UCS310: Database Management System
PL/SQL Project

Airline Management System

SUBMITTED BY

Pranav Pratap Singh: 101610068

Rahul Kaushik: 101603262

COE 18

2017-18

SUBMITTED TO

Mr Santosh Singh Rathore

Lecturer, CSED

Thapar University, Patiala

SUBMITTED ON

30th October 2017

Thursday

DEVELOPED IN

*Oracle iSQL*Plus Client for SQL and PL/SQL*

Requirement Analysis

<u>Problem Statement</u>: Develop a general purpose Airline Management System to be used by a parent company managing different small airline companies.

<u>Problem Details</u>: A parent company 'A' owns different sub-companies operating in the airline industry. Requirement is a tool to aid the management process for the employees of the company 'A'. Requirement may be extended further to support employees of the sub-companies also. Since the tool has to be hosted on a remote server, storage space is constrained. Hence, the tool must implement in minimum secondary storage per record.

<u>Problem Conclusions</u>:

- Normalization of table structure removes redundant records, hence is a must.
- Records for passengers must be saved as a separate entity from the bookings. Means all passengers will have a log-in access.
- Payments must be mapped with all tickets, so that the statistics for a specific company can be calculated.
- Some tables will be needed to keep input-values in check, this will be an additional overhead for the database storage, but is necessary as access is being provided to multiple users.

<u>Fields to be Recorded</u>:

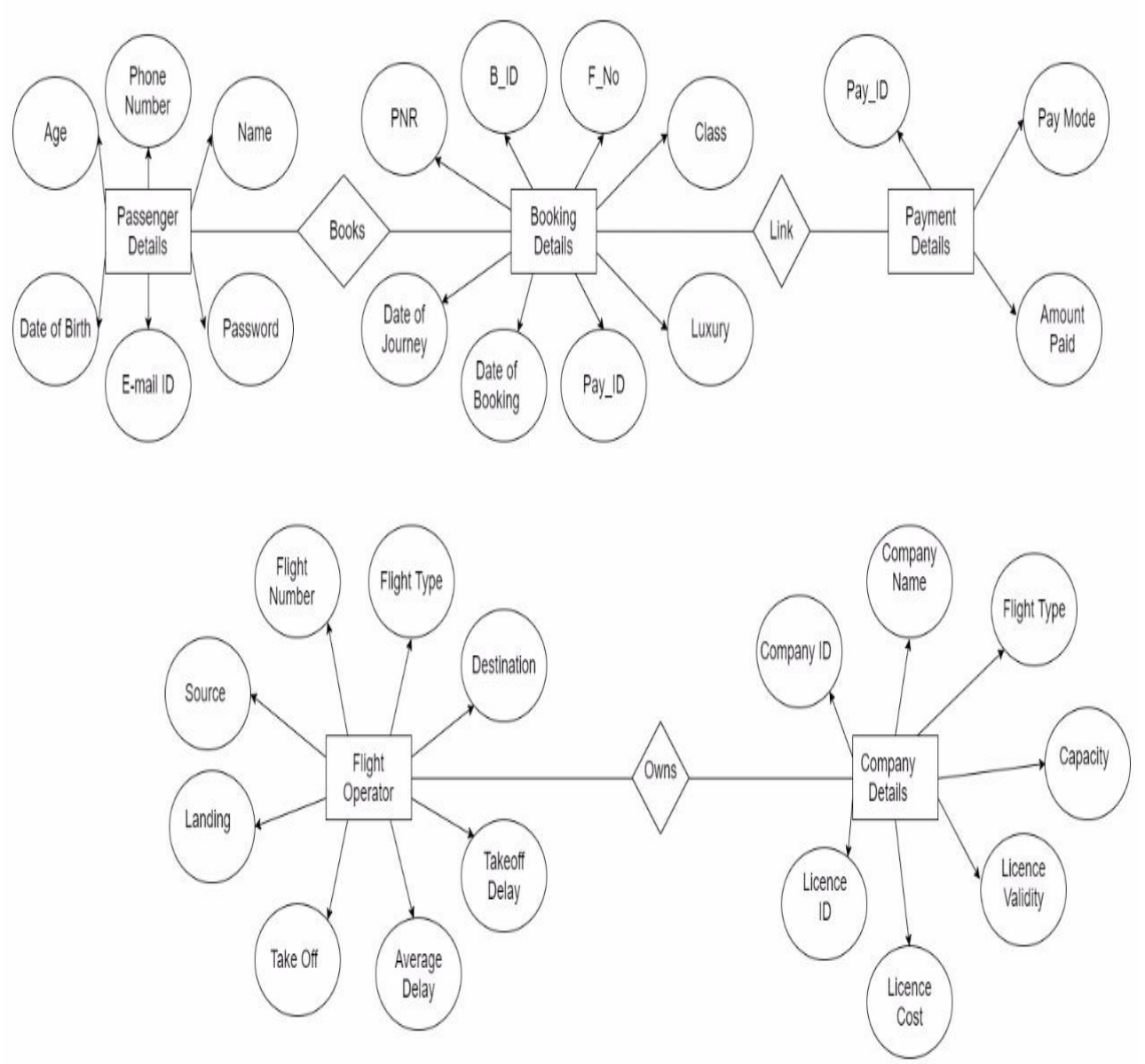
- Passenger Details
    - Name
    - Phone Number
    - Date of Birth
    - Mail ID
    - Password (for Login)
- Ticket Details
    - PNR Number (Unique Identifier)
    - Passenger Mapping
    - Class of Travel
    - Date of Journey
    - Payment Details
    - Flight Details
    - Date of Booking

        - Catering Information
- Flights Operated
    - Company
    - Flight Number
    - From
    - To
    - Timings
    - License Details

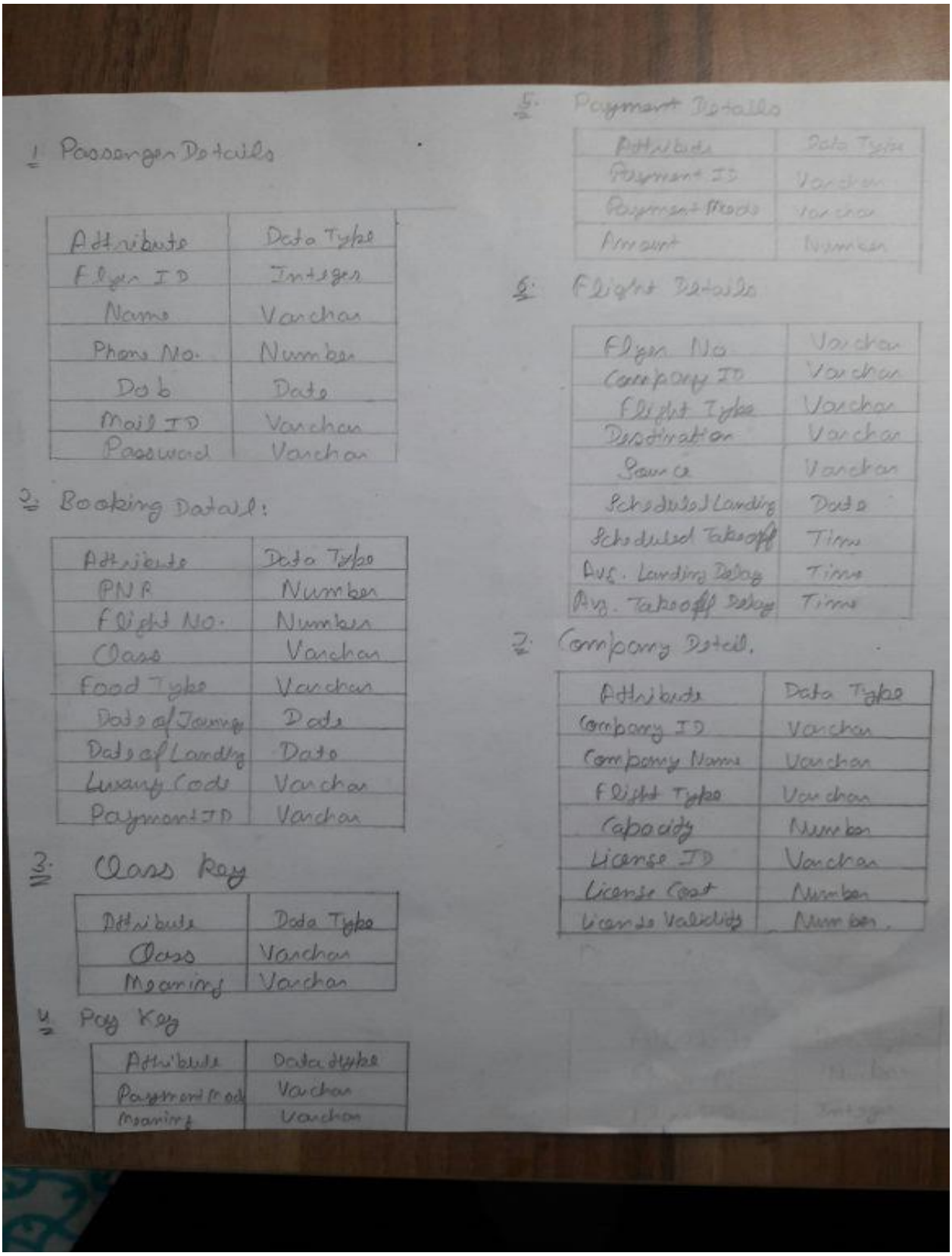
Conclusions on Basis of Fields:

- Separate tables will be made for Tickets, Passengers and Flights.
- Booking of a ticket can be permitted only when certain criteria are met
    - Passenger should be registered
    - Flight should exist
    - Capacity of aircrafts can also be a limiting factor to make ticket bookings.
- All these tables will need sequence based identifiers for Primary Key, so that indexed access is possible. It will also make linking easier and less expensive on the secondary storage perspective.
- Password should be sent in encrypted format from client to server, so that security infringement of password doesn't lead to unexpected transactions against the user's account.

E-R Diagram:


Age
Phone Number
Name
Passenger Details
Date of Birth
E-mail ID
Password
Books
PNR
B_ID
F_No
Class
Booking Details
Date of Journey
Date of Booking
Pay_ID
Luxury
Link
Pay_ID
Pay Mode
Payment Details
Amount Paid
Flight Number
Flight Type
Destination
Source
Flight Operator
Landing
Takeoff Delay
Take Off
Average Delay
Owns
Company ID
Company Name
Flight Type
Company Details
Capacity
Licence ID
Licence Validity
Licence Cost

Developed Tables:



1. Passenger Details

Attribute	Data Type
Flyer ID	Integer
Name	Varchar
Phone No.	Number
Dob	Date
Mail ID	Varchar
Password	Varchar

2. Booking Datail:

Attribute	Data Type
PNR	Number
Flight No.	Number
Class	Varchar
Food Type	Varchar
Date of Journey	Date
Date of Landing	Date
Luxury Code	Varchar
Payment ID	Varchar

3. Class Key

Attribute	Data Type
Class	Varchar
Meaning	Varchar

4. Pay Key

Attribute	Data dtype
Payment Mode	Varchar
Meaning	Varchar

5. Payment Details

Attribute	Data Type
Payment ID	Varchar
Payment Mode	Varchar
Amount	Number

6. Flight Details

Flyer No	Varchar
Company ID	Varchar
Flight Type	Varchar
Destination	Varchar
Source	Varchar
Scheduled Landing	Date
Scheduled Takeoff	Time
Avg. Landing Delay	Time
Avg. Takeoff Delay	Time

7. Company Detail.

Attribute	Data Type
Company ID	Varchar
Company Name	Varchar
Flight Type	Varchar
Capacity	Number
License ID	Varchar
License Cost	Number
License Validity	Number

Table Functional Dependencies

Identified Functional Dependencies

TABLE: PASSENGER_DETAILS

Flyer ID ---> Name
Flyer ID ---> Phone No
Flyer ID ---> DOB
Flyer ID ---> Mail ID
Flyer ID ---> Password
Mail ID ---> Flyer ID
Mail ID ---> Name
Mail ID ---> Phone No
Mail ID ---> DOB
Mail ID ---> Password

TABLE: BOOKING_DETAILS

PNR ---> Flyer ID
PNR ---> Flight No
PNR ---> Class
PNR ---> Food Type
PNR ---> Date of Journey
PNR ---> Date of Booking
PNR ---> Luxury Code
PNR ---> Insurance
PNR ---> Payment ID
Payment ID ---> Flyer ID
Payment ID ---> Flight No
Payment ID ---> Class
Payment ID ---> Food Type
Payment ID ---> Date of Journey
Payment ID ---> Date of Booking
Payment ID ---> Luxury Code
Payment ID ---> Insurance
Payment ID ---> Payment ID

TABLE: CLASSKEY

Class ---> Meaning

Meaning ---> Class

TABLE: PAYMENT_DETAILS

Payment ID ---> Payment Mode

Payment ID ---> Amount

TABLE: PAYKEY

Payment Mode ---> Meaning

Meaning ---> Payment Mode

TABLE: COMPANY_DETAILS

Company ID ---> Company Name

Company ID ---> Flight Type

Company ID ---> Capacity

Company ID ---> License ID

Company ID ---> License Cost

Company ID ---> License Validity

License ID ---> Company ID

License ID ---> Company Name

License ID ---> Flight Type

License ID ---> Capacity

License ID ---> License Cost

License ID ---> License Validity

TABLE: FLIGHT_OPERATORS

Flight No ---> Company ID

Flight No ---> Source

Flight No ---> Destination

Flight No ---> Scheduled Takeoff

Flight No ---> Scheduled Landing

Flight No ---> Average Takeoff Delay

Flight No ---> Average Landing Delay

Functional Dependecies to Ignore

- In table COMPANY_DETAILS, License ID is also capable of acting as a primary key. This means that one License ID may be associated with only one Company ID. Hence, making a separate license table is an unnecessary management overhead for the database.
- In table BOOKING_DETAILS, Payment ID can be used to uniquely map to any PNR. But Payment ID may be recycled in the future, after the permissable time of legal boundaries. Hence, removing any of the keys is not possible.
- Unique Flyer ID is used in the PASSENGER_DETAILS table instead of Mail ID because passengers need to be referred in other tables also, and this will help preserve secondary storage space.

Normalized form of Tables

Keeping in mind the points discussed above, certain normalizations can be made in the given tables. Following changes have been made after normalization of the tables, hence getting the final table structure of the Database.

- A table AIRPORTS introduced for integrity constraint. This will help remove Airport names from the FLIGHT_OPERATORS table fields SOURCE and DESTINATION.
- PASSEGNER_DETAILS table is already normalized by nature
- BOOKING_DETAILS table is already normalized by nature
- CLASSKEY table is a normalization table
- PAYKEY table is a normalization table
- PAYMENT_DETAILS table is already normalized by PAYKEY
- COMPANY_DETAILS table is already normalized by nature
- FLIGHT_OPERATORS table is now normalized by AIRPORTS table

Implementation of Project

With the processing as described above, it was possible to start coding for the project.

This tool uses PLSQL procedures to insert values into some tables, the ones with Auto-Incrementing Primary keys. This is so that the programmer doesn't need to insert the values of those fields explicitly. These values are inbuilt in the code of the procedures.

Another quite possible challenge is iterative searches in the densely populated tables. To solve these issues, the tool is equipped with PLSQL functions that search as per the need and return a VARCHAR2 literal with the result. This result can then be displayed on the screen, or used for further processing at the back-end, as per the requirement.

On successful insertions of records in any table, Triggers are called which inform the insertion status to the user on the DBMS console. This output can certainly be used to process and check status according to requirement of future scaling of the project.

All the fields capable of Auto-Incrementing values are supported by sequences which can be used to auto-generate the next value of the tuple.

Real implementation of the project tells us that there are a total of

- 2 Functions
  - GET_BOOKING_DETAILS (NAME_READ IN VARCHAR2)
  - GET_ALL_BOOKING_DETAILS(PNR_READ IN NUMBER)
- 6 Procedures
  - MAKE_CLASS (CNAME IN VARCHAR2)
  - MAKE_PAYKEY (PNAME IN VARCHAR2)
  - MAKE_PAYMENT (PMODE IN NUMBER, AMT IN NUMBER, CODE OUT NUMBER)
  - ADD_USER (NAME IN VARCHAR2, PHNO IN NUMBER, DOB IN DATE, MAIL IN VARCHAR2, PASSWD IN VARCHAR2)
  - ADD_TICKET (FLID IN NUMBER, FLNO IN VARCHAR2, CLASS IN NUMBER, FTYPE IN VARCHAR2, DOJ IN DATE, LCODE IN VARCHAR, PMODE IN NUMBER, PAMT IN NUMBER)
  - MAKE_TICKET (FLID IN NUMBER, FLNO IN VARCHAR2, CLASS IN NUMBER, FTYPE IN VARCHAR2, DOJ IN DATE, LCODE IN VARCHAR, PMODE IN NUMBER, PAMT IN NUMBER)
- 6 Sequences
  - FLYERID
  - INSKEY
  - PKEY
  - CKEY
  - PDET
  - PNR

in addition to the tables in the PL/SQL code. The complete system works on a time complexity varying from O(1) to O(n) with a space complexity varying from O(1) to O(n*log n).

Project incorporates cursors in function GET_BOOKING_DETAILS to traverse through all records and hence get the PNR of all bookings. A need

for cursors was not visible in any other function/procedure because all others fetch only ONE record.

The tool uses Error Reporting flags through exception handling mechanism of PL/SQL to identify the speculated errors and suggest possible solutions. Also, all the unspeculated exceptions are caught and mapped to an error code. This error code can be used to develop a stack trace of the exception and hence identify the bug. Suggesting possible makeshift solutions is made easy by such stack traces.

Since PL/SQL is not capable of returning the stack trace of an error for abstraction purposes, it is necessary for us to be able to develop stack traces for a given situation, because debugging a code of large size dealing with heavy tables in minimum time is possible only with a detailed stack trace. Hence, all error codes will be mapped to the function flagging the error, and the handled exceptions. Call of this function can be read from the code, and a stack trace can be developed. This will make the debugging process easier.

Most of the queries in a normalized Database structure involve joins, hence are complex in nature. Complexity of such queries may even go upto $O(n^m)$ where

- n = number of maximum records in any table
- m = number of tables joined to get desired result

The maximum number of tables used in the process is **5**. To restrict the complexity from going above O(n), RAM storage is used upto O(n*log n) and indexed columns are used to get faster searching and traversal.

- WHERE operations of PRIMARY KEY and INDEXED columns is of the complexity O(log n)
- All UNIQUE + NOT NULL columns can be indexed for increased traversal speeds.
- Cursor traversal is always linear, hence complexity achieved is O(n)

Keeping this in mind, an efficient use of the SQL selection with manual traversal helps us achieve the optimal time and space complexity to get the records displayed in minimum time possible, without making it an overhead for server RAM for the parent company.

User Manual for Airline Management Tool

Setting Up the Project on your iSQL*Plus

To set up project on your iSQL*Plus client means to load all the tables and ORACLE PL/SQL objects into your user so that all dependencies are ready and you are able to use the tool smoothly. In order to do this, follow these steps.

1. Log in to your Oracle iSQL*Plus account
2. Click on the Browse button
3. Locate the CREATE_TABLES.sql file in the dependencies
4. Click on UPLOAD
5. You will return to your client now.
6. Click Load Script.
7. At this stage, the Textarea will contain 62 lines of SQL code.
8. Click on Execute, and check the output.
9. The output should be something like what is shown.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

If the output is different from what is shown, follow the same steps but use CLEAR_DBSTR.sql file instead of CREATE_TABLES.sql file in STEP 03. Then, follow this exact same sequence of steps, this will re-initialize the required tables in the database. If the output still differs, please contact your vendor for debugging.

Once the desired output is achieved, all the tables have been created. Now, it is time to create the functions required for smooth and easy functioning. For this,

1. Follow the same steps as above for the file CREATE_SEQUENCES.sql
2. Log in to your Oracle iSQL*Plus account
3. In your file explorer, navigate to the Dependencies directory
4. Open the file CREATE_PROCEDURES.sql in any text editor, like Notepad for Windows or Gedit for Linux
5. Copy the code from beginning till the first blank line
6. Paste it in the Enter Statements textarea of the Oracle iSQL*Plus UI screen.
7. Click on Execute
8. Continue till all the procedures are created in your Oracle iSQL*Plus account.
9. Perform the same steps for the CREATE_FUNCTIONS.sql file and CREATE_TRIGGERS.sql file.

At this stage, your Oracle iSQL*Plus account is ready to accept the data, which means the initialization process is over now.

Please refer to the article on Inserting data in the PL/SQL structure on iSQL*Plus in this manual so that you don’t encounter any unexpected errors.

User Manual for Airline Management Tool

Understanding the Product

The product: Airline Management Tool is a very powerful utility designed to run on the Oracle iSQL*Plus panel. But in order to use this tool, it is necessary to understand some aspects of the product, and the consequences any action on the iSQL*Plus panel can entail.

First of all, it must be noted that all changes, unless stated otherwise, are auto-committing in nature. Which means, there is no going back. If you change/delete a particular record using your iSQL*Plus panel, the record and all it's traces are removed from the database. After that, there is no reliable method of getting back the old information. The tool DOES NOT use a recycle bin or record logs as an attempt to save the Secondary Storage space provided.

The product use PL/SQL codes at different places, and hence it is necessary that we enable server outputs for the iSQL*Plus panel so that the output of the processing can be seen, and any error messages can be understood. In order to do this, the user must execute the following command every time a new session connects to the iSQL*Plus panel.

```
SET SERVEROUTPUT ON;
SET VERIFY OFF;
```

Once these commands are executed, it is safe to start using the iSQL*Plus panel for the particular session.

There is heavy dependency among the tables in the tool. Any insertion operation must be performed after checking all dependencies and all tuples of the dependent tables. If this is not done, the insertion operation will fail with an error message similar to

```
INTEGRITY CONSTRAINT (USERNAME.SYS_ERRORCODE) VIOLATED - PARENT KEY NOT FOUND
```

This message means that one or more dependencies of the table don’t have the required entries. For example: an attempt to register a flight from or to an airport not registered in the AIRPORTS table.

User Manual for Airline Management Tool

Calling Queries

The product comes with two folders in addition to the Oracle iSQL*Plus package. One of them is the Dependencies folder which we have already used to set up the product, the other one is the Queries folder which hasn't be touched till now. This folder contains numerous scripts to make handling the Airline Management tool easier.

To execute any pre-defined script, follow these steps.

1. Log in to your Oracle iSQL*Plus account
2. Click on Browse
3. Navigate and upload the script
4. Click on Load Script
5. Click on Execute

All the scripts in the Queries folder are designed to ask the values at run-time from the user. There is no need to make any changes whatsoever in these scripts.