NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

DATABASE SYSTEM FOR A TOURISM COMPANY

GRADUATION PROJECT COM-400

Student: Faisal Iqbal Qureshi

Supervisor: Asst. Prof. Dr Firudin Muradov

Nicosia - 2004

Acknowledgements

Praise be to GOD Most Gracious most Merciful.

I would like to thanks my parents for there support in finishing this project.

My second thanks goes to my brothers for there valuable information and continuous support in writing this project.

Finally I would like to thank my supervisor ***Asst. Prof. Dr. Firudin Muradov*** *for his advice on the topics ,of whom without his support this project would never had come to end.*

Abstract

A database system is nothing more than a computerized record keeping system. The database can be regarded as a kind of electronic filling cabinet. The RDBMS such as Oracle is a complete reference as well as the most securest RDBMS that gives user a complete coverage as well as fulfills the needs of one who actually uses it.

Java is one of today's language which gives user a unique touch of object orientation. The user can connect to the database using the JDBC feature which allows Java to connect to the RDBMS.

The project fulfills the request of a database system for a Tourism company. In this system user can add, view, delete and change the database as of his/her requirements.

Table of Contents

List of Abbreviations

Abstract Window Toolkit (AWT)
A collection of graphical user interface (GUI) components that were implemented using native-platform versions of the components. These components provide that subset of functionality which is common to all native platforms. Largely supplanted by the Project Swing component set.

abstract
A Java(TM) programming language keyword used in a class definition to specify that a class is not to be instantiated, but rather inherited by other classes. An abstract class can have abstract methods that are not implemented in the abstract class, but in subclasses.

alpha value
A value that indicates the opacity of a pixel.

API
Application Programming Interface. The specification of how a programmer writing an application accesses the behavior and state of classes and objects.

applet
A component that typically executes in a Web browser, but can execute in a variety of other applications or devices that support the applet programming model.

argument
A data item specified in a method call. An argument can be a literal value, a variable, or an expression.

Bean
A reusable software component. Beans can be combined to create an application.

bit
The smallest unit of information in a computer, with a value of either 0 or 1.

bitwise operator
An operator that manipulates two values comparing each bit of one value to the corresponding bit of the other value.

block
In the Java(TM) programming language, any code between matching braces. Example: `{ x = 1; }`.

business logic
The code that implements the functionality of an application. In the Enterprise JavaBeans model, this logic is implemented by the methods of an enterprise bean.

byte
A sequence of eight bits. The Java(TM) programming language provides a corresponding `byte` type.

bytecode
Machine-independent code generated by the Java(TM) compiler and executed by the Java interpreter.

catch
A Java(TM) programming language keyword used to declare a block of statements to be executed in the event that a Java exception, or run time error, occurs in a preceding "try" block.

class
In the Java(TM) programming language, a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass will implicitly be `Object`.

class method
A method that is invoked without reference to a particular object. Class methods affect the class as a whole, not a particular instance of the class.

classpath
A classpath is an environmental variable which tells the Java(TM) virtual machine* and Java technology-based applications (for example, the tools located in the JDK(TM) 1.1.X\bin directory) where to find the class libraries, including user-defined class libraries.

class variable
A data item associated with a particular class as a whole--not with particular instances of the class. Class variables are defined in class definitions.

client
In the client/server model of communcations, the client is a process that remotely accesses resources of a compute server, such as compute power and large memory capacity.

codebase
Works together with the `code` attribute in the `<APPLET>` tag to give a complete specification of where to find the main applet class file: code specifies the name of the file, and codebase specifies the URL of the directory containing the file.

commit

The point in a transaction when all updates to any resources involved in the transaction are made permanent.

compilation unit
The smallest unit of source code that can be compiled. In the current implementation of the Java(TM) platform, the compilation unit is a file.

compiler
A program to translate source code into code to be executed by a computer. The Java(TM) compiler translates source code written in the Java programming language into bytecode for the Java virtual machine.

component
An application-level software unit supported by a container. Components are configurable at deployment time. The J2EE platform defines four types of components: enterprise beans, Web components, applets, and application clients.

constructor
A pseudo-method that creates an object. In the Java(TM) programming language, constructors are instance methods with the same name as their class. Constructors are invoked using the `new` keyword.

container
An entity that provides life cycle management, security, deployment, and runtime services to components. Each type of container (EJB, Web, JSP, servlet, applet, and application client) also provides component-specific services.

CORBA
Common Object Request Broker Architecture. A language independent, distributed object model specified by the Object Management Group (OMG).

declaration
A statement that establishes an identifier and associates attributes with it, without necessarily reserving its storage (for data) or providing the implementation

encapsulation
The localization of knowledge within a module. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but as long as the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten.

enterprise bean
A component that implements a business task or business entity; either an entity beans or a session bean.

Enterprise JavaBeans(TM) (EJB)

A component architecture for the development and deployment of object-oriented, distributed, enterprise-level applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user and secure.

exception
An event during program execution that prevents the program from continuing normally; generally, an error. The Java(TM) programming language supports exceptions with the try, catch, and throw keywords. See also exception handler.

exception handler
A block of code that reacts to a specific type of exception. If the exception is for an error that the program can recover from, the program can resume executing after the exception handler has executed.

executable content
An application that runs from within an HTML file.

extends
Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface extends another interface by adding methods. Class X is said to be a subclass of class Y.

garbage collection
The automatic detection and freeing of memory that is no longer in use. The Java(TM) runtime system performs garbage collection so that programmers never explicitly free objects.

GUI
Graphical User Interface. Refers to the techniques involved in using graphics, along with a keyboard and a mouse, to provide an easy-to-use interface to some program.

HTML
HyperText Markup Language. This is a file format, based on SGML, for hypertext documents on the Internet. It is very simple and allows for the embedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are embedded using URLs.

HTTP
HyperText Transfer Protocol. The Internet protocol, based on TCP/IP, used to fetch hypertext objects from remote hosts.

HTTPS
HTTP layered over the SSL protocol.

IDL
Interface Definition Language. APIs written in the Java(TM) programming language that provide standards-based interoperability and connectivity with CORBA (Common Object Request Broker Architecture).

IIOP
Internet Inter-ORB Protocol. A protocol used for communication between CORBA object request brokers.

implements
A Java(TM) programming language keyword optionally included in the class declaration to specify any interfaces that are implemented by the current class.

instance
An object of a particular class. In programs written in the Java(TM) programming language, an instance of a class is created using the `new` operator followed by the class name.

interpreter
A module that alternately decodes and executes every statement in some body of code. The Java(TM) interpreter decodes and executes bytecode for the Java virtual machine

JAR Files (.jar)
Java ARchive. A file format used for aggregating many files into one.

Java(TM)
is Sun's trademark for a set of technologies for creating and safely running software programs in both stand-alone and networked environments.

Java Application Environment (JAE)
The source code release of the Java Development Kit (JDK(TM)) software.

Java Development Kit (JDK(TM))
A software development environment for writing applets and applications in the Java programming language.

Java(TM) Platform
Consists of the Java language for writing programs; a set of APIs, class libraries, and other programs used in developing, compiling, and error-checking programs; and a Java virtual machine which loads and executes the class files.

JavaScript(TM)
A Web scripting language that is used in both browsers and Web servers. Like all scripting languages, it is used primarily to tie other components together or to accept user input.

JavaServer Pages(TM) (JSP)
An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client. Typically

the template data is HTML or XML elements, and in many cases the client is a Web browser.

Java(TM) virtual machine (JVM)
A software "execution engine" that safely and compatibly executes the byte codes in Java class files on a microprocessor (whether in a computer or in another electronic device).

Jini(TM) Technology
a set of Java APIs that may be incorporated an optional package for any Java 2 Platform Edition. The Jini APIs enable transparent networking of devices and services and eliminates the need for system or network administration intervention by a user.

The Jini technology is currently an optional package available on all Java platform editions.

JMAPI
Java(TM) Management API. A collection of Java programming language classes and interfaces that allow developers to build system, network, and service management applications.

JNDI
Java Naming and Directory Interface(TM). A set of APIs that assist with the interfacing to multiple naming and directory services.

JPEG
Joint Photographic Experts Group. An image file compression standard established by this group. It achieves tremendous compression at the cost of introducing distortions into the image which are almost always imperceptible.

JRE
Java(TM) runtime environment. A subset of the Java Developer Kit for end-users and developers who want to redistribute the runtime environment. The Java runtime environment consists of the Java virtual machine*, the Java core classes, and supporting files.

Just-in-time (JIT) Compiler
A compiler that converts all of the bytecode into native machine code just as a Java(TM) program is run. This results in run-time speed improvements over code that is interpreted by a Java virtual machine*.

JVM
Java(TM) Virtual Machine*. The part of the Java Runtime Environment responsible for interpreting bytecodes.

multithreaded
Describes a program that is designed to have parts of its code execute concurrently.

SAX

Simple API for XML. An event-driven, serial-access mechanism for accessing XML documents.

Secure Socket Layer (SSL)
A protocol that allows communication between a Web browser and a server to be encrypted for privacy.

servlet
A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a request-response paradigm.

SQL
Structured Query Language. The standardized relational database language for defining database objects and manipulating data.

TCP/IP
Transmission Control Protocol based on IP. This is an Internet protocol that provides for the reliable delivery of streams of data from one host to another.

thread
The basic unit of program execution. A process can have several threads running concurrently, each performing a different job, such as waiting for events or performing a time-consuming job that the program doesn't need to complete before going on. When a thread has finished its job, the thread is suspended or destroyed.

throw
A Java(TM) programming language keyword that allows the user to throw an exception or any class that implements the "throwable" interface.

throws
A Java(TM) programming language keyword used in method declarations that specify which exceptions are not handled within the method but rather passed to the next higher level of the program.

try
A Java(TM) programming language keyword that defines a block of statements that may throw a Java language exception. If an exception is thrown, an optional "catch" block can handle specific exceptions thrown within the "try" block. Also, an optional "finally" block will be executed regardless of whether an exception is thrown or not.

URL
Uniform Resource Locator. A standard for writing a text reference to an arbitrary piece of data in the WWW. A URL looks like "protocol://host/localinfo" where protocol specifies a protocol to use to fetch the object (like HTTP or FTP), host specifies the Internet name of the host on which to find it, and localinfo is a string (often a file name) passed to the protocol handler on the remote host.

Introduction

A Database management system is a collection of programs that enables users to create and maintain a database.

A RDBMS is a computerized record keeping system that stores maintains and provide access to the information. A Database system consists of four major components , that are Data , Hardware , Software and Users. DBMS are used by any reasonably self contained commercial , scientific ,technical or other organization for a single individual to a large company and a DBMS is used for a many reasons. The objective of this project was to design a software for a company which deals with Tourism Industry so fully qualified software was made for a Tourism Company.

This project consists of three chapters with appendix full of coding section.

Chapter One introduces some basics about Database and DBMS components that are used by any RDBMS software available now a days.

Chapter Two give a briefly detailed features of Oracle RDBMS and it's connectivity using Java as a modern language approach .

Chapter Three gives a view of a Tourism Company Database ,the ER-Diagram and Java Database connectivity.

Chapter 1

Introduction to Database Management Systems

1.1 Introduction

A Database system is essentially nothing more than a computerized record keeping system. The database itself can be regarded as a kind of electronic filling cabinet; or in other words it is repository for a collection of computerized data files. The information concerned can be any thing that is deemed to be significant to the individual or the organization , the system is intended to serve anything.

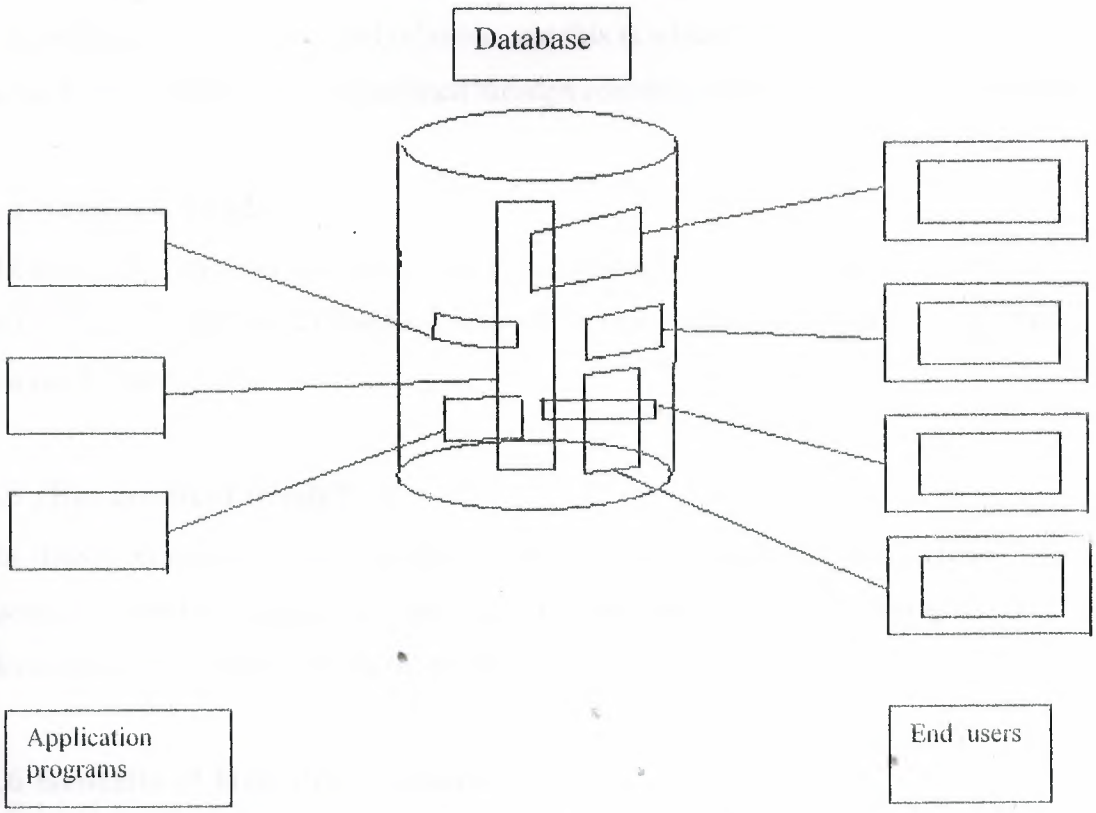

Figure 1.1 Simplified Picture of a Database Management System

1.2 Data Model

The model of data that they follow characterizes database management systems. A data model has tow components namely structure and operations. The structure refers to the way the system structures to the data or at least the way the users of the DBMS feel that

the data is structured. What is crucial is the way things feel to the user it does not matter how the designers of the dbms chose to implement these facilities behind the scenes.

There are three models for the vast majority of the DBMS's.

1) Relational Model
2) Network Model
3) Hierarchical Model

1.3 Relational Model

The user as being just a collection of tables perceives a relational model database. Formally these tables are called relations and this is where relational model gets it's name. Relation ships are implemented through common columns in two or more tables.

1.4 Network Model

The user as a collection of record types as relationships between these record types perceives a network model database such a structure is a network, and it is from there that model takes name.

1.5 Hierarchical Model

The data is represented to the user in the form of a set of tree structures and the operators provided for manipulating such structures include operators for traversing hierarchical paths up and down the trees .

1.6 Benefits of DBMS Approach

In this section I will only list the most benefits to be given to users via those who applied this approach are

1) Redundancy can be avoided
2) Inconsistency can be avoided
3) The Data can Be Shared
4) Standards can be Enforced
5) Security Restrictions can be Applied

6) Integrity can be maintained

7) Conflicting requirements can be balanced

1.7 Three Levels of Architecture

The three level architecture is an architecture for a DBMS to provide a framework for describing database concepts and structures. Not all DBMS fit neately into this architecture , but mostly do . This ANSI/SPARC architecture is divided into three levels, known as

1) **Internal level**

2) **Conceptual level**

3) **External level**

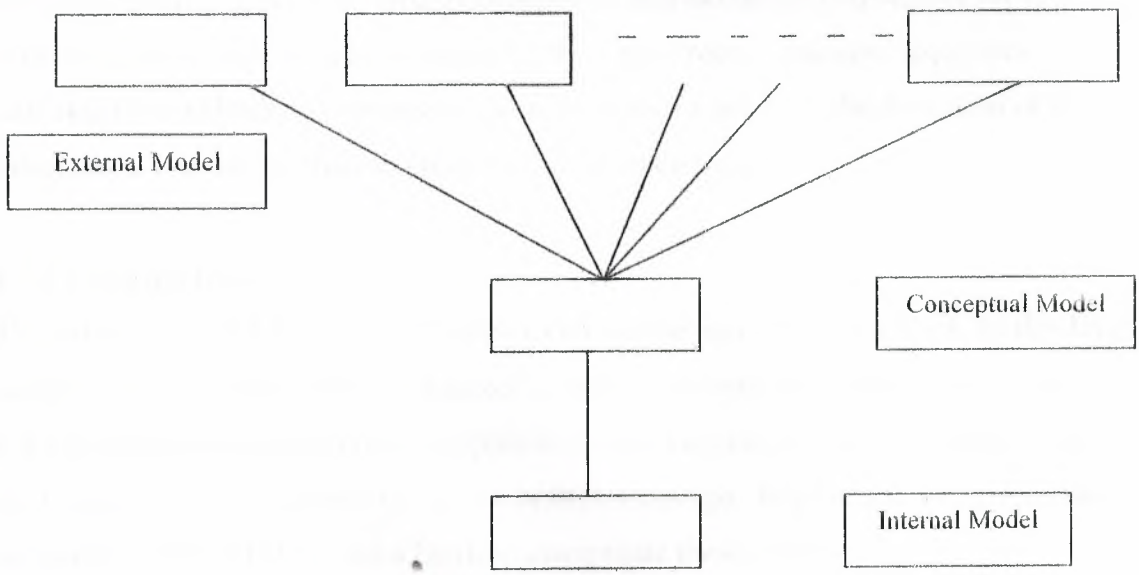

Figure 1.2 The Three Levels of the Architecture

1.7.1 Internal level

The internal level of the three level architecture is a low level representation of the entire database it consists of multiple types of internal records. It does not deal with block/pages or device dependent concepts like cylinders and tracks. The internal system defines types of stored records and index, how fields are represented, various storage

structure used, whether they chose pointer chains and hashing, what sequence they are in and so on. The internal schema is written yet by another data definition language , the internal DDL.

Programs accessing this level directly are dangerous since they have by passed the security and integrity checks which the DBMS program normally takes responsibility of.

1.7.2 Conceptual level

The conceptual level of the three level architecture is essentially a representation of the entire information content of the database in a form abstracted from physical storage . It may also be quite different or similar to external views held by a particular user. It is data as it really is . Rather than as users are forced to see it's multiple occurrences of multiple types of conceptual records.

The conceptual schema is defined by conceptual data definition language or DDL .There is no reference in the conceptual DDL to store record concepts sequences indexing hass addressing pointers etc the references are solely to the definition of the information content, in order to preserve data independence.

1.7.3 External level

The external level of the three level architecture is the individual user level. At this level each user has a language at their disposal of which they will use a data sub language. For the application programmer the application will be a conventional language . For the End user it is be a normal like query or SQL language . In principle any given data language consists of DDL and a DML to manipulate these objects.

An individual users view is an external view which is thus the content of the database as seen by that particular user. There will thus be multiple occurrences of multiple types of external records. The external view is defined by an external schema, which in turn is defined by the DDL part of the user's data sub language.

1.8 Who uses a DBMS

There are three broad classes of users who use a DBMS

1) Application Programmer

2) End Users

3) Database Administrators

1.9 Entity

An entity is any distinguishable real world object that is to be represented into the database. Each entity will have attributes and properties.

1.10 Relationship

It is defined as an association among entities or The Entities in a database are likely to interact with other entities. The inter connection between the entity sets are called relationship.

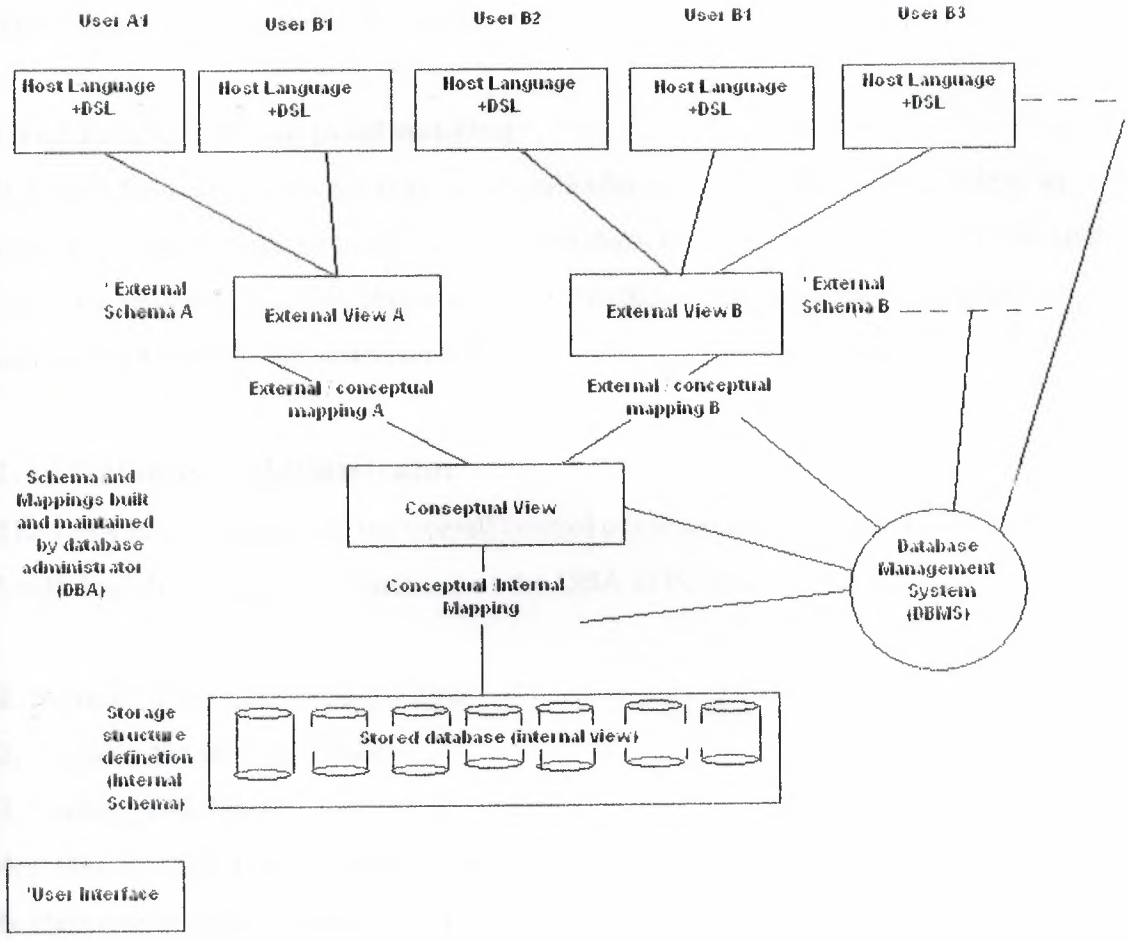

Figure 1.3 Detailed System Architecture

1.11 Mappings

Referring to figure 1.3 the person will observe two levels of mapping in the architecture one from the conceptual level to the internal level and one from the external level to the conceptual level.

1.11.1 Internal / Conceptual mapping

It defines the correspondence between the conceptual view and the stored database; It specifies how conceptual records and fields are represented at the internal level. If the structure of the stored database is changed then the conceptual internal mapping must be changed accordingly so that the conceptual schema can remain invariant. In other words

the effects of such changes must be isolated below the conceptual level in order that data independence might be preserved.

1.11.2 External / Conceptual mapping

It defines the correspondence between a particular external and conceptual view .In general the differences that can exist between these two levels are similar to those that can exist between these two levels are similar to those that can exist between the conceptual view and the stored database.

1.12 Database Administrator

The DBA is responsible for the overall control of the system at a technical level. Now I will describe some of the functions of the DBA in Points.

1. Defining The Conceptual Schema
2. Defining the Internal Schema
3. Liaising with Users
4. Defining Security and Integrity Rules
5. Defining Backup and Recovery Procedures
6. Monitoring Performance & Responding to Changing Requirements

1.13 The Database management System

The Database management System is a software that handles all access to the database The functions of DBMS are as follows

1. Data definition
2. Data Manipulation
3. Data Security and Integrity
4. Data Recovery and concurrency
5. Data Dictionary

1.14Relational Database Management System

A database is an integrated collection of related data. Given a specific data item, the structure of a database facilitates the access to data related to it. A relational database is a type of database based in the relational model. A relational database management system is the software that manages a relational database. These systems come in several varieties, ranging from single-user desktop systems to full-featured, global, enterprise-wide systems, such as Oracle8.

1.15 RDBMS Components

Two important pieces of an RDBMS architecture are the kernel, which is the software, and the data dictionary, which consists of the system-level data structures used by the kernel to manage the database.

1.16 The RDBMS Kernel

You might think of an RDBMS as an operating system or set of subsystems, designed specifically for controlling data access, its primary functions are storing, retrieving, and securing data. Like an operating system, Oracle8i manages and controls access to a given set of resources for concurrent database users. The subsystems of an RDBMS closely resemble those of a host operating system and tightly integrate with the host's services for machine-level access to resources such as memory, CPU, devices, and file structures. An RDBMS such as Oracle8i maintains its own list of authorized users and their associated privileges, manages memory caches and paging, controls locking for concurrent resource usage, dispatches and schedules user requests, and manages space usage within its tablespace structures. Figure 1.1 illustrates the primary subsystems of the Oracle8i kernel that manage the database.

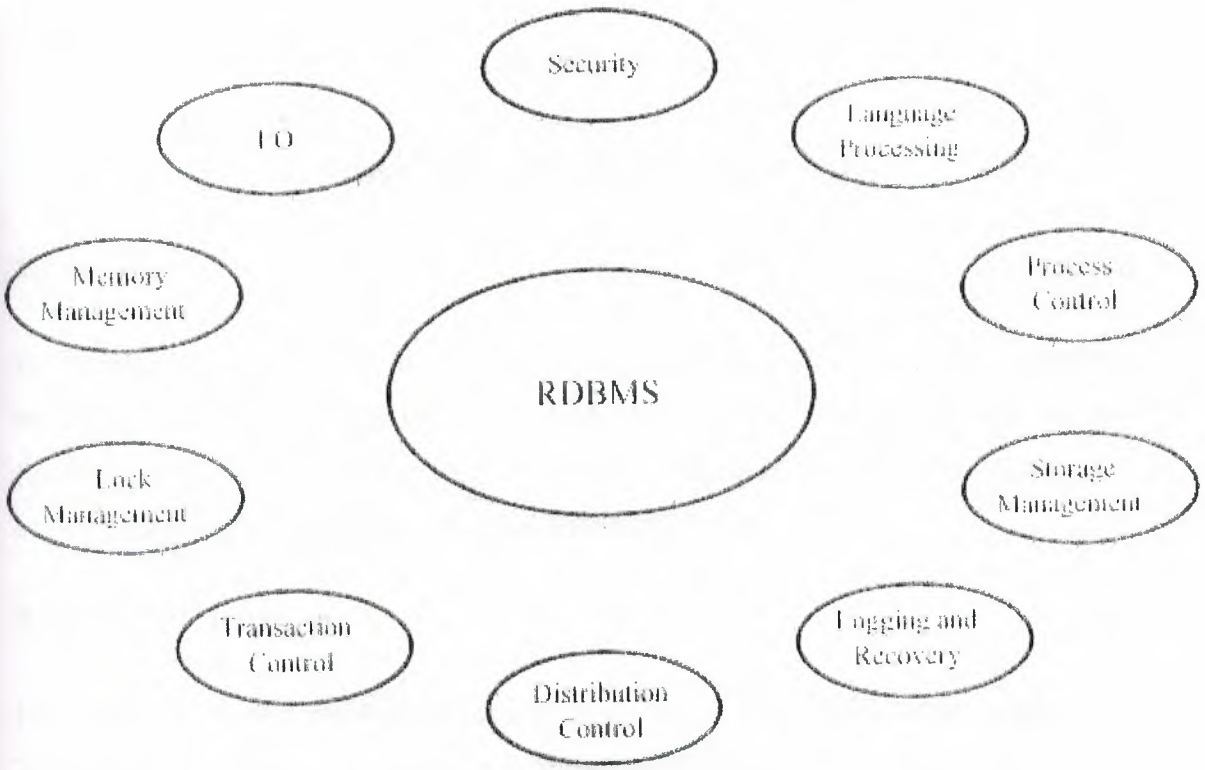

Figure 1.2 General architecture of a RDBMS ORACLE 8*i*

CHAPTER 2

Oracle and Java Interconnectivity

2.1 Overview

In this chapter I will briefly give a detail view of Oracle RDBMS and the connectivity of RDBMS with Java .Though topics I have covered in this chapter are the most fundamental parts of ORACLE RDBMS as well as those of Java .On other hand Java is a very powerful tool to develop a system based on database .

2.2 ORACLE Database

Physically, an Oracle database is nothing more than a set of files somewhere on disk. The physical location of these files is irrelevant to the function of the database. The files are binary files that we can only access using the Oracle kernel software. Querying data in the database files is typically done with one of the Oracle tools using the Structured Query Language.

Logically, the database is divided into a set of Oracle user accounts, each of which is identified by a username and password unique to that database. Tables and other objects are owned by one of these Oracle users, and access to the data is only available by logging in to the database using an Oracle username and password. Without a valid username and password for the database, you are denied access to anything on the database. The Oracle username and password is different from the operating system username and password.

2.3 ORACLE FILES

In this part, I discuss the different types of files that Oracle uses on the hard disk drive of any machine.

2.3.1 Database Files

The database files hold the actual data and are typically the largest in size, from a few megabytes to many gigabytes. The other files support the rest of the architecture. Depending on their sizes, the tables and other objects for all the user accounts can obviously go in one database file, but that's not an ideal situation because it does not make the database structure very flexible for controlling access to storage for different Oracle users, putting the database on different disk drives, or backing up and restoring just part of the database.

We must have at least one database file, but usually, we have many more than one. In terms of accessing and using the data in the tables and other objects, the number or location of the files is immaterial. The database files are fixed in size and never grow bigger than the size at which they were created.

2.3.2 Control Files

Any database must have at least one control file, although we typically have more than one to guard against loss. The control file records the name of the database, the date and time it was created, the location of the database and redo logs, and the synchronization information to ensure that all three sets of files are always in step. Every time we add a new database or redo log file to the database, the information is recorded in the control files.

2.3.3 Redo Logs

Any database must have at least two redo logs. These are the journals for the database, the redo logs record all changes to the user objects or system objects. If any type of failure occurs, such as loss of one or more database files, we can use the changes recorded in the redo logs to bring the database to a consistent state without losing any committed transactions. In the case of non-data loss failure, such as a machine crash, Oracle can apply the information in the redo logs automatically without intervention from the database administrator. The SMON background process automatically reapplies the committed changes in the redo logs to the database files.

Like the other files used by Oracle, the redo log files are fixed in size and never grow dynamically from the size at which they were created.

2.4 SYSTEM AND USER PROCESSES

In this part, I discuss some of the Oracle system processes that must be running for the database to be useable, including the optional processes and the processes that are created for users connecting to the Oracle database.

2.4.1 Mandatory System Processes

The four Oracle system processes that must always be up and running for the database to be useable include DBWR (Database Writer), LGWR (Log Writer), SMON (System Monitor), and PMON (Process Monitor).

2.4.1.1 DBWR (Database Writer)

The database writer background process writes modified database blocks in the SGA to the database files. It reads only the blocks that have changed. These blocks are also called *dirty* blocks. The database writer writes out the least recently used blocks first. These blocks are not necessarily written to the database when the transaction commits, the only thing that always happens on a commit is that the changes are recorded and written to the online redo log files. The database blocks will be written out later when there are not enough buffers free in the SGA to read in a new block.

2.4.1.2 LGWR (Log Writer)

The log writer process writes the entries in the SGA's redo buffer for one or more transactions to the online redo log files. For example, when a transaction commits, the log writer must write out the entries in the redo log buffer to the redo log files on disk before the process receives a message indicating that the commit was successful. Once committed, the changes are safe on disk even though the modified database blocks are still in the SGA's database buffer area waiting to be written out by DBWR. The SMON can always reapply the changes from the redo logs if the memory's most up-to-date copy of the database blocks is lost.

2.4.1.3 SMON (System Monitor)

The system monitor process looks after the instance. If two transactions are both waiting for each other to release locks and neither of them can continue known as a *deadlock* or *deadly embrace*, SMON detects the situation and one of the processes receives an error message indicating that a deadlock has occurred.

SMON also releases temporary segments that are no longer in use by the user processes which caused them to be created.

During idle periods, SMON compacts the free-space fragments in the database files, making it easier and simpler for Oracle to allocate storage for new database objects or for existing database objects to grow.

2.4.1.4 PMON (Process Monitor)

The process monitor monitors the user processes. If any failure occurs with the user processes, PMON automatically rolls back the work of the user process since the transaction started. It releases any locks taken out and other system resources taken up by the failed process.

2.4.2 User Processes

User processes logically consist of two halves. The Oracle server code, which translates and executes SQL statements and reads the database files and memory areas, and the tool-specific code, which is the executable code for the tool that is used. The server code is the same regardless of the tool that is executing the SQL statement, the same steps are involved. The server code is sometimes known as the Oracle kernel code.

We can configure the user processes in Oracle three different ways, all of which could coexist for the same instance. These three configurations are single task, dedicated server, or multi-threaded server.

2.4.2.1 Single Task

In the single-task configuration, the tool-specific code and database server code are both configured into one process running on the machine. Each connection to the database has one user process running on the machine.

2.4.2.2 Dedicated Server Processes

In the dedicated server configuration, the two parts of a user process are implemented as two separate processes running on the machine. They communicate with each other using the machine's interprocess communication mechanisms. Each connection to the database has two processes running on the machine. The Oracle kernel software in one process is sometimes called the shadow process.

This configuration is common for UNIX platforms because the operating system cannot protect the Oracle code and memory areas from the application code. It is also common for client/server configurations where the server code resides on the server machine and the tool-specific code runs on the client machine with communication over a network. The way the two component parts of one logical process communicate is fundamentally the same as if one process were implemented on the same machine, except that the two halves of the logical process happen to reside on two machines and communicate over the network using Net8 rather than the interprocess communication mechanisms of the operating system.

The dedicated server configuration can be wasteful because memory is allocated to the shadow process and the number of processes that must be serviced on the machine increases, even when the user is not making any database requests. The dedicated server will only process requests from one associated client process.

2.4.2.3 The Multi-Threaded Server

The multi-threaded server configuration enables one Oracle server process to perform work for many user processes. This overcomes the drawbacks of the dedicated server configuration. It reduces the number of processes running and the amount of memory

used on the machine and can improve system performance. The multi-threaded server introduces two new types of system processes that support this part of the architecture.

Using one of the shared server processes that comes as part of the multi-threaded server configuration is not appropriate when a user process is making many database requests such as an export backup of the database. For that process, we could use a dedicated server. A mixture of both configurations can coexist.

2.5 ORACLE Memory

In this part, I discuss how Oracle uses the machine's memory. Generally, the greater the real memory available to Oracle, the quicker the system runs.

2.5.1 System Global Area (SGA)

The system global area, sometimes known as the shared global area, is for data and control structures in memory that can be shared by all the Oracle background and user processes running on that instance. Each Oracle instance has its own SGA. In fact, the SGA and background processes is what defines an instance. The SGA memory area is allocated when the instance is started, and it's flushed and deallocated when the instance is shut down.

The contents of the SGA are divided into three main areas, the database buffer cache, the shared pool area, and the redo cache. The size of each of these areas is controlled by parameters in the INIT.ORA file. The bigger you can make the SGA and the more of it that can fit into the machine's real memory as opposed to virtual memory, the quicker your instance will run.

2.5.1.1 Database Buffer Cache

The database buffer cache of the SGA holds Oracle blocks that have been read in from the database files. When one process reads the blocks for a table into memory, all the processes for that instance can access those blocks.

If a process needs to access some data, Oracle checks to see if the block is already in this cache. If the Oracle block is not in the buffer, it must be read from the database files into the buffer cache. The buffer cache must have a free block available before the data block can be read from the database files.

The Oracle blocks in the database buffer cache in memory are arranged with the most recently used at one end and the least recently used at the other. This list is constantly changing as the database is used. If data must be read from the database files into memory, the blocks at the least recently used end are written back to the database files first. The DBWR process is the only process that writes the blocks from the database buffer cache to the database files. The more database blocks you can hold in real memory, the quicker your instance will run.

2.5.1.2 Redo Cache

The online redo log files record all the changes made to user objects and system objects. Before the changes are written out to the redo logs, Oracle stores them in the redo cache memory area. For example, the entries in the redo log cache are written down to the online redo logs when the cache becomes full or when a transaction issues a commit. The entries for more than one transaction can be included together in the same disk write to the redo log files.

The LGWR background process is the only process that writes out entries from this redo cache to the online redo log files.

2.5.1.3 Shared Pool Area

The shared pool area of the SGA has two main components, the SQL area and the dictionary cache. You can alter the size of these two components only by changing the size of the entire shared pool area.

2.5.1.4 SQL Area

A SQL statement sent for execution to the database server must be parsed before it can execute. The SQL area of the SGA contains the binding information, run-time buffers, parse tree, and execution plan for all the SQL statements sent to the database server. Because the shared pool area is a fixed size, you might not see the entire set of statements that have been executed since the instance first came up, Oracle might have flushed out some statements to make room for others.

If a user executes a SQL statement, that statement takes up memory in the SQL area. If another user executes exactly the same statement on the same objects, Oracle doesn't need to reparse the second statement because the parse tree and execution plan is already in the SQL area. This part of the architecture saves on reparsing overhead. The SQL area is also used to hold the parsed, compiled form of PL/SQL blocks, which can also be shared between user processes on the same instance.

2.5.1.5 Dictionary Cache

The dictionary cache in the shared pool area holds entries retrieved from the Oracle system tables, otherwise known as the Oracle data dictionary. The data dictionary is a set of tables located in the database files, and because Oracle accesses these files often, it sets aside a separate area of memory to avoid disk I/O.

The cache itself holds a subset of the data from the data dictionary. It is loaded with an initial set of entries when the instance is first started and then populated from the database data dictionary as further information is required. The cache holds information about all the users, the tables and other objects, the structure, security, storage, and so on.

The data dictionary cache grows to occupy a larger proportion of memory within the shared pool area as needed, but the size of the shared pool area remains fixed.

2.5.2 Process Global Area

The process global area, sometimes called the program global area or PGA, contains data and control structures for one user or server process. There is one PGA for each user process to the database.

The actual contents of the PGA depend on whether the multi-threaded server configuration is implemented, but it typically contains memory to hold the session's variables, arrays, some rows results, and other information. If you're using the multi-threaded server, some of the information that is usually held in the PGA is instead held in the common SGA.

The size of the PGA depends on the operating system used to run the Oracle instance, and once allocated, it remains the same. Memory used in the PGA does not increase according to the amount of processing performed in the user process. The database administrator can control the size of the PGA by modifying some of the parameters in the instance parameter file INIT.ORA.

2.6 ORACLE ACCESS WITH JDBC

Java is designed to be platform independent. A pure Java program written for a Windows machine will run without recompilation on a Solaris Sparc, an Apple Macintosh, or any platform with the appropriate Java virtual machine.

JDBC extends this to databases. If we write a Java program with JDBC, given the appropriate database driver, that program will run against any database without having to recompile the Java code. Without JDBC, our Java code would need to run platform specific native database code, thus violating the Java motto, Write Once, Run Anywhere.
JDBC allows us to write Java code, and leave the platform specific code to the driver. In the event we change databases, we simply change the driver used by our Java code and we are immediately ready to run against the new database.

JDBC is a rich set of classes that give us transparent access to a database with a single application programming interface, or API. This access is done with plug-in platform-specific modules, or drivers. Using these drivers and the JDBC classes, our programs will be able to access consistently any database that supports JDBC, giving us total freedom to concentrate on our applications and not to worry about the underlying database.

All access to JDBC data sources is done through SQL. Sun has concentrated on JDBC issuing SQL commands and retrieving their results in a consistent manner. Though we gain so much ease by using this SQL interface, we do not have the raw database access that we might be used to. With the classes we can open a connection to a database, execute SQL statements, and do what we will with the results.

2.6.1 Driver Types

As mentioned above, our Java JDBC code is portable because the database specific code is contained in a Java class known as the driver. The two most common kinds of driver for connecting to an Oracle database are the thin driver and the OCI driver.

2.6.1.1 Thin driver

The thin driver is known as a Type IV driver, it is a pure Java driver that connects to a database using the database's native protocol. While we can use the thin driver in any environment, the Type IV driver is intended for use in Java applets and other client-side programs. A Java client can be run on any platform. For that reason, the JDBC driver downloaded with an applet or used by a Java client may not have access to platform native code and must be pure Java.

2.6.1.2 OCI8 driver

The OCI8 driver is known as a Type II driver. It uses platform native code to call the database. Because it uses a native API, it can connect to and access a database faster than the thin driver. For the same reason, the Type II driver cannot be used where the

program does not have access to the native API. This usually applies to applets and other client programs which may be deployed on any arbitrary platform.

2.6.2 The DriverManager Class

The cornerstone of the JDBC package is the `DriverManager` class. This class keeps track of all the different available database drivers. We won't usually see the `DriverManager`'s work, though. This class mostly works behind the scenes to ensure that everything is cool for our connections.

The `DriverManager` maintains a `Vector` that holds information about all the drivers that it knows about. The elements in the `Vector` contain information about the driver such as the class name of the `Driver` object, a copy of the actual `Driver` object, and the `Driver` security context.

The `DriverManager`, while not a `static` class, maintains all `static` instance variables with `static` access methods for registering and unregistering drivers. This allows the `DriverManager` never to need instantiation. Its data always exists as part of the Java runtime. The drivers managed by the `DriverManager` class are represented by the `Driver` class.

2.6.3 The `Driver` Class

If the cornerstone of JDBC is the `DriverManager`, then the `Driver` class is most certainly the bricks that build the JDBC. The `Driver` is the software wedge that communicates with the platform-dependent database, either directly or using another piece of software. How it communicates really depends on the database, the platform, and the implementation.

It is the `Driver`'s responsibility to register with the `DriverManager` and connect with the database. Database connections are represented by the `Connection` class.

2.6.4 The `Connection` Class

The `Connection` class encapsulates the actual database connection into an easy-to-use package. Sticking with our foundation building analogy here, the `Connection` class is the mortar that binds the JDBC together. It is created by the `DriverManager` when its `getConnection()` method is called. This method accepts a database connection URL and returns a database `Connection` to the caller.

When we call the `getConnection()` method, the `DriverManager` asks each driver that has registered with it whether the database connection URL is valid. If one driver responds positively, the `DriverManager` assumes a match. If no driver responds positively, an `SQLException` is thrown. The `DriverManager` returns the error "no suitable driver," which means that of all the drivers that the `DriverManager` knows about, not one of them could figure out the URL you passed to it.

Assuming that the URL was good and a `Driver` loaded, then the `DriverManager` will return a `Connection` object to us. What can we do with a `Connection` object? Not much. This class is nothing more than an encapsulation of our database connection. It is a factory and manager object, and is responsible for creating and managing `Statement` objects.

2.6.5 The `Statement` Class

Picture the `Connection` as an open pipeline to our database. Database transactions travel back and forth between our program and the database through this pipeline. The `Statement` class represents these transactions.

The `Statement` class encapsulates SQL queries to our database. Using several methods, these calls return objects that contain the results of our SQL query. When we execute an SQL query, the data that is returned to us is commonly called the result set.

2.6.6 The ResultSet Class

As we've probably guessed, the `ResultSet` class encapsulates the results returned from an SQL query. Normally, those results are in the form of rows of data. Each row contains one or more columns. The `ResultSet` class acts as a cursor, pointing to one record at a time, enabling us to pick out the data we need.

2.7 SQLJ

SQLJ enables us to embed static SQL operations in Java code in a way that is compatible with the Java design philosophy. A SQLJ program is a Java program containing embedded static SQL statements that comply with the ANSI-standard SQLJ Language Reference syntax. Static SQL operations are predefined, the operations themselves do not change in real-time as a user runs the application, although the data values that are transmitted can change dynamically. Typical applications contain much more static SQL than dynamic SQL. Dynamic SQL operations are *not* predefined, the operations themselves can change in real-time and require direct use of JDBC statements. However, we can use SQLJ statements and JDBC statements in the same program.

SQLJ consists of both a translator and a runtime component and is smoothly integrated into our development environment. The developer runs the translator, with translation, compilation, and customization taking place in a single step when the sqlj front-end utility is run. The translation process replaces embedded SQL with calls to the SQLJ runtime, which implements the SQL operations. In standard SQLJ this is typically, but not necessarily, performed through calls to a JDBC driver. In the case of an Oracle database, we would typically use an Oracle JDBC driver. When the end user runs the SQLJ application, the runtime is invoked to handle the SQL operations.

The Oracle SQLJ translator is conceptually similar to other Oracle precompilers and allows the developer to check SQL syntax, verify SQL operations against what is available in the schema, and check the compatibility of Java types with corresponding

database types. In this way, errors can be caught by the developer instead of by a user at runtime.

The SQLJ methodology of embedding SQL operations directly in Java code is much more convenient and concise than the JDBC methodology. In this way, SQLJ reduces development and maintenance costs in Java programs that require database connectivity. When dynamic SQL is required, however, SQLJ supports interoperability with JDBC such that we can intermix SQLJ code and JDBC code in the same source file. Alternatively, we can use PL/SQL blocks within SQLJ statements for dynamic SQL.

2.7.1 SQLJ Components

Oracle SQLJ consists of two major components.

2.7.1.1 Oracle SQLJ Translator

This component is a precompiler that developers run after creating SQLJ source code. The translator, written in pure Java, supports a programming syntax that allows us to embed SQL operations inside SQLJ executable statements. SQLJ executable statements, as well as SQLJ declarations, are preceded by the #sql token and can be interspersed with Java statements in a SQLJ source code file. SQLJ source code file names must have the .sqlj extension.

The translator produces a .java file and one or more SQLJ profiles, which contain information about our SQL operations. SQLJ then automatically invokes a Java compiler to produce .class files from the .java file.

2.7.1.2 Oracle SQLJ Runtime

This component is invoked automatically each time an end user runs a SQLJ application. The SQLJ runtime, also written in pure Java, implements the desired actions of our SQL operations, accessing the database using a JDBC driver. The generic SQLJ standard does not require that a SQLJ runtime use a JDBC driver to access the database, however, the Oracle SQLJ runtime does require a JDBC driver, and, in fact, requires an Oracle JDBC driver if our application is customized with the default Oracle customizer.

In addition to the translator and runtime, there is a component known as the customizer. A customizer tailors our SQLJ profiles for a particular database implementation and vendor-specific features and datatypes. By default, the Oracle SQLJ front end invokes an Oracle customizer to tailor our profiles for an Oracle database and Oracle-specific features and datatypes.

When we use the Oracle customizer during translation, our application will require the Oracle SQLJ runtime and an Oracle JDBC driver when it runs.

2.7.1.3 SQLJ Profiles

SQLJ profiles are serialized Java resources generated by the SQLJ translator, which contain details about the embedded SQL operations in our SQLJ source code. The translator creates these profiles, then either serializes them and puts them into binary resource files, or puts them into .class files according to our translator option settings.

SQLJ profiles are used in implementing the embedded SQL operations in our SQLJ executable statements. Profiles contain information about our SQL operations and the types and modes of data being accessed. A profile consists of a collection of entries, where each entry maps to one SQL operation. Each entry fully specifies the corresponding SQL operation, describing each of the parameters used in executing this instruction.

SQLJ generates a profile for each connection context class in our application, where, typically, each connection context class corresponds to a particular set of SQL entities we use in our database operations. The SQLJ standard requires that the profiles be of standard format and content. Therefore, for our application to use vendor-specific extended features, our profiles must be customized. By default, this occurs automatically, with our profiles being customized to use Oracle-specific extended features.

2.7.2 Oracle Extensions to the SQLJ Standard

Beginning with Oracle8*i*, Oracle SQLJ supports the SQLJ ISO specification. Because the SQLJ ISO standard is a superset of the SQLJ ANSI standard, it requires a JDK 1.2

or later environment that complies with J2EE. The SQLJ ANSI standard requires only JDK 1.1.x. The Oracle SQLJ translator accepts a broader range of SQL syntax than the ANSI SQLJ Standard specifies.

The ANSI standard addresses only the SQL92 dialect of SQL, but allows extension beyond that. Oracle SQLJ supports Oracle's SQL dialect, which is a superset of SQL92. If we need to create SQLJ programs that work with other DBMS vendors, avoid using SQL syntax and SQL types that are not in the standard and, therefore, may not be supported in other environments.

2.7.3 Basic Translation Steps and Runtime Processing

2.7.3.1 Translation Steps

The following sequence of events occurs, presuming each step completes without fatal error.

1. The JVM invokes the SQLJ translator.

2. The translator parses the source code in the .sqlj file, checking for proper SQLJ syntax and looking for type mismatches between our declared SQL datatypes and corresponding Java host variables.

3. The translator invokes the semantics-checker, which checks the semantics of embedded SQL statements.

   The developer can use online or offline checking, according to SQLJ option settings. If online checking is performed, then SQLJ will connect to the database to verify that the database supports all the database tables, stored procedures, and SQL syntax that the application uses, and that the host variable types in the SQLJ application are compatible with datatypes of corresponding database columns.

4. The translator processes our SQLJ source code, converts SQL operations to SQLJ runtime calls, and generates Java output code and one or more SQLJ profiles. A separate profile is generated for each connection context class in our

source code, where a different connection context class is typically used for each interrelated set of SQL entities that we use in our database operations.

5. The JVM invokes the Java compiler, which is usually, but not necessarily, the *standard javac provided with the Sun Microsystems JDK.*

6. The compiler compiles the Java source file generated in step 4 and produces Java .class files as appropriate. This will include a .class file for each class we defined, a .class file for each of our SQLJ declarations, and a .class file for the profile-keys class.

7. The JVM invokes the Oracle SQLJ customizer or other specified customizer.

8. The customizer customizes the profiles generated in step 4.

2.7.3.2 Runtime Processing

When a user runs the application, the SQLJ runtime reads the profiles and creates "connected profiles", which incorporate database connections. Then the following occurs each time the application must access the database.

1. SQLJ-generated application code uses methods in a SQLJ-generated profile-keys class to access the connected profile and read the relevant SQL operations. There is mapping between SQLJ executable statements in the application and SQL operations in the profile.

2. The SQLJ-generated application code calls the SQLJ runtime, which reads the SQL operations from the profile.

3. The SQLJ runtime calls the JDBC driver and passes the SQL operations to the driver.

4. The SQLJ runtime passes any input parameters to the JDBC driver.

5. The JDBC driver executes the SQL operations.

6. If any data is to be returned, the database sends it to the JDBC driver, which sends it to the SQLJ runtime for use by our application.

2.7.4 SQLJ Declarations

A SQLJ declaration consists of the #sql token followed by the declaration of a class. SQLJ declarations introduce specialized Java types into our application. There are currently two kinds of SQLJ declarations, iterator declarations and connection context declarations, defining Java classes.

Iterator declarations define iterator classes. Iterators are conceptually similar to JDBC result sets and are used to receive multi-row query data. An iterator is implemented as an instance of an iterator class.

Connection context declarations define connection context classes. Each connection context class is typically used for connections whose operations use a particular set of SQL entities. That is to say, instances of a particular connection context class are used to connect to schemas that include SQL entities with the same names and characteristics. SQLJ implements each database connection as an instance of a connection context class.

context expression must *not* be preceded by a colon.

2.7.5 Stored Procedure and Function Calls

SQLJ provides convenient syntax for calling stored procedures and stored functions in the database. These procedures and functions could be written in Java, PL/SQL, or any other language supported by the database.

A stored function requires a result expression in our SQLJ executable statement to accept the return value and can optionally take input, output, or input-output parameters as well.

A stored procedure does not have a return value but can optionally take input, output, or input-output parameters. A stored procedure can return output through any output or input-output parameter.

2.7.6 Multithreading in SQLJ

We can use SQLJ in writing multithreaded applications; however, any use of multithreading in our SQLJ application is subject to the limitations of our JDBC driver. This includes any synchronization limitations.

We are required to use a different execution context instance for each thread. We can accomplish this in one of two ways.

1. Specify connection context instances for our SQLJ statements such that a different connection context instance is used for each thread. Each connection context instance automatically has its own default execution context instance.
2. If we are using the same connection context instance with multiple threads, then declare additional execution context instances and specify execution context instances for our SQLJ statements such that a different execution context instance is used for each thread.

If we are using one of the Oracle JDBC drivers, multiple threads can use the same connection context instance as long as different execution context instances are specified and there are no synchronization requirements directly visible to the user. However, that database access is sequential, only one thread is accessing the database at any given time.

If a thread attempts to execute a SQL operation that uses an execution context that is in use by another operation, then the thread is blocked until the current operation completes. If an execution context were shared between threads, the results of a SQL operation performed by one thread would be visible in the other thread. If both threads were executing SQL operations, a race condition might occur, the results of an execution in one thread might be overwritten by the results of an execution in the other thread before the first thread had processed the original results. This is why multiple threads are not allowed to share an execution context instance.

2.7.7 SQLJ and JDBC Interoperability

We can use SQLJ statements for static SQL operations, but not for dynamic operations. We can, however, use JDBC statements for dynamic SQL operations, and there might be situations where our application will require both static and dynamic SQL operations. SQLJ allows us to use SQLJ statements and JDBC statements concurrently and provides interoperability between SQLJ constructs and JDBC constructs.

Two kinds of interactions between SQLJ and JDBC are particularly useful:

- between SQLJ connection contexts and JDBC connections
- between SQLJ iterators and JDBC result sets

2.7.7.1 Converting from Connection Contexts to JDBC Connections

If we want to perform a dynamic SQL operation through a database connection that we have established in SQLJ, then we must convert the SQLJ connection context instance to a *JDBC connection instance.*

Any connection context instance in a SQLJ application, whether an instance of the sqlj. runtime.ref.Default Context class or of a declared connection context class, contains an underlying JDBC connection instance and a getConnection() method that returns that JDBC connection instance. Use the JDBC connection instance to create JDBC statement objects if you want to use any dynamic SQL operations.

2.7.7.2 Converting from JDBC Connections to Connection Contexts

If we initiate a connection as a JDBC Connection or OracleConnection instance but later want to use it as a SQLJ connection context instance, then we can convert the JDBC connection instance to a SQLJ connection context instance.

The DefaultContext class and all declared connection context classes have a constructor that takes a JDBC connection instance as input and constructs a SQLJ connection context instance.

2.7.7.3 Shared Connections

A SQLJ connection context instance and the associated JDBC connection instance share the same underlying database connection. When we get a JDBC connection instance from a SQLJ connection context instance, the Connection instance inherits the state of the connection context instance. Among other things, the Connection instance will retain the auto-commit setting of the connection context instance.

When we construct a SQLJ connection context instance from a JDBC connection instance, the connection context instance inherits the state of the Connection instance. Among other things, the connection context instance will retain the auto-commit setting of the Connection instance.

Given a SQLJ connection context instance and associated JDBC connection instance, calls to methods that alter session state in one instance will also affect the other instance, because it is actually the underlying shared database session that is being altered.

Because there is just a single underlying database connection, there is also a single underlying set of transactions. A COMMIT or ROLLBACK operation in one connection instance will affect any other connection instances that share the same underlying connection.

2.7.8 SQLJ In the Server

SQLJ code, as with any Java code, can run in the Oracle8*i* server in stored procedures, stored functions, triggers, Enterprise JavaBeans, or CORBA objects. Database access is through a server-side implementation of the SQLJ runtime in combination with the Oracle JDBC server-side internal driver.

In addition, an embedded SQLJ translator in the Oracle8*i* server is available to translate SQLJ source files directly in the server.

Considerations for running SQLJ in the server include several server-side coding issues as well as decisions about where to translate our code and how to load it into the server. We must also be aware of how the server determines the names of generated output. We

can either translate and compile on a client and load the class and resource files into the server, or we can load .sqlj source files into the server and have the files automatically translated by the embedded SQLJ translator.

The embedded translator has a different user interface than the client-side translator. Supported options can be specified using a database table, and error output is to a database table. Output files from the translator, .java and .ser, are transparent to the developer.

2.7.8.1 Creating SQLJ Code for Use within the Server

With few exceptions, writing SQLJ code for use within the target Oracle8*i* server is identical to writing SQLJ code for client-side use. The few differences are due to Oracle JDBC characteristics or general Java characteristics in the server, rather than being specific to SQLJ.

2.7.8.2 Database Connections within the Server

The concept of connecting to a server is different when our SQLJ code is running within this server itself, there is no explicit database connection. By default, an implicit channel to the database is employed for any Java program running in the server. We do not have to initialize this connection, it is automatically initialized for SQLJ programs. We do not have to register or specify a driver, create a connection instance, specify a default connection context, specify any connection objects for any of our #sql statements, or close the connection.

The internal driver does not support auto-commit functionality, the auto-commit setting is ignored within the server. Use explicit COMMIT or ROLLBACK statements to implement or cancel your database updates.

2.7.8.3 Name Resolution in the Server

Class loading and name resolution in the server follow a very different paradigm than on a client, because the environments themselves are very different. Java name resolution in the Oracle8*i* JVM includes the following:

1. Class resolver specs, which are schema lists to search in resolving a class schema object.
2. The resolver, which maintains mappings between class schema objects that reference each other in the server.

A class schema object is said to be resolved when all of its external references to Java names are bound. In general, all the classes of a Java program should be compiled or loaded before they can be resolved.

When all the class schema objects of a Java program in the server are resolved and none of them have been modified since being resolved, the program is effectively pre-linked and ready to run.

A class schema object must be resolved before Java objects of the class can be instantiated or methods of the class can be executed.

2.7.9 SQL Names versus Java Names

SQL names such as names of source, class, and resource schema objects are not global in the way that Java names are global. The Java Language Specification directs that package names use Internet naming conventions to create globally unique names for Java programs. By contrast, a fully qualified SQL name is interpreted only with respect to the current schema and database.

Because of this inherent difference, SQL names must be interpreted and processed differently from Java names. SQL names are relative names and are interpreted from the point of view of the schema where a program is executed. This is central to how the program binds local data stored at that schema. Java names are global names, and the classes that they designate can be loaded at any execution site, with reasonable expectation that those classes will be classes that were used to compile the program.

2.8 Introduction to Net8

Net8 enables the machines in our network to communicate with one another. It facilitates and manages communication sessions between a client application and a remote database. Specifically, Net8 performs three basic operations.

1. **Connection:** opening and closing connections between a client or a server acting as a client and a database server over a network protocol.
2. **Data Transport:** packaging and sending data such as SQL statements and data responses so that it can be transmitted and understood between a client and a server.
3. **Exception Handling:** initiating interrupt requests from the client or server.

2.8.1 Advantages of Net8

Net8 provides the following benefits to users of networked applications.

2.8.1.1 Network Transparency

Net8 provides support for a broad range of network transport protocols including TCP/IP, SPX/IPX, IBM LU6.2, Novell, and DECnet. It does so in a manner that is invisible to the application user. This enables Net8 to interoperate across different types of computers, operating systems, and networks to transparently connect any combination of PC, UNIX, legacy, and other system without changes to the existing infrastructure.

2.8.1.2 Protocol Independence

Net8 enables Oracle applications to run over any supported network protocol by using the appropriate Oracle Protocol Adapter. Applications can be moved to another protocol stack by installing the necessary Oracle Protocol Adapter and the industry protocol stack. Oracle Protocol Adapters provide Net8 access to connections over specific protocols or networks. On some platforms, a single Oracle Protocol Adapter will

operate on several different network interface boards, allowing you to deploy applications in any networking environment.

2.8.1.3 Media/Topology Independence

When Net8 passes control of a connection to the underlying protocol, it inherits all media and/or topologies supported by that network protocol stack. This allows the network protocol to use any means of data transmission, such as Ethernet, Token Ring, or other, to accomplish low level data link transmissions between two machines.

2.8.1.4 Heterogeneous Networking

Oracle's client-server and server-server models provide connectivity between multiple network protocols using Oracle Connection Manager.

2.8.1.5 Large Scale Scalability

By enabling us to use advanced connection concentration and connection pooling features, Net8 makes it possible for thousands of concurrent users to connect to a server.

2.8.2 Net8 Features

Net8 Release 8.0 features several enhancements that extend scalability, manageability and security for the Oracle network.

2.8.2.1 Scalability

Scalability refers to the ability to support simultaneous network access by a large number of clients to a single server. With Net8, this is accomplished by optimizing the usage of network resources by reducing the number of physical network connections a server must maintain. Net8 offers improved scalability through two new features.

1. Connection pooling.
2. Connection concentration.

Both of these features optimize usage of server network resources to eliminate data access bottlenecks and enable large numbers of concurrent clients to access a single

server. Additionally, other enhancements such as a new buffering methods and asynchronous operations further improve Net8 performance.

2.8.2.2 Manageability Features

Net8 introduces a number of new features that will simplify configuration and administration of the Oracle network for both workgroup and enterprise environments.

For workgroup environments, Net8 offers simple configuration-free connectivity through installation defaults and a new name resolution feature called host naming. For enterprise environments, Net8 centralizes client administration and simplifies network management with Oracle Names. In addition to these new features, Net8 introduces the Oracle Net8 Assistant.

i. Host Naming

Host Naming refers to a new naming method which resolves service names to network addresses by enlisting the services of existing TCP/IP hostname resolution systems. Host Naming can eliminate the need for a local naming configuration file in environments where simple database connectivity is desired.

ii. Oracle Net8 Assistant

The Oracle Net8 Assistant is a new end user, stand-alone Java application that can be launched either as a stand-alone application or from the Oracle Enterprise Manager console. It automates client configuration and provides an easy-to-use interface as well as wizards to configure and manage Net8 networks.

Because the Oracle Net8 Assistant is implemented in Java, it is available on any platform that supports the Java Virtual Machine.

2.8.2.3 Oracle Trace Assistant

Net8 includes the Oracle Trace Assistant to help decode and analyze the data stored in Net8 trace files. The Oracle Trace Assistant provides an easy way to understand and take advantage of the information stored in trace files, it is useful for diagnosing

network problems and analyzing network performance. It can be used to better pinpoint the source of a network problem or identify a potential performance bottleneck.

2.8.2.4 Native Naming Adapters

Native Naming Adapters, previously bundled with the Advanced Networking Option, are now included with Net8. These adapters provide native support for industry-standard name services, including Sun NIS/Yellow Pages and Novell NetWare Directory Services (NDS).

2.8.3 Net8 Operations

Net8 is responsible for enabling communications between the cooperating partners in an Oracle distributed transaction, whether they be client-server or server-server. Specifically, Net8 provides three basic networking operations:

1. Connect Operations.
2. Data Operations.
3. Exception Operations.

2.8.4 Connect Operations

Net8 supports two types of connect operations.

2.8.4.1 Connecting to Servers

Users initiate a connect request by passing information such as a username and password along with a short name for the database service that they wish to connect. That short name, called a *service name*, is mapped to a network address contained in a *connect descriptor*. Depending upon our specific network configuration, this connect descriptor may be stored in one of the following.

1. A local names configuration file called TNSNAMES.ORA.
2. A Names Server for use by Oracle Names.
3. A native naming service such as NIS or DCE CDS.

Net8 coordinates its sessions with the help of a network listener.

2.8.4.2 Establishing Connections with the Network Listener

The network listener is a single process or task setup specifically to receive connection requests on behalf of an application. Listeners are configured to "listen on" an address specified in a listener configuration file for a database or non-database service. Once started, the listener will receive client connect requests on behalf of a service, and respond in one of three ways:

1. Bequeath the session to a new dedicated server process.
2. Redirect to an existing server process.
3. Refuse the session.

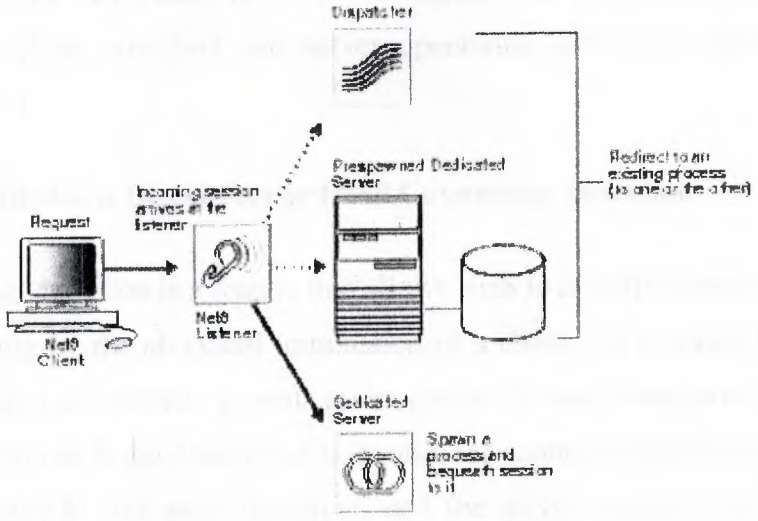

Figure2.1 Network Listener In a Typical Net8 Connection

2.8.5 Disconnecting from Servers

Requests to disconnect from the server can be initiated in the following ways.

2.8.5.1 User-Initiated Disconnect

A user can request a disconnection from the server when a client-server transaction completes. A server can also disconnect from a second server when all server-server data transfers have been completed, and no need for the link remains.

2.8.5.2 Additional Connection Request

If a client application is connected to a server and requires access to another user account on the same or other server, most Oracle tools will first disconnect the application from the server to which it is currently connected. Once the disconnection is completed, a connection request to the new user account on the appropriate server is initiated.

2.8.5.3 Abnormal Connection Termination

Other components will occasionally disconnect or abort communications without giving notice to Net8. In this event, Net8 will recognize the failure during its next data operation, and clean up client and server operations, effectively disconnecting the current operation.

2.8.5.4 Timer Initiated Disconnect or Dead Connection Detection

Dead connection detection is a feature that allows Net8 to identify connections that have been left hanging by the abnormal termination of a client. On a connection with dead connection detection enabled, a small probe packet is sent from server to client at a user-defined interval. If the connection is invalid, the connection will be closed when an error is generated by the send operation, and the server process will terminate the connection.

This feature minimizes the waste of resources by connections that are no longer valid. It also automatically forces a database rollback of uncommitted transactions and locks held by the user of the broken connection.

2.8.6 Data Operations

Net8 supports four sets of client-server data operations.

1. Send data synchronously.
2. Receive data synchronously.
3. Send data asynchronously.
4. Receive data asynchronously.

On the client side, a SQL dialogue request is forwarded using a send request in Net8. On the server side, Net8 processes a receive request and passes the data to the database. The opposite occurs in the return trip from the server.

Basic send and receive requests are synchronous. When a client initiates a request, it waits for the server to respond with the answer. It can then issue an additional request.

Net8 adds the capability to send and receive data requests asynchronously. This capability was added to support the Oracle shared server, also called a multi-threaded server, which requires asynchronous calls to service incoming requests from multiple clients.

2.8.7 Exception Operations

Net8 supports three types of exception operations.

1. Initiate a break over the connection.
2. Reset a connection for synchronization after a break.
3. Test the condition of the connection for incoming break.

The user controls only one of these three operations, that is, the initiation of a break. When the user presses the Interrupt key, the application calls this function. Additionally, the database can initiate a break to the client if an abnormal operation occurs, such as during an attempt to load a row of invalid data using SQL*Loader.

The other two exception operations are internal to products that use Net8 to resolve network timing issues. Net8 can initiate a test of the communication channel, for example, to see if new data has arrived. The reset function is used to resolve abnormal states, such as getting the connection back in synchronization after a break operation has occurred.

2.8.8 Net8 and the Transparent Network Substrate (TNS)

Net8 uses the Transparent Network Substrate and industry-standard networking protocols to accomplish its basic functionality. TNS is a foundation technology that is built into Net8 providing a single, common interface to all industry-standard protocols.

With TNS, peer-to-peer application connectivity is possible where no direct machine-level connectivity exists. In a peer-to-peer architecture, two or more computers can communicate with each other directly, without the need for any intermediary devices. In a peer-to-peer system, a node can be both a client and a server.

2.8.9 Net8 Architecture

Oracle networking environments are based on two concepts.

2.8.9.1 Distributed Processing

Oracle databases and client applications operate in what is known as a distributed processing environment. Distributed or cooperative processing involves interaction between two or more computers to complete a single data transaction. Applications such as an Oracle tool act as clients requesting data to accomplish a specific operation. Database servers store and provide the data.

In a typical network configuration, clients and servers may exist as separate logical entities on separate physical machines. This configuration allows for a division of labor where resources are allocated efficiently between a client workstation and the server machine. Clients normally reside on desktop computers with just enough memory to execute user friendly applications, while a server has more memory, disk storage, and processing power to execute and administer the database.

This type of client-server architecture also enables you to distribute databases across a network. A distributed database is a network of databases stored on multiple computers that appears to the user as a single logical database. Distributed database servers are connected by a database link, or path from one database to another. One server uses a database link to query and modify information on a second server as needed, thereby acting as a client to the second server.

2.8.9.2 Stack Communications in an Oracle networking environment

Stack communications allow Oracle clients and servers to share, modify, and manipulate data between themselves. The layers in a typical Oracle communications stack are similar to those of a standard OSI communications stack.

i. Client-Server Interaction

In an Oracle client-server transaction, information passes through the following layers

1. Client Application.
2. Oracle Call Interface.
3. Two Task Common.
4. Net8.
5. Oracle Protocol Adapters.
6. Network Specific Protocols.

Figure 2.6 depicts a typical communications stack in an Oracle networking environment.

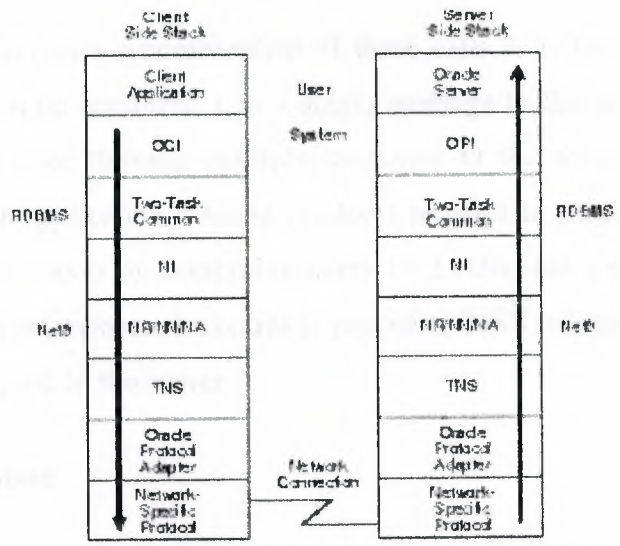

Figure2.6 Typical Communications Stack in an Oracle environment

1. Client Application

Oracle client applications provide all user-oriented activities, such as character or graphical user display, screen control, data presentation, application flow, and other application specifics. The application identifies database operations to send to the server and passes them through to the Oracle Call Interface.

2. Oracle Call Interface (OCI)

The OCI code contains all the information required to initiate a SQL dialogue between the client and the server. It defines calls to the server to:

1. Parse SQL statements for syntax validation.
2. Open a cursor for the SQL statement.
3. Bind client application variables into the server shared memory.
4. Describe the contents of the fields being returned based on the values in the server's data dictionary.
5. Execute SQL statements within the cursor memory space.
6. Fetch one or more rows of data into the client application.
7. Close the cursor.

The client application uses a combination of these calls to request activity within the server. OCI calls can be combined into a single message to the server, or they may be processed one at a time through multiple messages to the server, depending on the nature of the client application. Oracle products attempt to minimize the number of messages sent to the server by combining many OCI calls into a single message to the server. When a call is performed, control is passed to Net8 to establish the connection and transmit the request to the server.

3. Two-Task Common

Two-Task Common provides character set and data type conversion between different character sets or formats on the client and server. This layer is optimized to perform conversion only when required on a per connection basis.

At the time of initial connection, Two Task Common is responsible for evaluating differences in internal data and character set representations and determining whether conversions are required for the two computers to communicate.

4. Net8

Net8 provides all session layer functionality in an Oracle communications stack. It is responsible for establishing and maintaining the connection between a client application

and server, as well as exchanging messages between them. Net8 itself has three component layers that facilitate session layer functionality.

1. **Network Interface:** This layer provides a generic interface for Oracle clients, servers, or external processes to access Net8 functions. The NI handles the break and reset requests for a connection.
2. **Network Routing/ Network Naming/ Network Authentication:** NR provides routing of the session to the destination. This may include any intermediary destinations or 'hops', on the route to the server destination. NN resolves aliases to a Net8 destination address. NA negotiates any authentication requirement with the destination.
3. **Transparent Network Substrate:** TNS is an underlying layer of Net8 providing a common interface to industry standard protocols. TNS receives requests from Net8, and settles all generic machine-level connectivity issues, such as the location of the server or destination, whether one or more protocols will be involved in the connection, and how to handle interrupts between client and server based on the capabilities of each. The generic set of TNS functions passes control to an Oracle Protocol Adapter to make a protocol-specific call. Additionally, TNS supports encryption and sequenced cryptographic message digests to protect data in transit.

5. Oracle Protocol Adapters

Oracle Protocol Adapters are responsible for mapping TNS functionality to industry-standard protocols used in the client-server connection. Each adapter is responsible for mapping the equivalent functions between TNS and a specific protocol.

6. Network-Specific Protocols

All Oracle software in the client-server connection process require an existing network protocol stack to make the machine-level connection between the two machines. The network protocol is responsible only for getting the data from the client machine to the server machine, at which point the data is passed to the server-side Oracle Protocol Adapter.

7. Server-Side Interaction

Information passed from a client application across a network protocol is received by a similar communications stack on the server side. The process stack on the server side is the reverse of what occurred on the client side with information ascending through communication layers. The one operation unique to the server side is the act of receiving the initial connection through the network listener.

The following components above the Net8 session layer are different from those on the client side.

1. Oracle Program Interface
2. Oracle Server

1. Oracle Program Interface

The OPI performs a complementary function to that of the OCI. It is responsible for responding to each of the possible messages sent by the OCI. For example, an OCI request to fetch 25 rows would have an OPI response to return the 25 rows once they have been fetched.

2. Oracle Server

The Oracle Server side of the connection is responsible for receiving dialog requests from the client OCI code and resolving SQL statements on behalf of the client application. Once received, a request is processed and the resulting data is passed to the OPI for responses to be formatted and returned to the client application.

ii. Server-to-Server Interaction

When two servers communicate to complete a distributed transaction, the process, layers, and dialogues are the same as in the client-server scenario, except that there is no client application. The server has its own version of OCI, called the Network Program

Interface (NPI). The NPI interface performs all of the functions that the OCI does for clients, allowing a coordinating server to construct SQL requests for additional servers.

Chapter 3

Tourism Company Database

Overview

This chapter covers the topics related with the designing of the database .In this chapter I have discussed the ER Diagram of the Tourism Company ,the tables and the relationships between each one of them.

The tables in the Tourism Company database are as follows

1) Table_Customer
2) InstayType
3) Table_Tours
4) Tour_Days
5) Price_Table

3.1 Interconnectivity

The goal of this project is to make the Database system of Tourism Company. Student registration is a very time consuming process, but by using a flexible software, we can reduce this headache. The life of the staff can be made easy by the following features:

a) New information regarding clients, clients in stay time information ,Tour information, Clients tour price can be Entered.
b) Clients details can be viewed.
c) The data can be deleted.
d) The data can be changed.
e) The system is very easy to understand.
f) The system is very secure.

3.2 Program Implementation

The program is divided into two parts: The database part and the application part. For the database, to meet the security and flexibility issues, I have chosen Oracle database which is one of the main topic of my overall project as well.

For the application part, java is my choice due to its object-oriented functionality and security features.

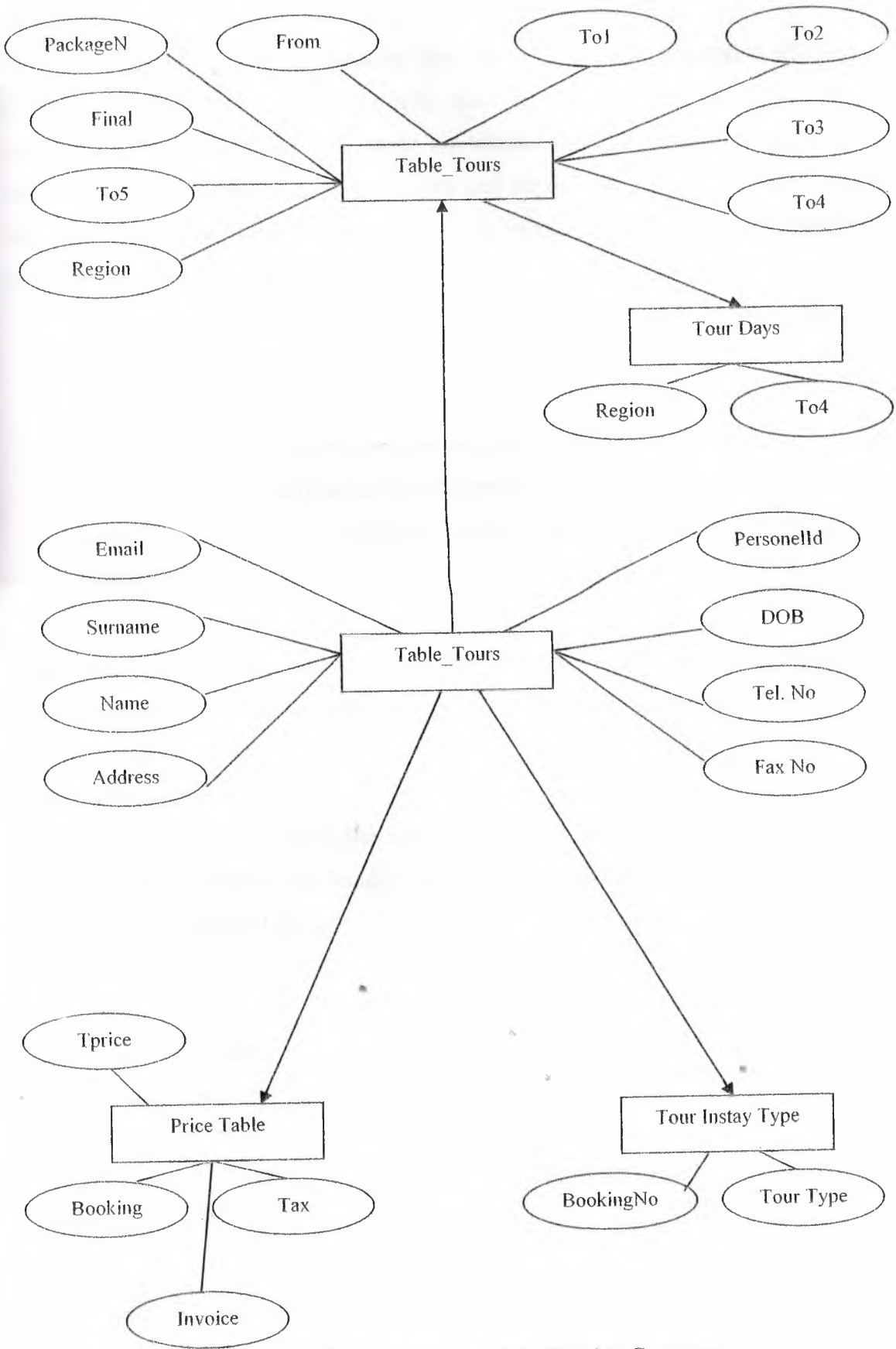

Fig 3.1 Shows the ER Diagram of the Tourism Company

3.2.1 Database

To create the database, first I installed the Oracle 9i Server's Enterprise edition and configured it properly to function correctly. After basic installation, I created a database on Oracle Server. In the database, there are tables which hold the Tour record. Each table has primary and foreign key constraints and are related to each other using one-to-one and one-to-many relationships for the data integrity issues. To create the tables, there are two ways I have used time by time.

1. The first way is by using the SQL*Plus, here we need to write the SQL commands to create the table, specify the fields, specify if we want any constraints.
2. The second way I have used is by using a Java application. In the Java application, using appropriate code, connect to the database and then write the table definition. After compiling, run the program, the table will be created in the database.

Once the table has been created, its definition can be altered any time. To alter the structure of the table, we can implement any of the two ways discussed above.

3.2.2 Application

In the application, I have used the standard packages in Java to create user interface, connect to the database, event handling and other functions used in my application. The important packages used are:

1. java.awt.*
2. java.awt.event.*
3. javax.swing.*
4. java.sql.*
5. java.util.*
6. java.lang.*

Java's Abstract Windowing Toolkit and Swing packages are to create the user interface. These packages have all the components necessary to give the application a viewable look. These packages contain the Frame, TextField, TextArea, Button, Label, List box,

Combo box, Check box, Option button, Horizontal and Vertical Scrollbar, Menu and Popup Menu components. The event handling mechanism in these packages is used to handle the events which occur during program execution and respond properly. All these components and event handling mechanism have been used in the application to give the flexibility to the user to understand and use the application easily.

The java.sql package is used for database connectivity and manipulation functionality. To connect to the database and to do transactions we need to import this package.

This project is for Tourism Company database where client its Instay data, Tour information , Clients data can be added, deleted, viewed and altered accurately with security. All the steps in the source code in Appendix A have been commented to understand the function of each step

.

Appendix.

Client delete

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import java.awt.*;

public class ClientDelete extends JFrame implements ActionListener{

private JTextField txtBookingNo = new JTextField();
private JTextField txtFirstName = new JTextField();
private JTextField txtLastName = new JTextField();
private JTextField txtAddress = new JTextField();
private JTextField txtTelNo = new JTextField();
private JTextField txtFaxNo = new JTextField();
private JTextField txtEmail = new JTextField();
private JTextField txtPersonId = new JTextField();
private JTextField txtDOB = new JTextField();
private JTextField txtSearch = new JTextField();

private JButton cmdSearch = new JButton();
private JLabel jLabel1 = new JLabel();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private JLabel jLabel4 = new JLabel();
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private JLabel jLabel7 = new JLabel();
private JLabel jLabel8 = new JLabel();
private JLabel jLabel9 = new JLabel();
private JLabel jLabel10 = new JLabel();
private JButton cmdOK = new JButton();

Connection connect;
String url;

 public ClientDelete()
 {
  try
  {

  this.getContentPane().setLayout(null);
  this.setSize(new Dimension(550, 450));
  txtBookingNo.setBounds(new Rectangle(190, 90, 140, 25));
  txtFirstName.setBounds(new Rectangle(190, 125, 140, 25));
  txtLastName.setBounds(new Rectangle(190, 160, 140, 25));
  txtAddress.setBounds(new Rectangle(190, 195, 140, 25));
```

```
}

public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdSearch){

        if(txtSearch.getText().length()==0){
                JOptionPane.showMessageDialog(null,"Enter Student Number to
Search.");
        }

        if(txtSearch.getText().length()!=0){
        try{
        Statement statement=connect.createStatement();

        String query="Select * from Table_customer "+
                "where BranchNo= '"+
                txtSearch.getText()+"'";

        ResultSet rs=statement.executeQuery(query);
        //display rs
        try{
                rs.next();
                int recordNumber=rs.getInt(1);
                if(recordNumber!=0){

                        txtBookingNo.setText(String.valueOf(recordNumber));
                        txtFirstName.setText(rs.getString(2));
                        txtLastName.setText(rs.getString(3));
                        txtAddress.setText(rs.getString(4));
                        txtTelNo.setText(String.valueOf(recordNumber));
                        txtFaxNo.setText(String.valueOf(recordNumber));
                        txtEmail.setText(rs.getString(10));
                        txtPersonId.setText(rs.getString(5));
                        txtDOB.setText(rs.getString(7));
                }
                else
                JOptionPane.showMessageDialog(null,"No Record Found");
        }catch(SQLException sqlex){
                sqlex.printStackTrace();
                JOptionPane.showMessageDialog(null,"No Such Record Found.");
        }
        statement.close();
        }
        catch(SQLException sqlex){
                sqlex.printStackTrace();
                //output.append(sqlex.toString());
        }
        cmdOK.addActionListener(this);
```

```
        }
        }

        if(ae.getSource()==cmdOK){

        try{
        Statement statement1=connect.createStatement();

        String query1="Delete * from Table_customer"+
                "where BranchNo= '"+
                txtSearch.getText()+"'";

                int result=statement1.executeUpdate(query1);
                if(result==1)
                        JOptionPane.showMessageDialog(null,"Changes Committed.");
                else
                        JOptionPane.showMessageDialog(null,"Could Not Update,
Please try some other time!!");
        statement1.close();
        }
        catch(SQLException sqlex){
                sqlex.printStackTrace();
        }
        }
}

public static void main(String args[]){

        ClientDelete app=new ClientDelete();

        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
        );
        }
}
```

*****Client New*****

```
import java.awt.*;
import javax.swing.border.BevelBorder;
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;

public class ClientNew extends JFrame implements ActionListener{
```

```
txtTelNo.setBounds(new Rectangle(190, 230, 140, 25));
txtFaxNo.setBounds(new Rectangle(190, 265, 140, 25));
txtEmail.setBounds(new Rectangle(190, 300, 140, 25));
txtPersonId.setBounds(new Rectangle(190, 335, 140, 25));
txtDOB.setBounds(new Rectangle(190, 370, 140, 25));
txtSearch.setBounds(new Rectangle(175, 20, 100, 25));
cmdSearch.setText("Search");
cmdSearch.setBounds(new Rectangle(300, 20, 90, 25));
cmdSearch.addActionListener(this);
jLabel1.setText("Booking Number:");
jLabel1.setBounds(new Rectangle(76, 20, 140, 25));
jLabel2.setText("Booking Number:");
jLabel2.setBounds(new Rectangle(90, 90, 100, 25));
jLabel3.setText("First Name:");
jLabel3.setBounds(new Rectangle(125, 125, 105, 20));
jLabel4.setText("Last Name:");
jLabel4.setBounds(new Rectangle(125, 155, 95, 25));
jLabel5.setText("Address:");
jLabel5.setBounds(new Rectangle(138, 195, 75, 20));
jLabel6.setText("TelNo:");
jLabel6.setBounds(new Rectangle(150, 230, 85, 25));
jLabel7.setText("FaxNo:");
jLabel7.setBounds(new Rectangle(150, 265, 85, 25));
jLabel8.setText("Email:");
jLabel8.setBounds(new Rectangle(152, 300, 85, 25));
jLabel9.setText("PersonId:");
jLabel9.setBounds(new Rectangle(135, 335, 85, 25));
jLabel10.setText("DOB:");
jLabel10.setBounds(new Rectangle(120, 385, 85, 25));
cmdOK.setText("OK");
cmdOK.setBounds(new Rectangle(350, 285, 70, 25));
this.getContentPane().add(cmdOK, null);
this.getContentPane().add(jLabel10, null);
this.getContentPane().add(jLabel9, null);
this.getContentPane().add(jLabel8, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(cmdSearch, null);
this.getContentPane().add(txtSearch, null);
this.getContentPane().add(txtDOB, null);
this.getContentPane().add(txtPersonId, null);
this.getContentPane().add(txtEmail, null);
this.getContentPane().add(txtFaxNo, null);
this.getContentPane().add(txtTelNo, null);
```

```
   this.getContentPane().add(txtAddress, null);
   this.getContentPane().add(txtLastName, null);
   this.getContentPane().add(txtFirstName, null);
   this.getContentPane().add(txtBookingNo, null);

        txtBookingNo.setEditable(false);
        txtFirstName.setEditable(false);
        txtLastName.setEditable(false);
        txtAddress.setEditable(false);
        txtTelNo.setEditable(false);
txtFaxNo.setEditable(false);
txtEmail.setEditable(false);
txtPersonId.setEditable(false);
txtDOB.setEditable(false);

}
   catch(Exception e)
   {
    e.printStackTrace();
   }

        setSize(500,400);
        setTitle("                    Client Delete Option");
        show();

        //setup database connection.
                try{
                        url="jdbc:oracle:thin:@faisalqureshi:1521:name";
                        String user="Faisal";
                        String passw="arsenal";
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        connect=DriverManager.getConnection(url,user,passw);
                }
                catch(ClassNotFoundException cnfex){
                        cnfex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(SQLException sqlex){
                        sqlex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(Exception ex){
                        ex.printStackTrace();
                        JOptionPane,showMessageDialog(null,"Could not connect to the
Database");
                }
```

```
        private JButton cmdRegister = new JButton();
        private JTextField txtBookingNo = new JTextField();
        private JTextField txtFirstName = new JTextField();
        private JTextField txtLastName = new JTextField();
        private JTextField txtAddress = new JTextField();
        private JTextField txtTelNo = new JTextField();
        private JTextField txtFaxNo = new JTextField();
        private JTextField txtEmail = new JTextField();
        private JTextField txtPersonId = new JTextField();
        private JTextField txtDOB = new JTextField();
        private JLabel jLabel1 = new JLabel();
        private JLabel jLabel2 = new JLabel();
        private JLabel jLabel3 = new JLabel();
        private JLabel jLabel4 = new JLabel();
        private JLabel jLabel5 = new JLabel();
        private JLabel jLabel6 = new JLabel();
        private JLabel jLabel7 = new JLabel();
        private JLabel jLabel8 = new JLabel();
        private JLabel jLabel9 = new JLabel();
        private Connection connect;
        String url;

public ClientNew(){
try
{
jbInit();
        setSize(550,450);
        setTitle("                              New Client Entry");
        show();

        //setup database connection.
                try{
                        url="jdbc:oracle:thin:@faislqureshi:1521:name";
                        String user="faisal";
                        String passw="arsenal";
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        connect=DriverManager.getConnection(url,user,passw);
                }
                catch(ClassNotFoundException cnfex){
                        cnfex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(SQLException sqlex){
                        sqlex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(Exception ex){
```

```
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }

}
catch(Exception e)
{
e.printStackTrace();
}

}

private void jbInit() throws Exception{

        this.getContentPane().setLayout(null);
        this.setSize(new Dimension(493, 350));
        cmdRegister.setText("Register");
        cmdRegister.addActionListener(this);
        cmdRegister.setBounds(new Rectangle(350, 330, 120, 35));
        cmdRegister.setToolTipText("Click Here to get confirmation.");
        txtBookingNo.setBounds(new Rectangle(195, 75, 115, 25));
        txtBookingNo.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtBookingNo.setForeground(SystemColor.desktop);
       txtFirstName.setBounds(new Rectangle(195, 105, 115, 25));
       txtFirstName.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtFirstName.setForeground(SystemColor.desktop);
       txtLastName.setBounds(new Rectangle(195, 135, 115, 25));
       txtLastName.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtLastName.setForeground(SystemColor.desktop);
       txtAddress.setBounds(new Rectangle(195, 165, 115, 25));
       txtAddress.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtAddress.setForeground(SystemColor.desktop);
       txtTelNo.setBounds(new Rectangle(195, 195, 115, 25));
       txtTelNo.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtTelNo.setForeground(SystemColor.desktop);
       txtFaxNo.setBounds(new Rectangle(195, 225, 115, 25));
       txtFaxNo.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtFaxNo.setForeground(SystemColor.desktop);
       txtEmail.setBounds(new Rectangle(195, 255, 115, 25));
       txtEmail.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtEmail.setForeground(SystemColor.desktop);
       txtPersonId.setBounds(new Rectangle(195, 285, 115, 25));
       txtPersonId.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtPersonId.setForeground(SystemColor.desktop);
       txtDOB.setBounds(new Rectangle(195, 315, 115, 25));
       txtDOB.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));
       txtDOB.setForeground(SystemColor.desktop);
       jLabel1.setText("Client Booking No:");
```

```
jLabel1.setBounds(new Rectangle(85, 75, 105, 25));
jLabel1.setBackground(Color.black);
jLabel1.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel1.setForeground(SystemColor.desktop);
jLabel2.setText("First Name:");
jLabel2.setBounds(new Rectangle(85, 105, 105, 25));
jLabel2.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel2.setForeground(SystemColor.desktop);
jLabel3.setText("Last Name:");
jLabel3.setBounds(new Rectangle(85, 135, 105, 25));
jLabel3.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel3.setForeground(SystemColor.desktop);
jLabel4.setText("Address:");
jLabel4.setBounds(new Rectangle(85, 165, 105, 25));
jLabel4.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel4.setForeground(SystemColor.desktop);
jLabel5.setText("TelNo:");
jLabel5.setBounds(new Rectangle(85, 195, 105, 25));
jLabel5.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel5.setForeground(SystemColor.desktop);
jLabel6.setText("FaxNo:");
jLabel6.setBounds(new Rectangle(85, 225, 105, 25));
jLabel6.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel6.setForeground(SystemColor.desktop);
jLabel7.setText("Email:");
jLabel7.setBounds(new Rectangle(85, 255, 105, 25));
jLabel7.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel7.setForeground(SystemColor.desktop);
jLabel8.setText("PersonId:");
jLabel8.setBounds(new Rectangle(85, 285, 105, 25));
jLabel8.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel8.setForeground(SystemColor.desktop);
jLabel9.setText("DOB:");
jLabel9.setBounds(new Rectangle(85, 315, 105, 25));
jLabel9.setBorder(BorderFactory.createLineBorder(Color.green, 1));
jLabel9.setForeground(SystemColor.desktop);
this.getContentPane().add(jLabel9, null);
this.getContentPane().add(jLabel8, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(txtBookingNo, null);
this.getContentPane().add(txtFirstName, null);
this.getContentPane().add(txtLastName, null);
this.getContentPane().add(txtAddress, null);
```

```
        this.getContentPane().add(txtTelNo, null);
        this.getContentPane().add(txtFaxNo, null);
        this.getContentPane().add(txtEmail, null);
        this.getContentPane().add(txtPersonId, null);
        this.getContentPane().add(txtDOB, null);
        this.getContentPane().add(cmdRegister, null);
}


public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdRegister){

        boolean flag=true;
        if(txtBookingNo.getText().length()==0){
                flag=false;
                JOptionPane.showMessageDialog(null,"Booking Number Required!");
        }
        if(flag){
                if(txtFirstName.getText().length()==0){
                flag=false;
                JOptionPane.showMessageDialog(null,"Enter First Name, then Press
Register.");
                }
        }

        if(flag){
                if(txtLastName.getText().length()==0){
                flag=false;
                JOptionPane.showMessageDialog(null,"Enter Last Name, then Press
Register.");
                }
        }

        if(flag){
                if(txtTelNo.getText().length()==0){
                flag=false;
                JOptionPane.showMessageDialog(null,"Enter Telephone No, then Press
Register.");
                }
        }
        if(flag){
                if(txtPersonId.getText().length()==0){
                flag=false;
                JOptionPane.showMessageDialog(null,"PersonId is Must Type, then
Proceed To Register.");
                }
        }
```

```
        if(flag){
        try{
        Statement statement=connect.createStatement();

        String query="INSERT INTO
Table_customer("+"BookingNo,FirstName,LastName,TelNo,PersonId"+")
values('"+txtBookingNo.getText()+"','"+txtFirstName.getText()+"','"+txtLastName.getT
ext()+"','"+txtTelNo.getText()+"','"+txtPersonId.getText()+"')";

        int result=statement.executeUpdate(query);

        if(result==1){
                JOptionPane.showMessageDialog(null,"Client Registered, Press OK to
Register Another.");
                txtBookingNo.setText("");
                txtFirstName.setText("");
                txtLastName.setText("");
                txtAddress.setText("");
                txtTelNo.setText("");
                txtFaxNo.setText("");
                txtEmail.setText("");
                txtPersonId.setText("");
                txtDOB.setText("");
        }
        else
                JOptionPane.showMessageDialog(null,"Could not Register into
DataBase");
        statement.close();
        }
        catch(SQLException sqlex){
                sqlex.printStackTrace();
                //output.append(sqlex.toString());
        }
        }
        }
}



public static void main(String args[]){

        ClientNew app=new ClientNew();

        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
```

```
    }
    );
    }}
```

*****Instay Info*****

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class InstayInfo extends JFrame implements ActionListener{

    private JMenuItem menuHelpAbout = new JMenuItem();
    private JMenu menuHelp = new JMenu();
    private JMenuItem menuNew = new JMenuItem();
    private JMenu menuFile = new JMenu();
    private JMenuBar menuBar = new JMenuBar();
    private JMenuItem menuChange = new JMenuItem();
    private JMenuItem menuDelete = new JMenuItem();
    private JMenuItem menuView = new JMenuItem();
    private JMenuItem menuExit = new JMenuItem();
    private JButton cmdNew = new JButton();
    private JButton cmdChange = new JButton();
    private JButton cmdDelete = new JButton();
    private JButton cmdView = new JButton();
    private JButton cmdExit = new JButton();


    public InstayInfo(){
        try{
            jbInit();
            setSize(500,400);
            setTitle("                    Instay Information");
            show();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }


    private void jbInit() throws Exception{

        this.setJMenuBar(menuBar);
        this.getContentPane().setLayout(null);
        this.setSize(new Dimension(447, 363));
        menuFile.setText("File");
        menuChange.setText("Change");
        menuChange.setForeground(Color.black);
        menuChange.addActionListener(this);
        menuDelete.setText("Delete");
```

```
menuDelete.setForeground(Color.black);
menuView.setText("View All");
menuView.setForeground(Color.black);
menuExit.setText("Exit");
menuExit.setForeground(Color.red);
cmdNew.setText("Register New Instay");
cmdNew.setBounds(new Rectangle(140, 35, 165, 35));
cmdNew.setBackground(Color.lightGray);
cmdNew.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdChange.setText("Change Instay Details");
cmdChange.setBounds(new Rectangle(140, 90, 165, 35));
cmdChange.setBackground(Color.lightGray);
cmdChange.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdDelete.setText("Delete Instay Details");
cmdDelete.setBounds(new Rectangle(140, 145, 165, 40));
cmdDelete.setBackground(Color.lightGray);
cmdDelete.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdView.setText("View All Instay Record");
cmdView.setBounds(new Rectangle(140, 205, 165, 40));
cmdView.setBackground(Color.lightGray);
cmdView.setBorder(BorderFactory.createLineBorder(Color.blue, 1));

cmdNew.addActionListener(this);
cmdChange.addActionListener(this);
cmdDelete.addActionListener(this);
cmdView.addActionListener(this);

menuDelete.addActionListener(this);
menuView.addActionListener(this);
menuExit.addActionListener(this);

menuNew.setText("New");
menuNew.setForeground(Color.black);
menuNew.addActionListener(this);
menuHelp.setText("Help");
menuHelpAbout.setText("About");
menuHelpAbout.setForeground(Color.black);
menuHelpAbout.addActionListener(this);
menuFile.add(menuNew);
menuFile.add(menuChange);
menuFile.add(menuDelete);
menuFile.add(menuView);
menuBar.add(menuFile);
menuHelp.add(menuHelpAbout);
menuBar.add(menuHelp);
this.getContentPane().add(cmdExit, null);
this.getContentPane().add(cmdView, null);
this.getContentPane().add(cmdDelete, null);
this.getContentPane().add(cmdChange, null);
```

```
            this.getContentPane().add(cmdNew, null);
    }

    public void actionPerformed(ActionEvent ae){

            if(ae.getSource()==cmdNew)
                    new InstayNew();

            if(ae.getSource()==menuNew)
                    new InstayNew();

            if(ae.getSource()==cmdChange)
                    new InstayChange();

            if(ae.getSource()==menuChange)
                    new InstayChange();

            if(ae.getSource()==cmdDelete)
                    new InstayDelete();

            if(ae.getSource()==menuDelete)
                    new InstayDelete();

            if(ae.getSource()==menuView)
                    new InstayView();

            if(ae.getSource()==cmdView)
                    new InstayView();

    }

    public static void main(String args[]){

            InstayInfo app=new InstayInfo();

            app.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent e){
                            System.exit(0);
                    }
            }
            );
    }
}
```

*****Main*****

```
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;

public class Main extends JFrame implements ActionListener{

        private JMenuItem menuHelpAbout = new JMenuItem();
        private JMenu menuHelp = new JMenu();
        private JMenuItem menuNew = new JMenuItem();
        private JMenu menuFile = new JMenu();
        private JMenuBar menuBar = new JMenuBar();

        private JMenuItem menuClientInfo = new JMenuItem();
        private JMenuItem menuInstayInfo = new JMenuItem();
        private JMenuItem menuTourInfo = new JMenuItem();
      private JMenuItem menuTourDaysInfo = new JMenuItem();
        private JMenuItem menuPriceInfo = new JMenuItem();
        private JMenuItem menuExit = new JMenuItem();

      private JButton cmdClientInfo = new JButton();
        private JButton cmdInstayInfo = new JButton();
        private JButton cmdTourInfo = new JButton();
      private JButton cmdTourDaysInfo = new JButton();
      private JButton cmdPriceInfo = new JButton();
        private JButton cmdExit = new JButton();


        public Main(){
                try{
                        jbInit();
                        setSize(500,450);
                        setTitle("                ***Welcome to a Turism Company
DataBase***");
                        show();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }

        private void jbInit() throws Exception{

                this.setJMenuBar(menuBar);
                this.getContentPane().setLayout(null);
                this.setSize(new Dimension(447, 363));
                menuFile.setText("File");
                menuClientInfo.setText("Client Info");
                menuClientInfo.setForeground(Color.black);
                menuInstayInfo.setText("Instay Info");
```

```
            menuInstayInfo.setForeground(Color.black);
            menuTourInfo.setText("Tour Info");
            menuTourInfo.setForeground(Color.black);
            menuTourDaysInfo.setText("Tour Days Info");
            menuTourDaysInfo.setForeground(Color.black);
        menuPriceInfo.setText("Price Info");
            menuExit.setForeground(Color.red);

        cmdClientInfo.setText("Client Information");
            cmdClientInfo.setBounds(new Rectangle(140, 35, 165, 35));
            cmdClientInfo.setBackground(Color.lightGray);
            cmdClientInfo.setBorder(BorderFactory.createLineBorder(Color.green,
1));
            cmdInstayInfo.setText("Instay Information");
            cmdInstayInfo.setBounds(new Rectangle(140, 90, 165, 35));
            cmdInstayInfo.setBackground(Color.lightGray);
            cmdInstayInfo.setBorder(BorderFactory.createLineBorder(Color.green,
1));
            cmdTourInfo.setText("Tour Informatiom");
            cmdTourInfo.setBounds(new Rectangle(140, 145, 165, 40));
            cmdTourInfo.setBackground(Color.lightGray);
            cmdTourInfo.setBorder(BorderFactory.createLineBorder(Color.green,
1));
            cmdTourDaysInfo.setText("Tour Days Informatiom");
            cmdTourDaysInfo.setBounds(new Rectangle(140, 205, 165, 40));
            cmdTourDaysInfo.setBackground(Color.lightGray);

      cmdTourDaysInfo.setBorder(BorderFactory.createLineBorder(Color.green, 1));
        cmdPriceInfo.setText("Price Information");
            cmdPriceInfo.setBounds(new Rectangle(140, 265, 165, 40));
            cmdPriceInfo.setBackground(Color.lightGray);
            cmdPriceInfo.setBorder(BorderFactory.createLineBorder(Color.green,
1));
        cmdExit.setText("Exit");
            cmdExit.setBounds(new Rectangle(360, 305, 70, 30));
            cmdExit.setBackground(Color.lightGray);
            cmdExit.setBorder(BorderFactory.createLineBorder(Color.cyan, 1));

        cmdClientInfo.addActionListener(this);
            cmdInstayInfo.addActionListener(this);
            cmdTourInfo.addActionListener(this);
            cmdTourDaysInfo.addActionListener(this);
        cmdPriceInfo.addActionListener(this);
            cmdExit.addActionListener(this);

        menuClientInfo.addActionListener(this);
        menuInstayInfo.addActionListener(this);
            menuTourInfo.addActionListener(this);
            menuTourDaysInfo.addActionListener(this);
```

```
        menuPriceInfo.addActionListener(this);
            menuExit.addActionListener(this);

        menuClientInfo.setText("Client Info");
            menuClientInfo.setForeground(Color.black);
            menuHelp.setText("Help");
            menuHelpAbout.setText("About");
            menuHelpAbout.setForeground(Color.black);
            menuHelpAbout.addActionListener(this);

        menuFile.add(menuClientInfo);
            menuFile.add(menuInstayInfo);
            menuFile.add(menuTourInfo);
            menuFile.add(menuTourDaysInfo);
        menuFile.add(menuPriceInfo);

        menuFile.add(menuExit);
            menuBar.add(menuFile);
            menuHelp.add(menuHelpAbout);
            menuBar.add(menuHelp);
            this.getContentPane().add(cmdExit, null);
            this.getContentPane().add(cmdPriceInfo, null);
            this.getContentPane().add(cmdTourDaysInfo, null);
        this.getContentPane().add(cmdTourInfo, null);
            this.getContentPane().add(cmdInstayInfo, null);
            this.getContentPane().add(cmdClientInfo, null);

}


  public void actionPerformed(ActionEvent ae){

          if(ae.getSource()==cmdClientInfo)
                  new ClientInfo();

          if(ae.getSource()==menuClientInfo)
                  new ClientInfo();

          if(ae.getSource()==cmdInstayInfo)
                  new InstayInfo();

          if(ae.getSource()==menuInstayInfo)
                  new InstayInfo();

          if(ae.getSource()==cmdTourInfo)
                  new TourInfo();

          if(ae.getSource()==menuTourInfo)
                  new TourInfo();
```

```
            if(ae.getSource()==cmdTourDaysInfo)
                    new TourDaysInfo();

            if(ae.getSource()==menuTourDaysInfo)
                    new TourDaysInfo();


          if(ae.getSource()==menuPriceInfo)
                    new PriceInfo();

            if(ae.getSource()==cmdPriceInfo)
                    new PriceInfo();


         if(ae.getSource()==cmdExit)
                    System.exit(0);

            if(ae.getSource()==menuExit)
                    System.exit(0);

       }


     public static void main(String args[]){

            Main app=new Main();

            app.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent e){
                            System.exit(0);
                    }
            }
            );
   }}
```

*****Price Change*****

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import java.awt.*;

public class PriceChange extends JFrame implements ActionListener{


private JTextField txtTPrice = new JTextField();
private JTextField txtBookingNo = new JTextField();
private JTextField txtInvoice = new JTextField();
private JTextField txtTax = new JTextField();
private JTextField txtSearch = new JTextField();
```

```
    this.getContentPane().add(txtInvoice, null);
    this.getContentPane().add(txtBookingNo, null);
    this.getContentPane().add(txtTPrice, null);

        txtTPrice.setEditable(false);
        txtBookingNo.setEditable(false);
        txtInvoice.setEditable(false);
        txtTax.setEditable(false);
}
   catch(Exception e)
   {
    e.printStackTrace();
   }

        setSize(400,300);
        setTitle("          Change Price Information");
        show();

        //setup database connection.
                try{
                        url="jdbc:oracle:thin:@FaisalqureshiWinXP:1521:db1";
                        String user="faisal";
                        String passw="arsenal";
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        connect=DriverManager.getConnection(url,user,passw);
                }
                catch(ClassNotFoundException cnfex){
                        cnfex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(SQLException sqlex){
                        sqlex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(Exception ex){
                        ex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }

}

public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdSearch){

        if(txtSearch.getText().length()==0){
```

```
            JOptionPane.showMessageDialog(null,"Enter Booking Number to
Search.");
        }

        if(txtSearch.getText().length()!=0){
        try{
        Statement statement=connect.createStatement();

        String query="Select * from Price_Table "+
                "where BookingNo= '"+
                txtSearch.getText()+"'";

        ResultSet rs=statement.executeQuery(query);
        //display rs
        try{
                rs.next();
                int recordNumber=rs.getInt(1);
                if(recordNumber!=0){
                        txtTPrice.setEditable(true);
                        txtInvoice.setEditable(true);
                        txtTax.setEditable(true);


                        txtTPrice,setText(String.valueOf(recordNumber));
                        txtBookingNo.setText(String.valueOf(recordNumber));
                        txtInvoice.setText(rs.getString(10));
                        txtTax.setText(String.valueOf(recordNumber));
                        }
                else
                JOptionPane.showMessageDialog(null,"No Record Found");
        }catch(SQLException sqlex){
                sqlex.printStackTrace();
                JOptionPane.showMessageDialog(null,"No Such Record Found.");
        }
        statement.close();
        }
        catch(SQLException sqlex){
                sqlex.printStackTrace();
                //output.append(sqlex.toString());
        }
        cmdOK.addActionListener(this);
        }
        }


        if(ae.getSource()==cmdOK){


        try{
```

```
        Statement statement1=connect.createStatement();

        String query1="Update Price_Table set "+
                "TPrice='"+txtTPrice.getText()+
                "',Invoice='"+txtInvoice.getText()+
                "',Tax='"+txtTax.getText()+
                "'where BookingNo="+txtBookingNo.getText();

                int result=statement1.executeUpdate(query1);
                if(result==1)
                        JOptionPane.showMessageDialog(null,"Changes Committed.");
                else
                        JOptionPane.showMessageDialog(null,"Could Not Update,
Please Retry Later!");
        statement1.close();
        }
        catch(SQLException sqlex){
                sqlex.printStackTrace();
        }
        }
}

public static void main(String args[]){

        PriceChange app=new PriceChange();

        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
        );
        }
}
```

*****Price Info*****

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PriceInfo extends JFrame implements ActionListener{

        private JMenuItem menuHelpAbout = new JMenuItem();
        private JMenu menuHelp = new JMenu();
        private JMenuItem menuNew = new JMenuItem();
        private JMenu menuFile = new JMenu();
        private JMenuBar menuBar = new JMenuBar();
        private JMenuItem menuChange = new JMenuItem();
        private JMenuItem menuDelete = new JMenuItem();
        private JMenuItem menuView = new JMenuItem();
```

```
private JMenuItem menuExit = new JMenuItem();
private JButton cmdNew = new JButton();
private JButton cmdChange = new JButton();
private JButton cmdDelete = new JButton();
private JButton cmdView = new JButton();


public PriceInfo(){
        try{
                jbInit();
                setSize(500,400);
                setTitle("                              Price Information");
                show();
        }
        catch(Exception e){
                e.printStackTrace();
        }
}

private void jbInit() throws Exception{

        this.setJMenuBar(menuBar);
        this.getContentPane().setLayout(null);
        this.setSize(new Dimension(447, 363));
        menuFile.setText("File");
        menuChange.setText("Change");
        menuChange.setForeground(Color.black);
        menuChange.addActionListener(this);
        menuDelete.setText("Delete");
        menuDelete.setForeground(Color.black);
        menuView.setText("View All");
        menuView.setForeground(Color.black);
        menuExit.setText("Exit");
        menuExit.setForeground(Color.red);
        cmdNew.setText("New Price");
        cmdNew.setBounds(new Rectangle(140, 35, 165, 35));
        cmdNew.setBackground(Color.lightGray);
        cmdNew.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
        cmdChange.setText("Change Price");
        cmdChange.setBounds(new Rectangle(140, 90, 165, 35));
        cmdChange.setBackground(Color.lightGray);
        cmdChange.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
        cmdDelete.setText("Delete Price");
        cmdDelete.setBounds(new Rectangle(140, 145, 165, 40));
        cmdDelete.setBackground(Color.lightGray);
        cmdDelete.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
        cmdView.setText("Show All Prices");
```

```
        cmdView.setBounds(new Rectangle(140, 205, 165, 40));
        cmdView.setBackground(Color.lightGray);
        cmdView.setBorder(BorderFactory.createLineBorder(Color.blue, 1));


        cmdNew.addActionListener(this);
        cmdChange.addActionListener(this);
        cmdDelete.addActionListener(this);
        cmdView.addActionListener(this);

        menuDelete.addActionListener(this);
        menuView.addActionListener(this);
        menuExit.addActionListener(this);

        menuNew.setText("New");
        menuNew.setForeground(Color.black);
        menuNew.addActionListener(this);
        menuHelp.setText("Help");
        menuHelpAbout.setText("About");
        menuHelpAbout.setForeground(Color.black);
        menuHelpAbout.addActionListener(this);
        menuFile.add(menuNew);
        menuFile.add(menuChange);
        menuFile.add(menuDelete);
        menuFile.add(menuView);
        menuFile.add(menuExit);
        menuBar.add(menuFile);
        menuHelp.add(menuHelpAbout);
        menuBar.add(menuHelp);

        this.getContentPane().add(cmdView, null);
        this.getContentPane().add(cmdDelete, null);
        this.getContentPane().add(cmdChange, null);
        this.getContentPane().add(cmdNew, null);
}

public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdNew)
                new PriceNew();

        if(ae.getSource()==menuNew)
                new PriceNew();

        if(ae.getSource()==cmdChange)
                new PriceChange();

        if(ae.getSource()==menuChange)
```

```
                    new PriceChange();

                if(ae.getSource()==cmdDelete)
                    new PriceDelete();

                if(ae.getSource()==menuDelete)
                    new PriceDelete();

                if(ae.getSource()==menuView)
                    new PriceView();

                if(ae.getSource()==cmdView)
                    new PriceView();


        }

        public static void main(String args[]){

                PriceInfo app=new PriceInfo();

                app.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent e){
                                System.exit(0);
                        }
                }
                );
        }
}
```

*****Tour Days Change*****

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import java.awt.*;

public class TourDaysChange extends JFrame implements ActionListener{


private JTextField txtPackageId = new JTextField();
private JTextField txtOperational = new JTextField();
private JTextField txtSearch = new JTextField();

private JButton cmdSearch = new JButton();
private JLabel jLabel1 = new JLabel();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private JButton cmdOK = new JButton();
```

```
Connection connect;
String url;

 public TourDaysChange()
 {
  try
  {

  this.getContentPane().setLayout(null);
  this.setSize(new Dimension(400, 250));
  txtPackageId.setBounds(new Rectangle(160, 90, 140, 25));
  txtOperational.setBounds(new Rectangle(160, 125, 140, 25));
  txtSearch.setBounds(new Rectangle(120, 20, 120, 25));
  cmdSearch.setText("Search");
  cmdSearch.setBounds(new Rectangle(260, 20, 90, 25));
  cmdSearch.addActionListener(this);
  jLabel1.setText("Package Id :");
  jLabel1.setBounds(new Rectangle(20, 20, 100, 25));
  jLabel2.setText("Package No:");
  jLabel2.setBounds(new Rectangle(80, 90, 100, 25));
  jLabel3.setText(" Days Operated:");
  jLabel3.setBounds(new Rectangle(60, 125, 105, 20));
  cmdOK.setText("OK");
  cmdOK.setBounds(new Rectangle(250, 170, 70, 25));
  this.getContentPane().add(cmdOK, null);
  this.getContentPane().add(jLabel3, null);
  this.getContentPane().add(jLabel2, null);
  this.getContentPane().add(jLabel1, null);
  this.getContentPane().add(cmdSearch, null);
  this.getContentPane().add(txtSearch, null);

  this.getContentPane().add(txtOperational, null);
  this.getContentPane().add(txtPackageId, null);

       txtPackageId.setEditable(false);
       txtOperational.setEditable(false);
}
  catch(Exception e)
  {
   e.printStackTrace();
  }

       setSize(400,250);
       setTitle("          Change Tour Days Information");
       show();

       //setup database connection.
            try{
```

```
                url="jdbc:oracle:thin:@FaisalqureshiWinXP:1521:db1";
                String user="faisal";
                String passw="arsenal";
                Class.forName("oracle.jdbc.driver.OracleDriver");
                connect=DriverManager.getConnection(url,user,passw);
            }
            catch(ClassNotFoundException cnfex){
                cnfex.printStackTrace();
                JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
            }
            catch(SQLException sqlex){
                sqlex.printStackTrace();
                JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
            }
            catch(Exception ex){
                ex.printStackTrace();
                JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
            }

}

public void actionPerformed(ActionEvent ae){

    if(ae.getSource()==cmdSearch){

    if(txtSearch.getText().length()==0){
        JOptionPane.showMessageDialog(null,"Enter Tour  Id Number to
Search.");
    }

    if(txtSearch.getText().length()!=0){
    try{
    Statement statement=connect.createStatement();

    String query="Select * from Tour_Days "+
        "where PackageId= '"+
        txtSearch.getText()+"'";

    ResultSet rs=statement.executeQuery(query);
    //display rs
    try{
        rs.next();
        int recordNumber=rs.getInt(1);
        if(recordNumber!=0){
            txtPackageId.setEditable(true);
        txtOperational.setEditable(true);
```

```
                txtPackageId.setText(String.valueOf(recordNumber));
                txtOperational.setText(rs.getString(10));
            }
            else
            JOptionPane.showMessageDialog(null,"No Record Found");
        }catch(SQLException sqlex){
            sqlex.printStackTrace();
            JOptionPane.showMessageDialog(null,"No Such Record Found.");
        }
        statement.close();
        }
        catch(SQLException sqlex){
            sqlex.printStackTrace();
            //output.append(sqlex.toString());
        }
        cmdOK.addActionListener(this);
        }
        }

        if(ae.getSource()==cmdOK){


        try{
        Statement statement1=connect.createStatement();

        String query1="Update Tour_Days set "+
            "Operational='"+txtOperational.getText()+
            "'where PackageId="+txtPackageId.getText();

            int result=statement1.executeUpdate(query1);
            if(result==1)
                JOptionPane.showMessageDialog(null,"Changes Committed.");
            else
                JOptionPane.showMessageDialog(null,"Could Not Update,
Please Retry Later!");
        statement1.close();
        }
        catch(SQLException sqlex){
            sqlex.printStackTrace();
        }
        }
}

public static void main(String args[]){

        TourDaysChange app=new TourDaysChange();
```

```
        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
        );
        }
}
```

Tour Days Info

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TourDaysInfo extends JFrame implements ActionListener{

        private JMenuItem menuHelpAbout = new JMenuItem();
        private JMenu menuHelp = new JMenu();
        private JMenuItem menuNew = new JMenuItem();
        private JMenu menuFile = new JMenu();
        private JMenuBar menuBar = new JMenuBar();
        private JMenuItem menuChange = new JMenuItem();
        private JMenuItem menuDelete = new JMenuItem();
        private JMenuItem menuView = new JMenuItem();
        private JMenuItem menuExit = new JMenuItem();
        private JButton cmdNew = new JButton();
        private JButton cmdChange = new JButton();
        private JButton cmdDelete = new JButton();
        private JButton cmdView = new JButton();


        public TourDaysInfo(){
                try{
                        jbInit();
                        setSize(500,400);
                        setTitle("                    Tour Days Information");
                        show();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }


        private void jbInit() throws Exception{

                this.setJMenuBar(menuBar);
```

```
this.getContentPane().setLayout(null);
this.setSize(new Dimension(447, 363));
menuFile.setText("File");
menuChange.setText("Change");
menuChange.setForeground(Color.black);
menuChange.addActionListener(this);
menuDelete.setText("Delete");
menuDelete.setForeground(Color.black);
menuView.setText("View All");
menuView.setForeground(Color.black);
menuExit.setText("Exit");
menuExit.setForeground(Color.red);
cmdNew.setText("New Tour Day");
cmdNew.setBounds(new Rectangle(140, 35, 165, 35));
cmdNew.setBackground(Color.lightGray);
cmdNew.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdChange.setText("Change Tour Days");
cmdChange.setBounds(new Rectangle(140, 90, 165, 35));
cmdChange.setBackground(Color.lightGray);
cmdChange.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdDelete.setText("Remove Tour Days");
cmdDelete.setBounds(new Rectangle(140, 145, 165, 40));
cmdDelete.setBackground(Color.lightGray);
cmdDelete.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
cmdView.setText("Show All Tour's Days");
cmdView.setBounds(new Rectangle(140, 205, 165, 40));
cmdView.setBackground(Color.lightGray);
cmdView.setBorder(BorderFactory.createLineBorder(Color.blue, 1));


cmdNew.addActionListener(this);
cmdChange.addActionListener(this);
cmdDelete.addActionListener(this);
cmdView.addActionListener(this);

menuDelete.addActionListener(this);
menuView.addActionListener(this);
menuExit.addActionListener(this);

menuNew.setText("New");
menuNew.setForeground(Color.black);
menuNew.addActionListener(this);
menuHelp.setText("Help");
menuHelpAbout.setText("About");
menuHelpAbout.setForeground(Color.black);
menuHelpAbout.addActionListener(this);
menuFile.add(menuNew);
menuFile.add(menuChange);
menuFile.add(menuDelete);
```

```
        menuFile.add(menuView);
        menuFile.add(menuExit);
        menuBar.add(menuFile);
        menuHelp.add(menuHelpAbout);
        menuBar.add(menuHelp);

        this.getContentPane().add(cmdView, null);
        this.getContentPane().add(cmdDelete, null);
        this.getContentPane().add(cmdChange, null);
        this.getContentPane().add(cmdNew, null);
}

public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdNew)
                new TourDaysNew();

        if(ae.getSource()==menuNew)
                new TourDaysNew();

        if(ae.getSource()==cmdChange)
                new TourDaysChange();

        if(ae.getSource()==menuChange)
                new TourDaysChange();

        if(ae.getSource()==cmdDelete)
                new TourDaysDelete();

        if(ae.getSource()==menuDelete)
                new TourDaysDelete();

        if(ae.getSource()==menuView)
                new TourDaysView();

        if(ae.getSource()==cmdView)
                new TourDaysView();

}

public static void main(String args[]){

        TourDaysInfo app=new TourDaysInfo();

        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
```

```
                }
            }
            );
        }
}
```

Tour Change

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import java.awt.*;

public class TourChange extends JFrame implements ActionListener{


private JTextField txtPackageId = new JTextField();
private JTextField txtRegion = new JTextField();
private JTextField txtFrom = new JTextField();
private JTextField txtTo1 = new JTextField();
private JTextField txtTo2 = new JTextField();
private JTextField txtTo3 = new JTextField();
private JTextField txtTo4 = new JTextField();
private JTextField txtTo5 = new JTextField();
private JTextField txtFinal = new JTextField();
private JTextField txtSearch = new JTextField();

private JButton cmdSearch = new JButton();
private JLabel jLabel1 = new JLabel();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private JLabel jLabel4 = new JLabel();
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private JLabel jLabel7 = new JLabel();
private JLabel jLabel8 = new JLabel();
private JLabel jLabel9 = new JLabel();
private JLabel jLabel10 = new JLabel();
private JButton cmdOK = new JButton();

Connection connect;
String url;

 public TourChange()
 {
  try
  {

  this.getContentPane().setLayout(null);
  this.setSize(new Dimension(550, 450));
  txtPackageId.setBounds(new Rectangle(190, 90, 140, 25));
```

```
txtRegion.setBounds(new Rectangle(190, 125, 140, 25));
txtFrom.setBounds(new Rectangle(190, 160, 140, 25));
txtTo1.setBounds(new Rectangle(190, 195, 140, 25));
txtTo2.setBounds(new Rectangle(190, 230, 140, 25));
txtTo3.setBounds(new Rectangle(190, 265, 140, 25));
txtTo4.setBounds(new Rectangle(190, 300, 140, 25));
txtTo5.setBounds(new Rectangle(190, 335, 140, 25));
txtFinal.setBounds(new Rectangle(190, 370, 140, 25));
txtSearch.setBounds(new Rectangle(200, 20, 120, 25));
cmdSearch.setText("Search");
cmdSearch.setBounds(new Rectangle(330, 20, 90, 25));
cmdSearch.addActionListener(this);
jLabel1.setText("Tour Id :");
jLabel1.setBounds(new Rectangle(150, 20, 100, 25));
jLabel2.setText("Tour Id:");
jLabel2.setBounds(new Rectangle(135, 90, 100, 25));
jLabel3.setText("Region:");
jLabel3.setBounds(new Rectangle(135, 125, 105, 20));
jLabel4.setText("From:");
jLabel4.setBounds(new Rectangle(135, 155, 95, 25));
jLabel5.setText("To1:");
jLabel5.setBounds(new Rectangle(145, 195, 75, 20));
jLabel6.setText("To2:");
jLabel6.setBounds(new Rectangle(145, 230, 85, 25));
jLabel7.setText("To3:");
jLabel7.setBounds(new Rectangle(145, 266, 85, 25));
jLabel8.setText("To4:");
jLabel8.setBounds(new Rectangle(145, 298, 85, 25));
jLabel9.setText("To5:");
jLabel9.setBounds(new Rectangle(145, 335, 85, 25));
jLabel10.setText("Final:");
jLabel10.setBounds(new Rectangle(145, 370, 85, 25));
cmdOK.setText("OK");
cmdOK.setBounds(new Rectangle(380, 285, 70, 25));
this.getContentPane().add(cmdOK, null);
this.getContentPane().add(jLabel10, null);
this.getContentPane().add(jLabel9, null);
this.getContentPane().add(jLabel8, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(cmdSearch, null);
this.getContentPane().add(txtSearch, null);

this.getContentPane().add(txtFinal, null);
```

```
        this.getContentPane().add(txtTo5, null);
        this.getContentPane().add(txtTo4, null);
        this.getContentPane().add(txtTo3, null);
        this.getContentPane().add(txtTo2, null);
        this.getContentPane().add(txtTo1, null);
        this.getContentPane().add(txtFrom, null);
        this.getContentPane().add(txtRegion, null);
        this.getContentPane().add(txtPackageId, null);

            txtPackageId.setEditable(false);
            txtRegion.setEditable(false);
            txtFrom.setEditable(false);
            txtTo1.setEditable(false);
            txtTo2.setEditable(false);
            txtTo3.setEditable(false);
            txtTo4.setEditable(false);
            txtTo5.setEditable(false);
            txtFinal.setEditable(false);
    }
       catch(Exception e)
       {
        e.printStackTrace();
       }

            setSize(550,450);
            setTitle("                              Tour Change Information");
            show();

            //setup database connection.
                    try{
                            url="jdbc:oracle:thin:@FaisalqureshiWinXP:1521:db1";
                            String user="faisal";
                            String passw="arsenal";
                            Class.forName("oracle.jdbc.driver.OracleDriver");
                            connect=DriverManager.getConnection(url,user,passw);
                    }
                    catch(ClassNotFoundException cnfex){
                            cnfex.printStackTrace();
                            JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                    }
                    catch(SQLException sqlex){
                            sqlex.printStackTrace();
                            JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                    }
                    catch(Exception ex){
                            ex.printStackTrace();
```

```
                    JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }

}

public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==cmdSearch){

        if(txtSearch.getText().length()==0){
                JOptionPane.showMessageDialog(null,"Enter Tour Id Number to
Search.");
        }

        if(txtSearch.getText().length()!=0){
        try{
        Statement statement=connect.createStatement();

        String query="Select * from Table_Tours "+
                "where PackageId= '"+
                txtSearch.getText()+"'";

        ResultSet rs=statement.executeQuery(query);
        //display rs
        try{
                rs.next();
                int recordNumber=rs.getInt(1);
                if(recordNumber!=0){
                        txtRegion.setEditable(true);
                        txtFrom.setEditable(true);
                        txtTo1.setEditable(true);
                        txtTo2.setEditable(true);
                        txtTo3.setEditable(true);
                        txtTo4.setEditable(true);
                        txtTo5.setEditable(true);
                        txtFinal.setEditable(true);

                        txtPackageId.setText(String.valueOf(recordNumber));
                        txtRegion.setText(rs.getString(10));
                        txtFrom.setText(rs.getString(10));
                        txtTo1.setText(rs.getString(15));
                        txtTo2.setText(String.valueOf(recordNumber));
                        txtTo3.setText(String.valueOf(recordNumber));
                        txtTo4.setText(rs.getString(10));
                        txtTo5.setText(String.valueOf(recordNumber));
                        txtFinal.setText(rs.getString(7));
                }
                else
```

```
                JOptionPane.showMessageDialog(null,"No Record Found");
            }catch(SQLException sqlex){
                sqlex.printStackTrace();
                JOptionPane.showMessageDialog(null,"No Such Record Found.");
            }
            statement.close();
            }
            catch(SQLException sqlex){
                sqlex.printStackTrace();
                //output.append(sqlex.toString());
            }
            cmdOK.addActionListener(this);
            }
            }


            if(ae.getSource()==cmdOK){

            try{
            Statement statement1=connect.createStatement();

            String query1="Update Table_Tour set "+
                    "Region='"+txtRegion.getText()+
                    "',From='"+txtFrom.getText()+
                    "',To1='"+txtTo1.getText()+
                    "',To2='"+txtTo2.getText()+
                    "',To3='"+txtTo3.getText()+
                    "',To4='"+txtTo4.getText()+
                    "',To5='"+txtTo5.getText()+
                    "',Final='"+txtFinal.getText()+
                    "'where PackageId='"+txtPackageId.getText();

                    int result=statement1.executeUpdate(query1);
                    if(result==1)
                            JOptionPane.showMessageDialog(null,"Changes Committed.");
                    else
                            JOptionPane.showMessageDialog(null,"Could Not Update,
Please Retry Later!");
            statement1.close();
            }
            catch(SQLException sqlex){
                sqlex.printStackTrace();
            }
            }
}

public static void main(String args[]){
```

```
        TourChange app=new TourChange();

        app.addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
        );
        }
}
```

*****Tour Delete*****

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
import java.awt.*;

public class TourDelete extends JFrame implements ActionListener{

private JTextField txtPackageId = new JTextField();
private JTextField txtRegion = new JTextField();
private JTextField txtFrom = new JTextField();
private JTextField txtTo1 = new JTextField();
private JTextField txtTo2 = new JTextField();
private JTextField txtTo3 = new JTextField();
private JTextField txtTo4 = new JTextField();
private JTextField txtTo5 = new JTextField();
private JTextField txtFinal = new JTextField();
private JTextField txtSearch = new JTextField();

private JButton cmdSearch = new JButton();
private JLabel jLabel1 = new JLabel();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private JLabel jLabel4 = new JLabel();
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private JLabel jLabel7 = new JLabel();
private JLabel jLabel8 = new JLabel();
private JLabel jLabel9 = new JLabel();
private JLabel jLabel10 = new JLabel();
private JButton cmdOK = new JButton();

Connection connect;
String url;

 public TourDelete()
 {
  try
  {
```

```
this.getContentPane().setLayout(null);
this.setSize(new Dimension(550, 450));
txtPackageId.setBounds(new Rectangle(190, 90, 140, 25));
txtRegion.setBounds(new Rectangle(190, 125, 140, 25));
txtFrom.setBounds(new Rectangle(190, 160, 140, 25));
txtTo1.setBounds(new Rectangle(190, 195, 140, 25));
txtTo2.setBounds(new Rectangle(190, 230, 140, 25));
txtTo3.setBounds(new Rectangle(190, 265, 140, 25));
txtTo4.setBounds(new Rectangle(190, 300, 140, 25));
txtTo5.setBounds(new Rectangle(190, 335, 140, 25));
txtFinal.setBounds(new Rectangle(190, 370, 140, 25));
txtSearch.setBounds(new Rectangle(175, 20, 100, 25));
cmdSearch.setText("Search");
cmdSearch.setBounds(new Rectangle(300, 20, 90, 25));
cmdSearch.addActionListener(this);
jLabel1.setText("Package ID:");
jLabel1.setBounds(new Rectangle(80, 20, 120, 25));
jLabel2.setText("Package Id:");
jLabel2.setBounds(new Rectangle(115, 90, 100, 25));
jLabel3.setText("Region:");
jLabel3.setBounds(new Rectangle(135, 125, 105, 20));
jLabel4.setText("From:");
jLabel4.setBounds(new Rectangle(145, 155, 95, 25));
jLabel5.setText("To1:");
jLabel5.setBounds(new Rectangle(150, 195, 75, 20));
jLabel6.setText("To2:");
jLabel6.setBounds(new Rectangle(150, 230, 85, 25));
jLabel7.setText("To3:");
jLabel7.setBounds(new Rectangle(150, 265, 85, 25));
jLabel8.setText("To4:");
jLabel8.setBounds(new Rectangle(150, 300, 85, 25));
jLabel9.setText("To5:");
jLabel9.setBounds(new Rectangle(150, 335, 85, 25));
jLabel10.setText("Final:");
jLabel10.setBounds(new Rectangle(145, 385, 85, 25));
cmdOK.setText("OK");
cmdOK.setBounds(new Rectangle(350, 285, 70, 25));
this.getContentPane().add(cmdOK, null);
this.getContentPane().add(jLabel10, null);
this.getContentPane().add(jLabel9, null);
this.getContentPane().add(jLabel8, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel1, null);
```

```
    this.getContentPane().add(cmdSearch, null);
    this.getContentPane().add(txtSearch, null);
    this.getContentPane().add(txtFinal, null);
    this.getContentPane().add(txtTo5, null);
    this.getContentPane().add(txtTo4, null);
    this.getContentPane().add(txtTo3, null);
    this.getContentPane().add(txtTo2, null);
    this.getContentPane().add(txtTo1, null);
    this.getContentPane().add(txtFrom, null);
    this.getContentPane().add(txtRegion, null);
    this.getContentPane().add(txtPackageId, null);

        txtPackageId.setEditable(false);
        txtRegion.setEditable(false);
        txtFrom.setEditable(false);
        txtTo1.setEditable(false);
        txtTo2.setEditable(false);
txtTo3.setEditable(false);
txtTo4.setEditable(false);
txtTo5.setEditable(false);
txtFinal.setEditable(false);

}
   catch(Exception e)
   {
    e.printStackTrace();
   }

        setSize(500,400);
        setTitle("                    Tour Delete Option");
        show();

        //setup database connection.
                try{
                        url="jdbc:oracle:thin:@faisalqureshiWinXP:1521:db1";
                        String user="Faisal";
                        String passw="arsenal";
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        connect=DriverManager.getConnection(url,user,passw);
                }
                catch(ClassNotFoundException cnfex){
                        cnfex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
                }
                catch(SQLException sqlex){
                        sqlex.printStackTrace();
                        JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
```

```
            }
            catch(Exception ex){
                  ex.printStackTrace();
                  JOptionPane.showMessageDialog(null,"Could not connect to the
Database");
            }

}

public void actionPerformed(ActionEvent ae){

      if(ae.getSource()==cmdSearch){

      if(txtSearch.getText().length()==0){
            JOptionPane.showMessageDialog(null,"Enter Package Id to Search.");
      }
      if(txtSearch.getText().length()!=0){
      try{
      Statement statement=connect.createStatement();

      String query="Select * from Table_Tour "+
            "where packageId= '"+
            txtSearch.getText()+"'";

      ResultSet rs=statement.executeQuery(query);
      //display rs
      try{
            rs.next();
            int recordNumber=rs.getInt(1);
            if(recordNumber!=0){

                  txtPackageId.setText(String.valueOf(recordNumber));
                  txtRegion.setText(rs.getString(2));
                  txtFrom.setText(rs.getString(3));
                  txtTo1.setText(rs.getString(4));
                  txtTo2.setText(String.valueOf(recordNumber));
                  txtTo3.setText(String.valueOf(recordNumber));
                  txtTo4.setText(rs.getString(10));
                  txtTo5.setText(rs.getString(5));
                  txtFinal.setText(rs.getString(7));
            }
            else
            JOptionPane.showMessageDialog(null,"No Record Found");
      }catch(SQLException sqlex){
            sqlex.printStackTrace();
            JOptionPane.showMessageDialog(null,"No Such Record Found.");
      }
      statement.close();
```

Conclusion

The field database has evolved from ordinary files to a complete Database Management System. There are over a thousand RDBMS software's now available in the world . Some has advantages and other lack behind in many respect. Amongst these is Oracle which has tremendous amount of capability in handling Database Applications, from simple desktop to mainframes .

Oracle with Java has a grate amount of potential as Java is new to this field new developments are adding up day by day . One such development is JDBC which enables Java to connect to the Database using the most securest way of accessing a database. But still lacks behind in compilation which is time consuming .

The project fulfills the request of a database system for a Tourism Company. In this system user can add, delete, view and change the database as of company's requirements.

References

[1] CJ Date, An Introduction to Database Systems 2nd ed., 1997

[2] John Carter, Database Design and Programming Second Edition, 2003

[3] Ramon Mata Toledo, Pauline Cushman, Fundamentals of SQL Programming Mc Graw Hill, 2000

[4] Deitel And Deitel, Java How to Program Third Edition, 2000

[5] Quin Charatan, Aaron Kans, Java in Two Semesters, Mc Graw Hill, 2002

[6] Herbert Shildt, Java 2 The Complete Reference, Mc Graw Hill Fourth Edition, 2000

[7] Java™ 2 SDK, Standard Edition Documentation Version 1.3, 2003

[8] John Papageorge Getting Started With JDBC

[9] Sun Micro Systems from the World Wide Web *http://www.sun.com*

[10] Oracle Corporation from the World Wide Web *http://www.oracle.com*

[11] Ministry of Tourism Turkey from the World Wide Web *http://www.go-turkey.gov.tr*

[12] Anadolu Tourism Associate from the World Wide Web *http://www.anadoluturizm.net.tr*

[13] Yahoo web pages from the World Wide Web *http://www.yahoo.com*

[14] Google Search Engine from the World Wide Web *http://www.google.com*