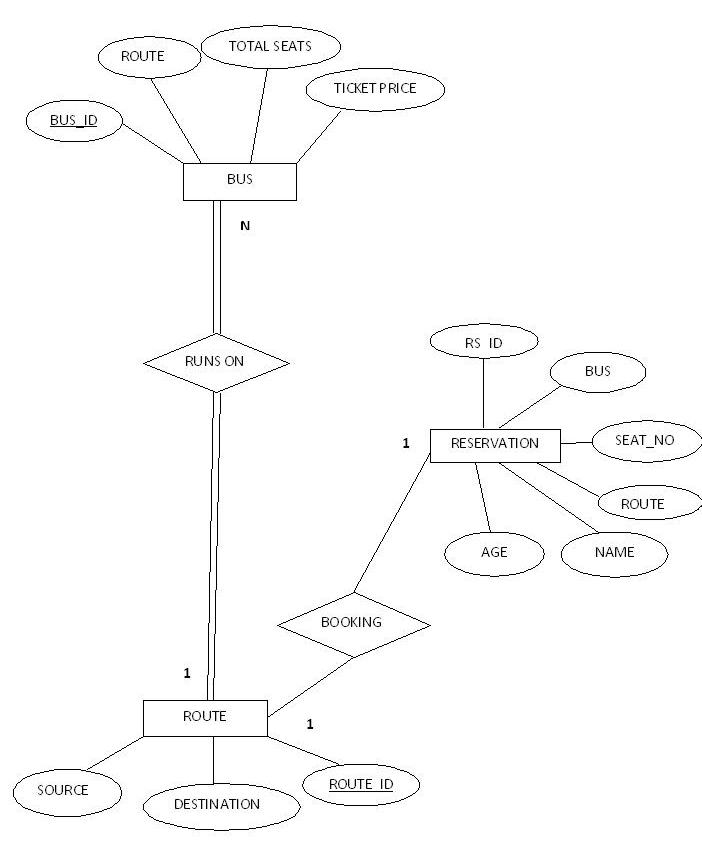


Figure: The ER Diagram

## Database Design

**Relational schemas and functional dependencies:**

1. **BUS ( BUS\_ID , ROUTE\_ID , TOTAL\_SEATS , TICKET\_PRICE )**

The bus table records the details of the buses in the travel company. The BUS\_ID is the primary key in this table. This table has two functional dependencies:

1. *BUS\_ID 🡪 { TOTAL\_SEATS , TICKET\_PRICE }*

TOTAL\_SEATS, TICKET\_PRICE are entirely dependent on the BUS\_ID as each bus has its own capacity and depending on the quality of service provides it has its tickets priced.

1. *ROUTE\_ID 🡪 BUS\_ID*

Again BUS\_ID is dependent on the ROUTE\_ID as a bus can ply over only one route.

1. **RESERVATION ( RS\_ID , BUS\_ID , SEAT\_NO , NAME , AGE,BOOKING\_DATE)**
2. *RS\_ID 🡪{BUS\_ID, SEAT\_NO, NAME, AGE, BOOKING\_DATE}*

The reservation table records the reservations of seats made by the passengers daily. The RS\_ID is the primary key in this table and all other attributes are dependent on it. Each tuple contains the information on the basis of which the tickets are issued to the passengers.

1. **ROUTE ( SOURCE , DESTINATION , ROUTE\_ID )**
2. *ROUTE\_ID 🡪{SOURCE,DESTINATION}*

The route table contains all the routes in which the buses travel. Each route has a particular ROUTE\_ID which is the primary key in the table. All the other attributes are dependent on it.

## DATA FLOW DIAGRAM

An analytical flow that shows the flow of information and the transforms that are applied as data moves from input to output is called Data Flow Diagram. The Data Flow diagram is used to represent a system or software at any level of abstraction. The level 0 DFD also called *context diagram* represents the entire software as a single bubble with input and output indicated by incoming and outgoing arrows respectively. A level 0 DFD also shows the external entities with which a software interacts. The level 0 DFD is then further partitioned into level 1 and so on. The partitioning is done to represent additional processes and data flow paths with more details. Each of the processes represented in higher levels of the DFD is a sub function of the entire system depicted in the context diagram.

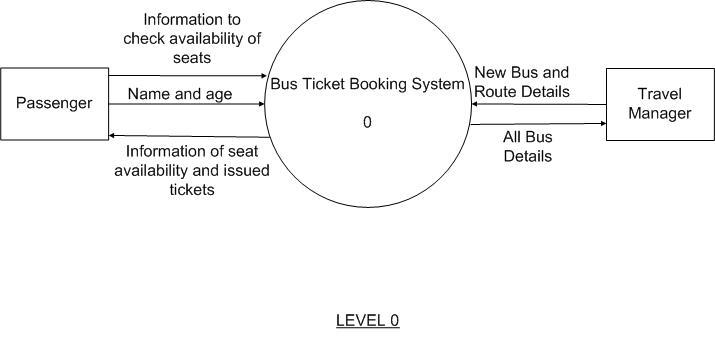


Figure: The Data Flow Diagram

The Context Diagram shows the Bus Ticket Booking System represented as a single module. The data values that are given to the system as input comprise of two types. First of all the new bus and route details can be the input to the system to add as many buses along with routes, seating capacity and ticket prices which is given by the travel manager. Secondly, a passenger may call in to enquire in which case the travel manager gets the source and destination from the passenger and check for availability of buses. If the seats are available on those buses then the name and age of the passengers are taken and ticket is issued. If the seats are not available then the passenger is informed of it. Apart from this the system also gives the details of all the buses present in the database.

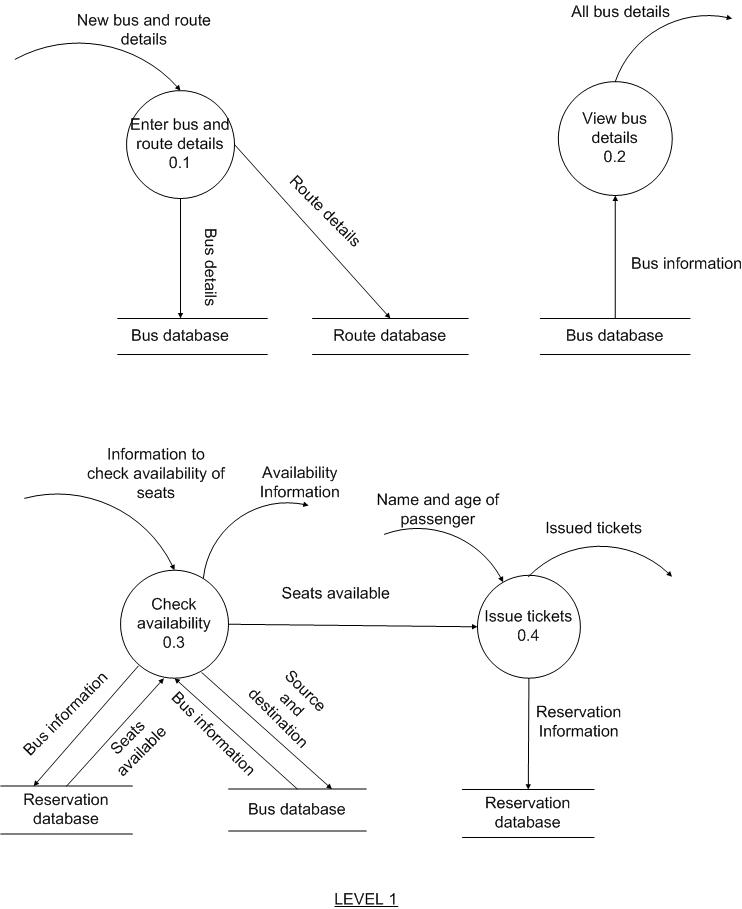


Figure: Level 1 DFD

The Level 1 of the Data Flow Diagram shows the Context Diagram broken down into four modules. The *Enter Bus And Route Details* module takes the new bus and route details as input and makes a new entry in the bus and route database. This module handles the task to be performed by the travel manager when he wants to add new buses and routes to the database along with the seating capacity and ticket prices. The *View Bus Details* module takes all the details of all buses from the bus database and displays it. This module shows all the bus details to be viewed by the travel manager. The *Check Availability* module takes the source and destination supplied by the passenger and looks up in the bus database for available buses in the specified route. If such buses exist then it checks if seats are available in those buses by checking the reservation database. If seats are available then it passes the number of seats available to the *Issue Tickets* which in turn issues tickets after taking the name and age of the passenger and makes an entry in the reservation database.

# CODES

## Main Form:

The Main Form is the MDI Form from which the other forms can be accessed. The above diagram shows the functioning of an MDI form with the selected form being highlighted.

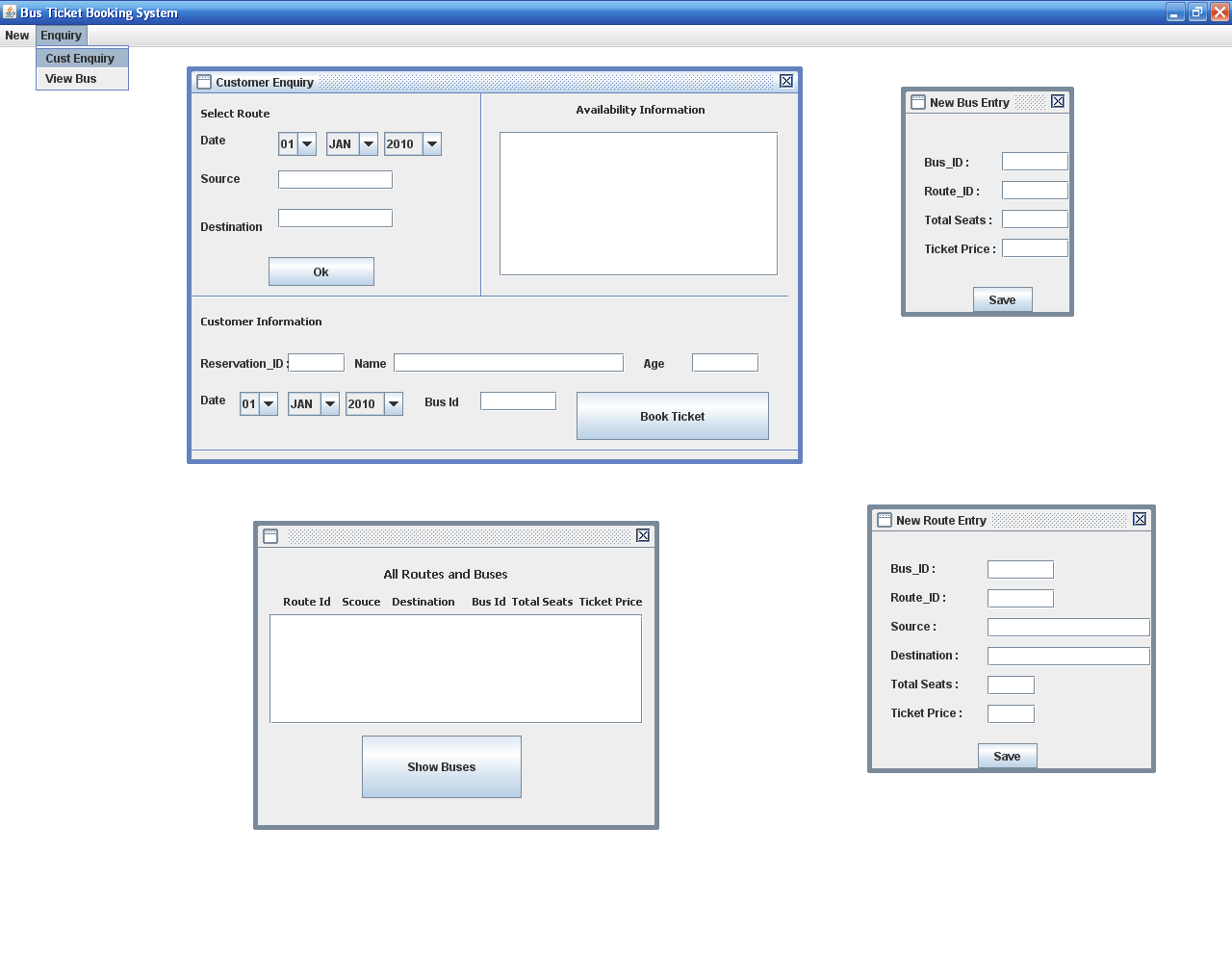
****

Figure: MDI Form showing all other forms

## New Bus Entry form:

The New Bus Entry Form performs the task of making entry of new buses. It is used to add buses on the existing routes. To make an entry the travel manager needs to know the existing route ids and new buses can be easily added on that route. The following code performs this task as explained.

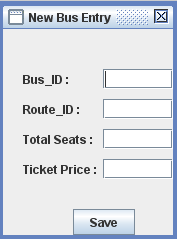


Figure: New Bus Entry Form

## Code Snippets:

Below is the code behind the ‘Save’ Button. When ‘Save’ button is pressed the following code is triggered to show the availability information.

First we make a connection with the database using the following code:

Class.forName ("oracle.jdbc.driver.OracleDriver");

Now we write the connection string that actually prepares and maintains the connection between the java application and the Oracle Listener Service through port 1521 at the localhost using Oracle thin client JDBC driver, where SCT is the database name and username and password are the respective name and password of the database user.

con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:SCT","username","password");

Next we create a statement object as follows:

stmt = con.createStatement();

Next we execute the query which is supposed to perform the task of inserting a record into the database.

stmt.executeQuery("insert into bus values('" + Bus\_ID\_TextField.getText()+"', '"+Route\_ID\_TextField.getText()+"',"+Ticket\_Price\_TextField.getText()+","+Total\_Seats\_TextField.getText()+")");

This statement takes the required input from the corresponding text fields and complete completes the statement which is then executed as a complete query. The query inserts a row into the table bus.

## New Route Entry:

The New Route Entry Form performs the task of making new entry of routes and the buses which will cover that route. The travel manager performs this task. He needs to add the new bus and route ids along with the source and destination of the new route. He also needs to add the details related to at least one bus covering that route. Later he can add as many buses on the same route as he wants.

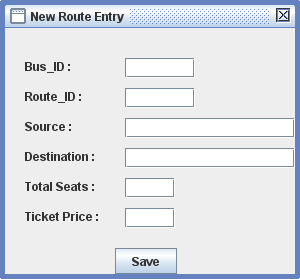


Figure: New Route Entry Form

## Code Snippets:

Below is the code behind the ‘Save’ Button. When ‘Save’ button is pressed the following code is triggered to show the availability information.

First we make a connection with the database using the following code:

Class.forName ("oracle.jdbc.driver.OracleDriver");

Now we write the connection string that actually prepares and maintains the connection between the java application and the Oracle Listener Service through port 1521 at the localhost using Oracle thin client JDBC driver, where SCT is the database name and username and password are the respective name and password of the database user.

con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:SCT","username","password");

Next we create a statement object as follows:

stmt = con.createStatement();

Then we execute the queries required to perform the task. First we make an entry into the route table followed by an entry into the bus table. The codes are shown below:

stmt.executeQuery ("insert into route values ('" +Route\_ID\_TextField.getText () +"','"+Source\_TextField.getText () +"','"+Destination\_TextField.getText () +"')");

stmt.executeQuery("insert into bus values('" + Bus\_ID\_TextField.getText()+"','"+Route\_ID\_TextField.getText()+"',"+Ticket\_price\_TextField.getText()+","+Total\_seats\_TextField.getText()+")");

The above statements take the required details from the corresponding text fields and complete the statements. The statements are then executed as a query which makes an entry into the route table and the bus table respectively.

## Customer Enquiry Form

This form is a java file that receives the Date, Source, Destination from the customer and finds the Availability Information, i.e. it shows the buses that are available on that route along with its total seats and the number of already seats booked in that bus, so that the Travel manager can see the Availability Information and convey to the customer which buses are available according to the customers’ needs. When a decision has been taken the travel manager then enters the customer information below along with the bus the customer has chosen to book ticket and the respective seat is booked under the name of that customer.

Below is a snapshot of the customer enquiry form:

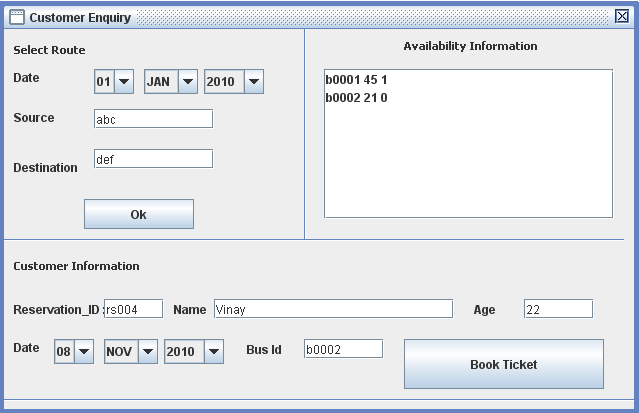


Figure: Customer Enquiry Form

## Code Snippets:

Below is the code behind the Select Route Button. When ‘Select Route’ button is pressed the following code is triggered to show the availability information.

First we establish the connection with the Oracle database, for that we need to include the JDBC driver for Oracle:

Class.forName("oracle.jdbc.driver.OracleDriver");

Now we write the connection string that actually prepares and maintains the connection between the java application and the Oracle Listener Service through port 1521 at the localhost using Oracle thin client JDBC driver, where SCT is the database name and username and password are the respective name and password of the database user.

con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:SCT", "username", "password");

Now we prepare the Query to be run for the intended purpose:

s1 = con.createStatement();

s="select X.bus\_id,total\_seats,Maximum from (select bus\_id,total\_seats from bus,route where bus.route=route.route\_id and source='" + sourceTxt.getText() + "' and destination='" + destnTxt.getText() + "') X left outer join (select bus,max(seat\_no)as Maximum from reservation where booking\_dt='" + DayComboBox.getSelectedItem()+ "-"+ MonthComboBox.getSelectedItem()+"-"+YearComboBox.getSelectedItem() + "'group by bus)y on X.bus\_id = y.bus";

Now we run the query and obtain the result in a record set called ‘records’:

records=s1.executeQuery(s);

Now we loop through the record set and display the results in suitable format:

while (records.next())

{

BusId=records.getString(1);

TotalSeats=records.getInt(2);

MaxSeatNo=records.getInt(3);

s="";

s= BusId + " " + Integer.toString(TotalSeats) + " " + Integer.toString(MaxSeatNo);

AvailabilityListBox.setModel(LM);

LM.add(i, s);

i++;

}

## View Bus and Route Information Form:

In this form the Travel manager can view every details about the various routes on which the travel company operates along with the details of the buses that are running on each of those routes. The main objective of this form is to allow the travel manager to get a full view of the entire system of the company with respect to the services that the company offers.

Below is the Snapshot of the View Bus and Route Form:

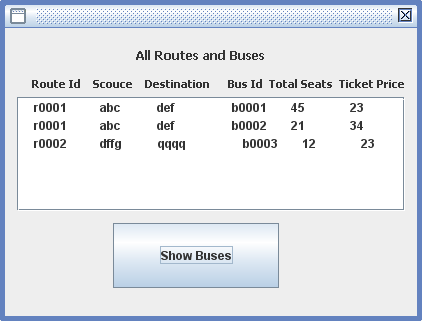


Figure: View All Routes And Bus Form

## Code Snippets:

Below is the code behind the ‘Show Buses’ Button. When ‘Show Buses’ button is pressed the following code is triggered to show the availability information.

First we establish the connection with the Oracle database, for that we need to include the JDBC driver for Oracle

Class.forName("oracle.jdbc.driver.OracleDriver");

Now we write the connection string that actually prepares and maintains the connection between the java application and the Oracle Listener Service through port 1521 at the localhost using Oracle thin client JDBC driver, where SCT is the database name and username and password are the respective name and password of the database user.

con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:SCT","username", "password");

Now we prepare the Query to be run for the intended purpose :

s1 = con.createStatement();

s="select route.route\_id,source,destination,bus\_id,total\_seats,bus.ticket\_price from ROUTE,BUS where ROUTE.ROUTE\_ID=BUS.ROUTE order by ROUTE.ROUTE\_ID";

Now we run the query and obtain the result in a record set called ‘records’:

records=s1.executeQuery(s);

Now we loop through the record set and display the results in suitable format:

while (records.next())

{

route\_id=records.getString("route\_id");

src=records.getString("source");

destn=records.getString("destination");

BusId=records.getString("bus\_id");

TotalSeats=records.getInt("total\_seats");

ticketprice=records.getInt("ticket\_price");

s="";

s= " " + route\_id + " " + src + " " + destn + " " + BusId + " " + Integer.toString(TotalSeats) + " " + Integer.toString(ticketprice);

ShowBusList.setModel(LM2);

LM2.add(i, s);

i++;

}

}