Lesson: 1.1

JDBC Basics

Objectives:

In this lesson, you will exercise to:

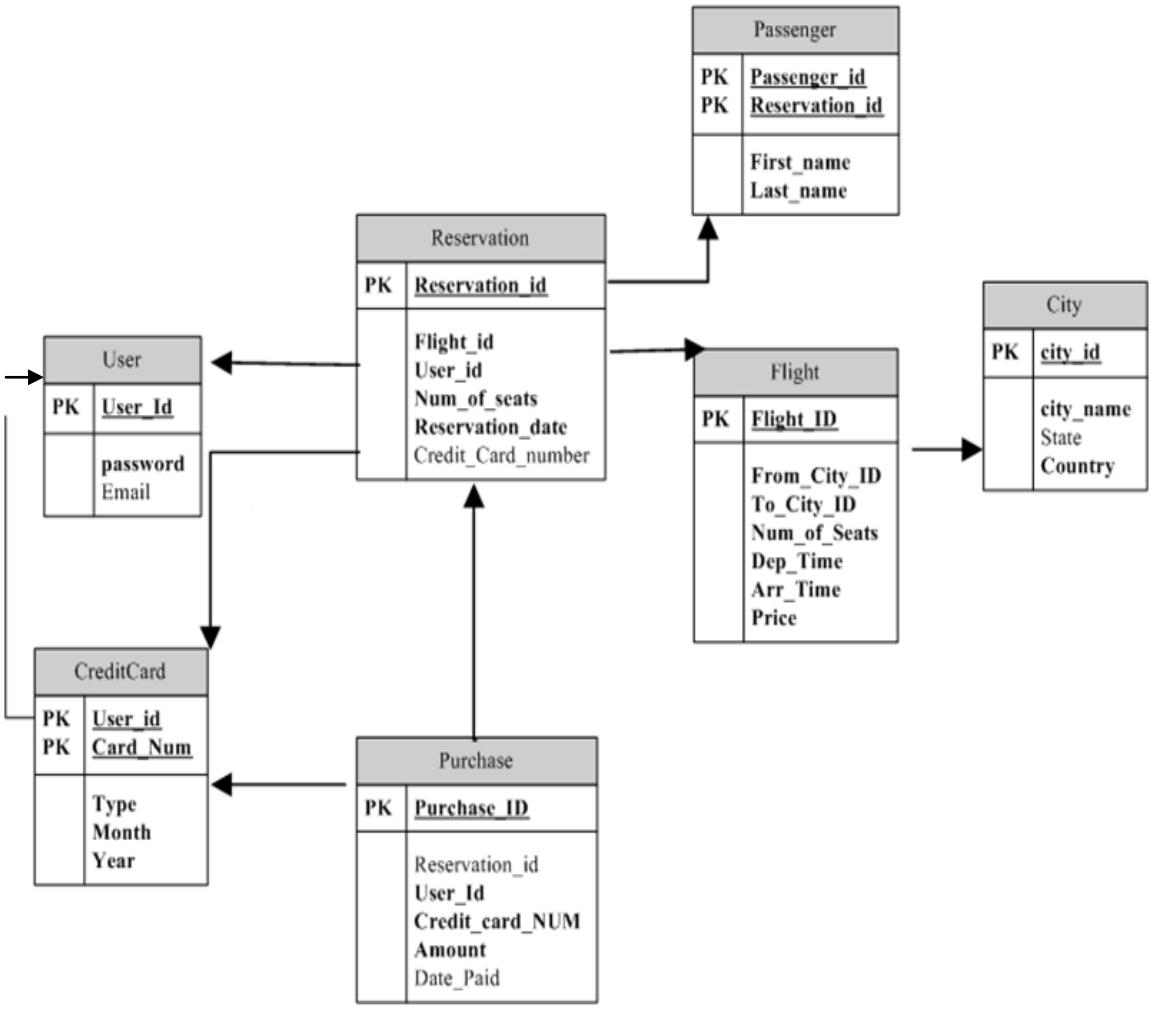
- ✓ Understand about JDBC drivers.
- ✓ Know about JDBC API.
- ✓ Administrating ODBC Data source.
- ✓ Java Database Programming steps.
- ✓ Connect to database using Type 1 and Type 4 driver.
- ✓ Database operations.

Scenario

BIG BIRD Airlines is a leading travel company offering leisure and business travelers the widest selection of low fares as well as deals on lodging and vacation packages. The site was created to address the need for an unbiased, comprehensive display of fares and rates in a single location for consumers. *BIG BIRD* search results are presented in an easy-to-use Matrix that displays a vast array of travel options for travelers. BIG BIRD Airlines is the first and only travel site with a *BIG BIRD Customer Care* Team that monitors nationwide travel conditions for our travelers around the clock, everyday.

BIG BIRD mission is to offer flexible leisure travelers a quick and easy way to get better deals on airline tickets. All purchases made on BIG BIRD Airlines are not final and can be cancelled, refunded or changed. If you want, you can also receive your updates by calling toll-free number and using your personalized PIN. When customers prefer a specific travel itinerary, they offer the widest range of flight options and fares.

For the given system all the data is maintained through the following table's hierarchy and structure.



Note: Use the above tables to solve the various coding related tasks throughout the complete workbook.

JDBC Basics

- JDBC is an API that provides a mid-level access to DBMS from Java applications.
- Intended to be an open cross-platform standard for database access in Java.
- The goal of JDBC is to be a generic SQL database access framework that works for any database system with no changes to the interface code.
- Provides a standard set of interfaces for any DBMS with a JDBC driver – using SQL to specify the databases operations.
- JDBC provides a standard library for accessing relational databases.
- By using the JDBC API, you can access a wide variety of SQL databases with exactly the same Java syntax.
- Officially, JDBC is not an acronym and thus does not stand for anything.
- Unofficially, "**Java Database Connectivity**" is commonly used as the long form of the name.
- The Java Database Connectivity (JDBC) abstracts Database Connectivity from the Java Application.
- It allows Java programmers to write a database application that can run on different platforms, i.e. database can be transferred from one vendor to another and the same java program can be used without alterations.

✍ Task 1 : Explore the API

1.1. Identify the steps for writing a java program using JDBC.
1.2. Identify the main components of JDBC.
1.3. Identify the two logical layers into which a JDBC application can be divided.
1.4. Identify the new features added to JDBC 4.0.
1.5. Describe the JDBC architecture.

Understand about JDBC Drivers

- A **JDBC driver** translates standard JDBC calls into a network or database protocol or into a database library API call that facilitates communication with the database.
- A **JDBC driver** is a software component enabling a Java application to interact with a database and is analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers.
- If the back-end database changes, only the JDBC driver need be replaced with few code modifications.
- The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

- In simplest terms, a JDBC technology-based driver ("JDBC driver") makes it possible to do three things:
  - ✓ Establish a connection with a data source.
  - ✓ Send queries and update statements to the data source.
  - ✓ Process the results.
- JDBC technology-based drivers generally fit into one of four categories:
  - ✓ Type 1: JDBC-ODBC Bridge driver.
  - ✓ Type 2: Native-API/partly Java driver.
  - ✓ Type 3: JDBC Net-protocol/all-Java driver.
  - ✓ Type 4: Native-protocol/all-Java driver.
- The **JDBC type1 driver**, also known as the JDBC-ODBC Bridge, is a database driver implementation that employs the ODBC driver to connect to the database.
- The **JDBC type 2 driver**, also known as the Native-API driver, is a database driver implementation that uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.
- The **JDBC type 3 driver**, also known as the Pure Java Driver for Database Middleware, is a database driver implementation which makes use of a middle tier between the calling program and the database.
- The **JDBC type 4 driver**, also known as the Direct to Database Pure Java Driver, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol. Therefore it is called a **THIN** driver.
- Driver categories 3 and 4 are the preferred way to access databases using the JDBC API. Driver categories 1 and 2 are interim solutions where direct pure Java drivers are not yet available.

✍ Task 2 : Explore the API

2.1. Identify the fastest JDBC driver.
2.2. How do we load a database driver with JDBC 4.0 / Java 6?
2.3. What is JDBC Driver interface?
2.4. Identify the protocol used in type 4 driver.
2.5. What are thin driver and thick driver? Why it is called so?

✍ Fill in Queries:

1. JDBC-ODBC Bridge does not work with Microsoft J++, because it does not support

2. List out the 4 different types of JDBC drivers, for use in different deployment scenarios provided by Oracle

3. Which Driver is preferable for using JDBC API in Applets?

4. What is the statement we need to use particularly when using type2 and type4 drivers?

5. The Oracle connection URL for the thin client-side driver ojdbc14.jar has the following format.

Know about JDBC API

- The JDBC API provides universal data access from the Java programming language.
- Using the JDBC API, one can access virtually any data source, from relational databases to spreadsheets and flat files.
- JDBC technology also provides a common base on which tools and alternate interfaces can be built.
- The JDBC API consists of a set of classes and interfaces written in the Java programming language that provide a standard API for tool/database developers and makes it possible to write industrial strength database applications using an all-Java API.
- An JDBC API makes it possible to do three things:
  - ✓ Establish a connection with a data source.
  - ✓ Send queries and update statements to the data source.
  - ✓ Process the results.
- The JDBC API is used to invoke (or "call") SQL commands directly.
- The JDBC API is comprised of two packages:
  - ✓ **java.sql package**.
  - ✓ **javax.sql package**, which adds server-side capabilities.
- **java.sql** package provides the API for accessing and processing data in a data source using the Java programming language.
- This API includes a framework whereby different drivers can be installed dynamically to access different data sources.
- The **java.sql** package contains API for the following:
  - ✓ Making a connection with a data source.
  - ✓ Sending SQL statements to a database.
  - ✓ Retrieving and updating the results of a query.

- ✓ Mapping an SQL value to the standard mapping in the Java programming language.
- ✓ Throwing exceptions.
- ✓ Providing security.

- **javax.sql** package is the JDBC Optional Package API.
- This package extends the functionality of the JDBC API from a client-side API to a server-side API, and it is an essential part of Java 2 SDK, Enterprise Edition technology.

✍ Task 3 : Explore the API

3.1. Identify the different classes and interfaces as part of the JDBC API for making a connection with a data source.
3.2. Identify how the JDBC API can be used to access a desktop database like Microsoft Access over the network?
3.3. What causes the "No suitable driver" error?
3.4. Identify the interfaces used for custom mapping of an SQL user-defined type to a class in the Java programming language.
3.5. Identify the interface and the related methods which each driver class must implement.

✍ Fill in Queries:

1. Complete the method parameter list: **public <*modifier*> Connection connect (*Parameter1*, *Parameter2*) throws <*Exception*>**

2. Method explicitly called before executing a stored procedure to register the *java.sql.Type* of each "**out**" parameter.

3. Method which returns a Java object whose type corresponds to the SQL type that was registered for this parameter using *registerOutParameter*.

4. Method which returns true if the driver thinks that it can open a connection to the given URL.

5. Complete the method declaration: **static public <*modifier*> Connection getConnection (*Parameter1*) <*Exception*>**

Task 1 Workspace

Task 2 Workspace

Task 3 Workspace

Administrating ODBC Data source

- The Open Database Connectivity (ODBC) standard is a common application programming interface for accessing data files. In other words, ODBC allows you to move data back and forth easily between ODBC compliant applications.
- A connection to a specific database is called a **Data Source Name** or **DSN.**
- ODBC can be used as a translation engine, facilitating the movement of data between different applications with dissimilar file structures.
- ODBC is composed of four main components: the **client application**, **ODBC driver manager**, **ODBC drivers**, and the **ODBC data source**.
- The **ODBC driver manager** is a program used to manage the links between the various applications and to configure data transfer settings.
- The **ODBC drivers** serve as the link between the client and ODBC data, and must be installed for each application.
- Finally, an **ODBC data source** is a user-generated configuration of a particular data file created using the ODBC driver manager for access by client applications.

Creating an ODBC Source with Access:

- Microsoft Access is a relational database program with each database file containing one or more *tables* of data.

- With ODBC, you can read Access database or query tables. As with Microsoft Excel, you are able to use either the generic ODBC data source, or you can create a file-specific data source to simplify the task of transferring data.
- To create a file-specific data source. You would have to do the following:
    1. Launch the ODBC Administrator program.
    2. When the ***Data Sources*** dialogue box appears, click the **Add...** button.
    3. From the list of available ODBC drivers that then appears, click once on **Microsoft Access Driver (*.mdb,*.accdb)** and then click **Finish**.
        - Now you must assign a data source name.
    4. Now you must specify the file that is going to be associated with this data source. Click the **Select...** button to assign a filename to this data source.
    5. When the ***Select Database*** dialogue box appears, just type in the filename in the text box in the upper left hand corner and click **OK**.
    6. Click **OK** to return to the ***Data Sources*** dialogue box. You will now see your new data source in the list of ***User Data Sources***.
    7. We are now finished configuring this data source, so click **OK** to exit the ODBC Administrator program.
    8. You are now ready to use this Access database with other ODBC applications.

✍ Fill in Queries:

1. What are the flow statements of JDBC?

2. Is the JDBC-ODBC Bridge multi-threaded?

3. Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?

4. Define Data Source.

5. What Is JDBC-ODBC Bridge?

Java Database Programming Steps

- The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.
- The JDBC API consists of classes and interfaces for establishing connections with databases, sending SQL statements to databases, processing the results of the SQL statements, and obtaining database metadata.
- The JDBC interfaces and classes can be classified into types of driver, connection, statement, result set, metadata, processing support, and exception.
- JDBC helps you to write Java applications that manage these three programming activities:
    1. Connect to a data source, like a database.
    2. Send queries and update statements to the database.
    3. Retrieve and process the results received from the database in answer to your query.
- A typical Java program takes the following steps to access the database:
    - Load drivers.
        - JDBC allows multiple drivers to be loaded in one program.
        - May use the static method *registerDriver* in the *DriverManager* class to explicitly register a driver with the driver manager.
    - Establish connections.
        - Code a call to the *DriverManager* object's *getConnection( )* method to establish a database connection.
    - Create statements.
        - You can create statements for executing SQL in three types: *Statement*, *PreparedStatement*, or *CallableStatement*.
        - The Statement class can be used to execute static SQL.
        - The *PreparedStatement* and *CallableStatement* classes are used to construct dynamic SQL.
    - Execute statements.
        - The statements can be executed using the methods *executeQuery*, *executeUpdate*, or *execute*.
        - The result of the query is returned in *ResultSet*.
    - Process ResultSet.
        - The methods such as *absolute, first, last, next,* and *previous* can be used to set the row position.
    - Close the connections.
        - Releases the *Connection* object's database and JDBC resources immediately instead of waiting for them to be automatically released.

✍ Task 4 : Explore the API

4.1 Identify the method which creates a *Statement* object that will generate ResultSet objects with the given type and concurrency.
4.2 Identify the method which returns a new *Statement* object that will generate ResultSet objects with the given type, concurrency, and holdability.
4.3 Identify the method which returns a *ResultSet* object that contains the data produced by the query; never null.
4.4 Identify the method which attempts to locate a driver that understands the given URL and returns a *Driver* object representing a driver that can connect to the given URL.
4.5 Identify the method which executes the given SQL statement, which may return multiple results, and signals the driver that any auto-generated keys should be made available for retrieval.

✍ Code 1 : Test the Coding Skills

Code the Program

```
//SimpleJdbcTest.java
package jdbcdemo;

/* A JDBC example that connects to the Microsoft Access Sample database, issues a
simple SQL query to the Passenger table, and prints the results. */

//import all the required classes and interfaces;

public class SimpleJdbcTest {
public static void main(String[] args) {
String driver = _____; //Type 1 driver
String url = "jdbc:odbc:_____";
String username = ""; // No username/password required
String password = ""; // for desktop access to MS Access.
showPassengerTable(_____,_____,_____,_____);
} //End of main()

/* Query the Passenger table and print the First and Last names. */
public static void showPassengerTable(String driver, String url, String username,
String password)
{
try {
// Load database driver if it's not already loaded.
_____;
// Establish network connection to database.
Connection connection = _____;
```

Code the Program…. continued

```
System.out.println ("Passenger\n" + "==========");
// Create a statement for executing queries.
Statement statement = connection._____();
String query = _____;
// Send query to database and store results.
ResultSet resultSet = _____;
// Print results.
while(_____)
  {
        System.out.print (_____("Firstname") + " ");
        System.out.println (_____("Lastname"));
   }
_____;   //Close the connection object
} //End of try
catch (_____  cnfe) {
        System.err.println ("Error loading driver: " + cnfe);   }
catch (_____ sqle) {
        System.err.println ("Error with connection: " + sqle);    }
   }
} //End of class.
```

Using the Native-Protocol Driver (Type 4)

- A native-protocol driver translates JDBC calls into network protocol used by a DBMS and communicates with the DBMS directly.
- The driver is written in 100% pure Java to implement high-level network communication to the server.
- Oracle's JDBC thin driver is a native-protocol driver that provides an implementation of the Oracle high-level communications protocol known as *TNS* (*Transparent Network Substrate*).
- Have to add *Type IV driver jar* file to Library folder by right clicking on Library folder and select Add Jar/Folder option in the NetBeans IDE.
- This driver can access local or remote servers of Oracle 8i or higher from a client on any platform.

Core Approach

Use the column name instead of the column index when accessing data in a ResultSet.

✍ Code 2 : Test the Coding Skills

Code the Program

```
//TestThinApp.java
package jdbcdemo;

/* A JDBC example that connects to the Oracle database, issues a simple SQL query to
the City table, and prints the results. */

//import all the required classes and interfaces;



class TestThinApp {
public static void main (String args[]) throws _____
{
 Class.forName ("_____");
// or you can use:
// DriverManager._____ (_____());

Connection conn = DriverManager._____(
"_____","scott","tiger"); //Connection Object

Statement stmt = conn._____( );
ResultSet rset =
stmt._____("_____");

while(rset.next( )) {
System.out.println(_____);
System.out.println(_____); }

rset._____( );
stmt._____( );
_____.close( );
 }
} // End of class.
```

Core Approach

If you use **Type 4** driver we need to add *ojdbc14.jar* file to library folder of current project.
If you use **Type 1** driver, we need to create a DSN.

Database Operations

- In computer programming, **create, read, update and delete** (**CRUD**) are the four basic functions of persistent storage.
- All database operations can be classified into 4 major categories:
    1. Insert data (Create).
    2. Get existing data (Read).
    3. Modify existing data (Update).
    4. Delete data (Delete).
- These sets of tasks are called *CRUD*.

✍ Code 3 : Test the Coding Skills

Code the Program

```
//In this case you can run only one query at a time on the Purchase table. So the other
  two are commented.
public class CrudDemo {
        public static void main(String[] args) {
              try{
            //loading  the Type 1 driver
            Class.forName("_____");

            //Setting Connection String
            String dburl = "_____";

            //Getting connection Object
            Connection con = DriverManager.getConnection (_____);

            //Performing database Operation
            Statement st = _____;

             String sql="_____";
           //String sql="_____";
           //String sql="_____";

             int result = _____;
             System.out.println("No of rows affected=" + result);

           //Releasing Resources
             st._____(); con._____ ();
             System.out.println ("Connection closed");
                  }
                  catch (_____ ex) {ex.printStackTrace ();   } //End of main().
              } //End of class.
```

Task 4 Workspace / Unguided Practice

//Program to get the column names from the *Flight* table, using the Type 4 driver and displaying the column names from same table.