



Java Programming

9-1

JDBC Introduction



Objectives

This lesson covers the following topics:

- Describe the JDBC
- Introduce Oracle JDBC Driver
- Outline the steps in JDBC programming
- Describe the JDBC Statement



What is JDBC?

- Java provides the JDBC API for developing database applications that works with any relational database systems.
- The JDBC API has two main parts:
 - Java application developer Interface for java programming language developers.
 - JDBC driver developers implementation interface
- This lesson introduces the application developers interface.

JDBC Packages

- The JDBC API is comprised of two packages:
 - `java.sql`
 - `javax.sql`
- You automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

JDBC Driver Types

JDBC Drivers are categorized in four types:

- Type 1 Driver: JDBC-ODBC bridge - Allows ODBC drivers to be used as JDBC drivers.
- Type 2 Driver: Native-API Driver - Built on top of a native database client library.
- Type 3 Driver: Network-Protocol Driver - Pure java client to communicate with a middleware server seated between the client and database.
- Type 4 Driver: Native-Protocol Driver - pure Java to implement the network protocol for a specific data source.

Oracle JDBC Drivers

- In addition to supporting the standard JDBC application programming interfaces (APIs), Oracle drivers have extensions to support Oracle-specific data types and to enhance performance.
- Oracle provides the following JDBC Drivers:
 - JDBC Thin driver - Type 4 Driver. It is a pure Java driver used on the client-side, without an Oracle client installation. It can be used with both applets and applications. The JDBC Thin driver enables a direct connection to the database.
 - Oracle Call Interface (OCI) driver - Type 2 Driver. It is used on the client-side with an Oracle client installation.

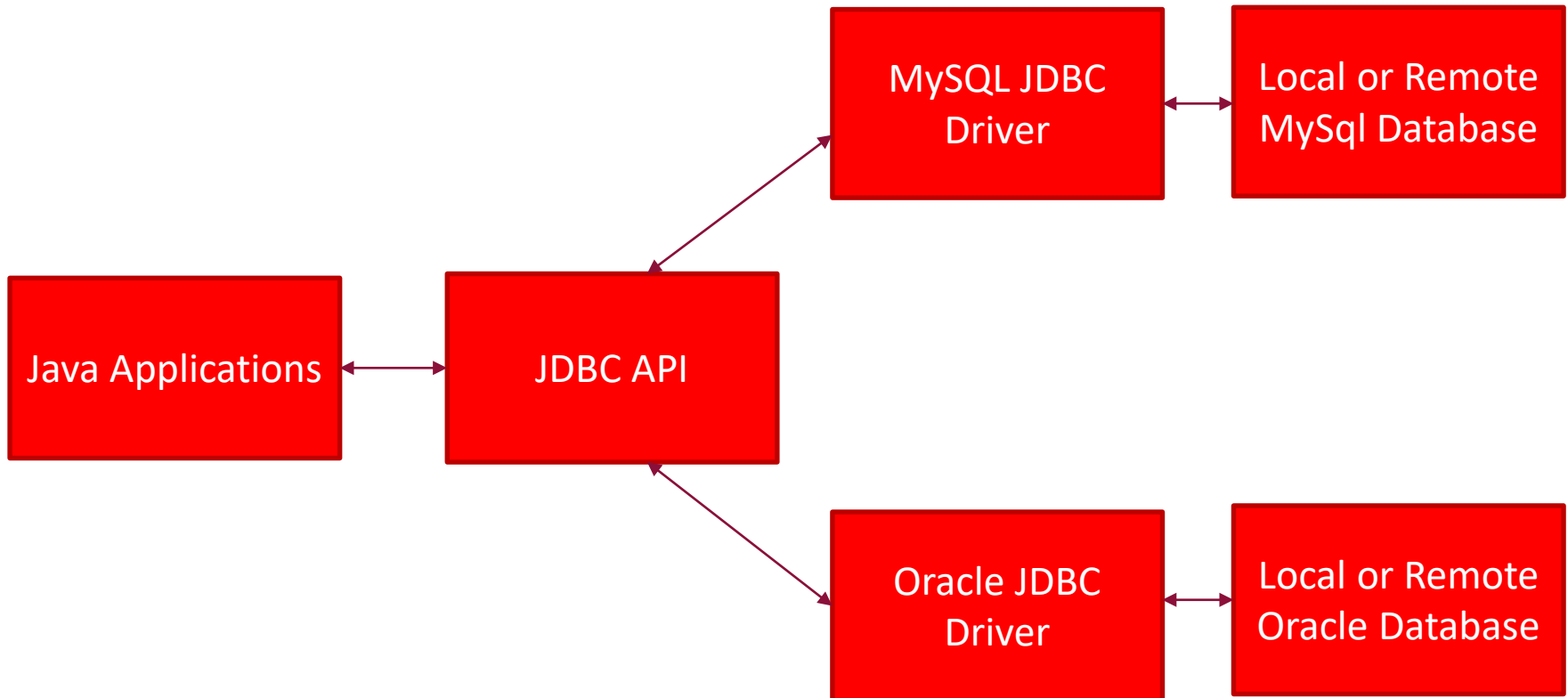
Oracle JDBC Thin Driver

- The JDBC Thin driver is a pure Java, Type IV driver that can be used in applications and applets.
- It is platform-independent and does not require any additional Oracle software on the client-side.
- The JDBC Thin driver communicates with the server using SQL*Net to access Oracle Database.

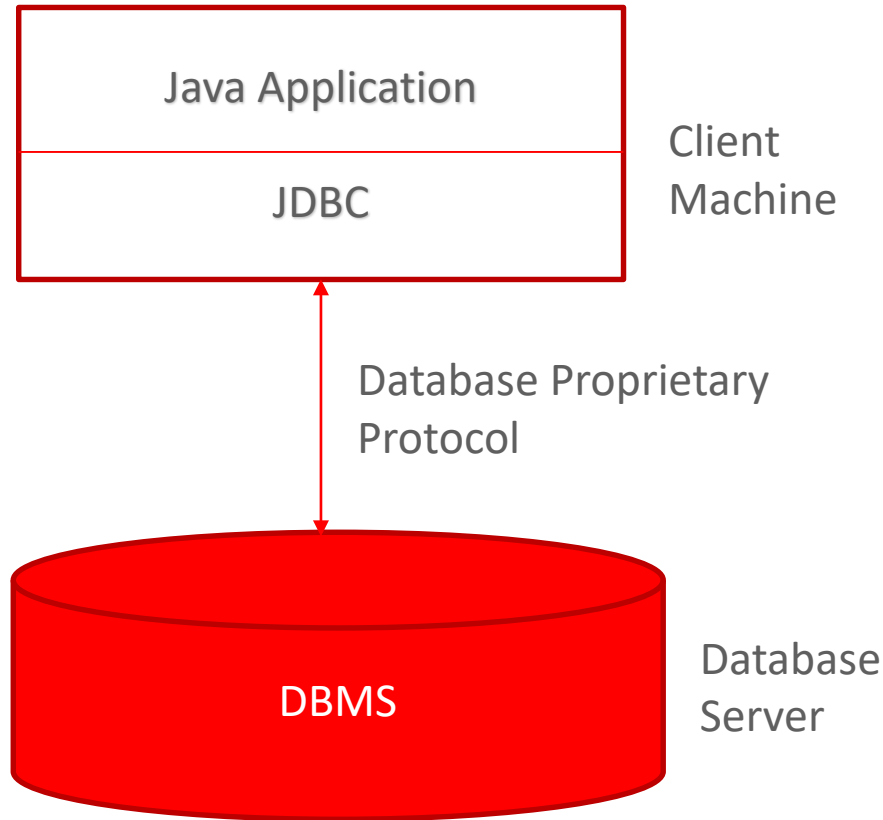
Oracle JDBC Thin Driver

- The JDBC Thin driver allows a direct connection to the database by providing an implementation of SQL*Net on top of Java sockets.
- The driver supports the TCP/IP protocol and requires a TNS listener on the TCP/IP sockets on the database server.

Java Application and JDBC Driver

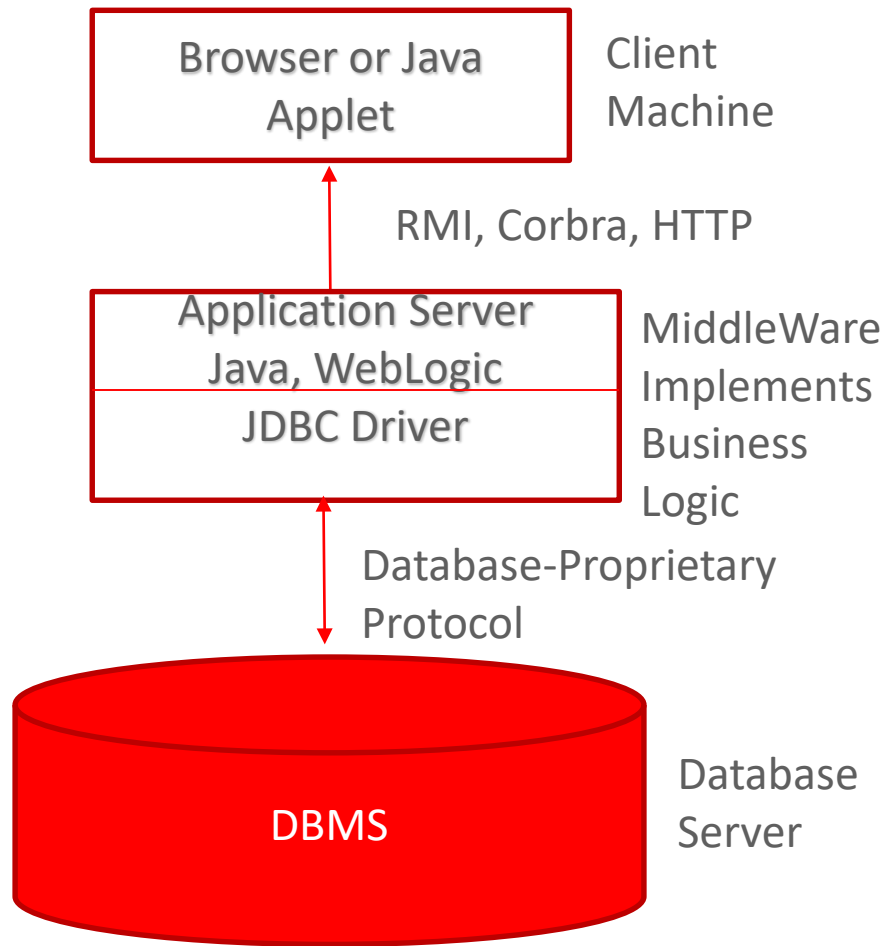


JDBC Architecture (Two-Tier)



- Java application talks directly to the data source.
- Commands are sent to the database and the results of those statements are sent back to the user.

JDBC Architecture (Three-Tier)



Requests are sent to the middle tier server, which then sends commands to the data source.

Developing a Database Application Using JDBC

JDBC helps you to write Java applications that manage these three programming activities:

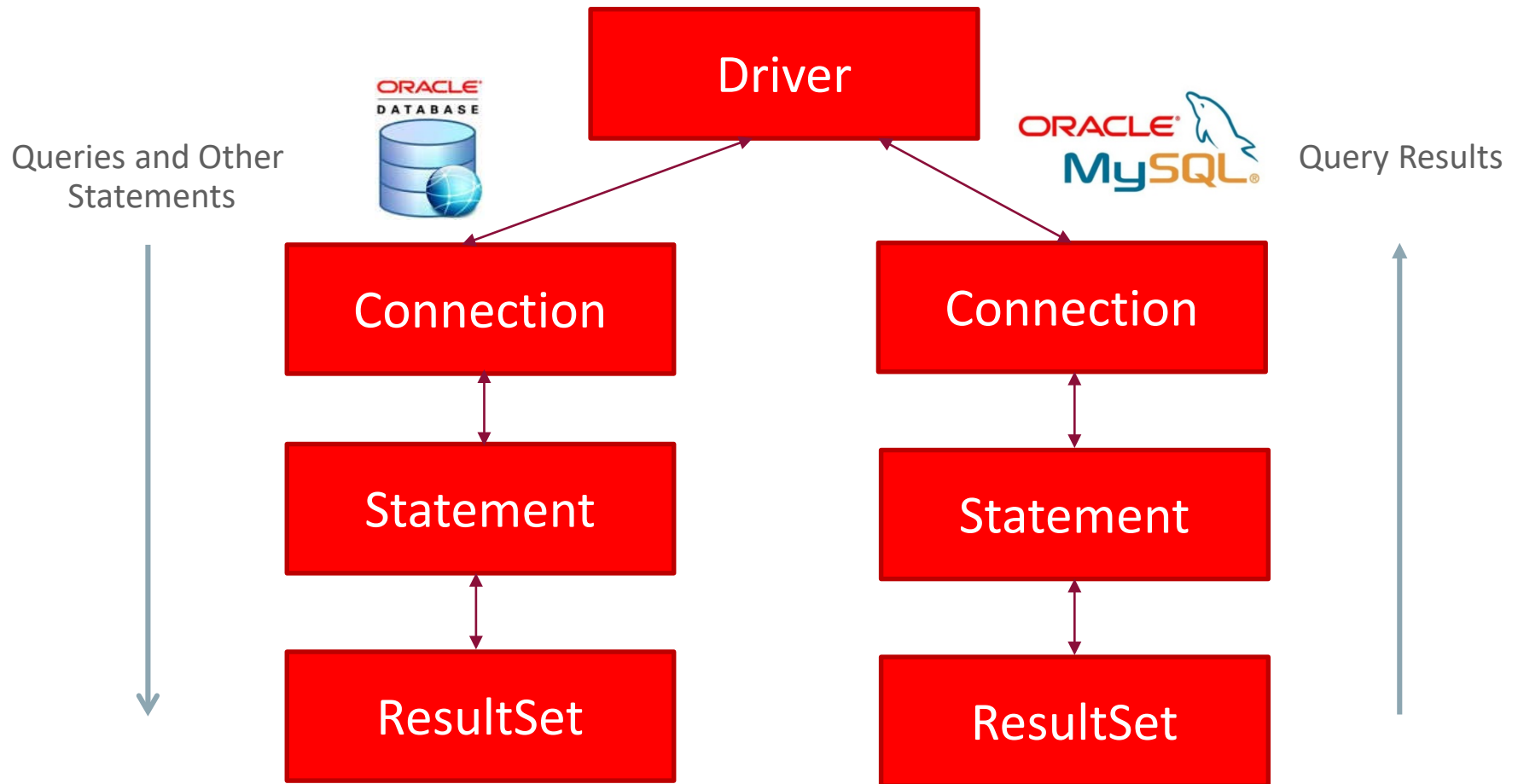
- Connect to a data source, like a database
- Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to your query

JDBC API

The JDBC API is a Java application program interface to generic SQL databases that enables Java developers to develop DBMS-independent Java applications using a uniform interface.

- Importing Packages
- Establishing a Connection
- Creating a Statement
- Executing the Statement
- Retrieving and processing ResultSet object
- Closing ResultsSets, Statements, and Connections

JDBC API



Establishing a Connection - DriverManager

- A JDBC Application connects to a target database using one of the classes:
 - DriverManager: This fully implemented class connects an application to a data source, which is specified by a database URL. When this class first attempts to establish a connection, it automatically loads any JDBC 4.0 drivers found within the class path.
- Note that your application must manually load any JDBC drivers prior to version 4.0.

Establishing a Connection - DataSource

- This interface is preferred over DriverManager because it allows details about the underlying data source to be transparent to your application.
- A DataSource object's properties are set so that it represents a particular data source.
- The DataSource interface is implemented by a driver vendor.

Establishing a Connection - DataSource

There are three types of DataSource implementations:

- Basic implementation
 - Produces a standard Connection object.
- Connection pooling implementation
 - Produces a Connection object that will automatically participate in connection pooling.
 - This implementation works with a middle-tier connection pooling manager.
- Distributed transaction implementation

Loading and registering the Oracle Database

- In order to load and register the Oracle Database, the following packages must be Imported to any application:

```
import java.sql.*;
```

- Provides the API for accessing and processing data stored in a data source.

```
import oracle.jdbc.*;
```

- Oracle Extensions to JDBC.

```
import oracle.jdbc.pool.*;
```

- Provides the OracleDataSource function.

Oracle JDBC URL Format

The Oracle JDBC URL format is:

`jdbc:oracle:<drivertype>:[<username>/<password>]@<database_specifier>`

- Drivertype: thin
- Database_specifier: Oracle Database SID

OracleDataSource

- Oracle implements the `javax.sql.DataSource` interface with the `oracle.jdbc.pool.OracleDataSource` class in the `oracle.jdbc.pool` package.
- This class provides advanced connection caching extensions.
- The `OracleDataSource` class and all subclasses implement the `java.io.Serializable` and `javax.naming.Referenceable` interfaces.

JDBC Connections

- The `java.sql.Connection` interface defines the JDBC Connection object that is obtained from the `DataSources`.
- The `oracle.jdbc.OracleConnection` interface defines an Oracle JDBC Connection object.
- It is Oracle's extension to `java.sql.Connection`.

JDBC Statements

- The objects used for executing a static SQL statement and returning the results it produces.
- The JDBC specification furnishes three types of statements:
 - Statement Interface
 - PreparedStatement Interface
 - CallableStatement Interface

JDBC Statements:

- Statement Interface:
 - The object used for executing a static SQL statement and returning the results it produces.
- PreparedStatement Interface:
 - A SQL statement is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.
- CallableStatement Interface:
 - The interface used to execute SQL stored procedures.

Statement interface:

- An instance of a Statement object is obtained through the invocation of the `createStatement()` method on the Connection object.
- Sample Code:

```
OracleDataSource ods = new OracleDataSource();  
ods.setURL("jdbc:oracle:thin:dfot/dfot@localhost:1521:xe");  
Connection conn = ods.getConnection();  
Statement stmt=conn.createStatement();
```

Running a Query and retrieving a ResultSet Object:

- To query the database, use the executeQuery method of the Statement object.
- This method takes a SQL statement as input and returns a JDBC ResultSet object.

```
ResultSet rset = stmt.executeQuery ("SELECT last_name FROM employees;");
```

Processing the Result Set Object:

- Once you run your query, use the `next()` method of the `ResultSet` object to iterate through the results.
- This method steps through the result set row by row, detecting the end of the result set when it is reached.

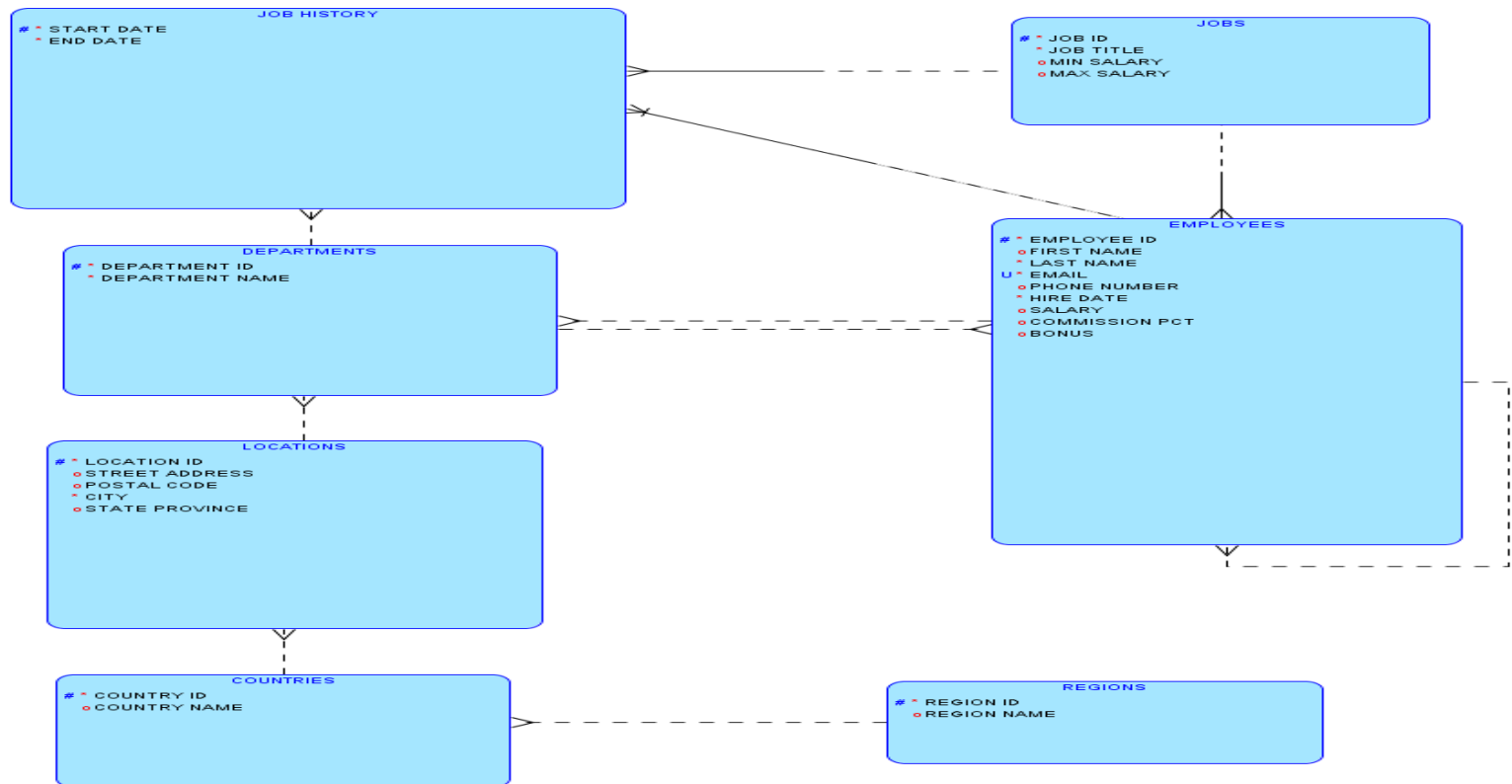
```
while (rset.next())  
    System.out.println (rset.getString(1));
```

Closing the Result Set and Statement Objects:

- You must explicitly close the ResultSet and Statement objects after you finish using them.
- This applies to all ResultSet and Statement objects you create when using Oracle JDBC drivers.
- Sample Code:
`rset.close();`
`stmt.close();`

JDBC Example

In the following example we will use the HR Database Schema.



JDBC Example – Employee table

In the following example we will use the Employee table from the HR Database Schema.

A sample of data from the Employee table :

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	BONUS
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90	(null)
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	(null)	100	90	(null)
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	(null)	100	90	(null)
4	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10	(null)
5	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110	(null)
6	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110	(null)
7	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	0.2	100	80	1500

JDBC Example

```
import java.sql.*;
import oracle.jdbc.pool.OracleDataSource;

class SampleJDBC {
    public static void main(String args[]) throws SQLException {
        // Connect to a database
        OracleDataSource ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:dfot/dfot@localhost:1521:xe");
        Connection conn = ods.getConnection();
        Statement stmt = conn.createStatement();
```

JDBC Example

```
// Execute a statement
    ResultSet rset = stmt.executeQuery("Select Last_Name, Employee_ID
from Employees");
// Iterate through the result and print the employee names and ID
    while (rset.next()) {
        System.out.println(rset.getString(1) + "\t" + rset.getString(2));
    }
    rset.close();
    stmt.close();
}
```


JDBC Example (Cont)

```
D:\>java SampleJDBC
```

```
King 100
```

```
Kochhar 101
```

```
De Haan 102
```

```
Whalen 200
```

```
Higgins 205
```

```
Gietz 206
```

```
Zlotkey 149
```

```
Abel 174
```

```
Taylor 176
```

```
Grant 178
```

```
Mourgos 124
```

```
Rajs 141
```

```
Davies 142
```

Summary

In this lesson, you should have learned how to:

- Describe the JDBC
- Introduce Oracle JDBC Driver
- Outline the steps in JDBC programming
- Describe the JDBC Statement



