



ORACLE

Academy



Java Programming

6-1

JDBC Introduction

ORACLE
Academy



Objectives

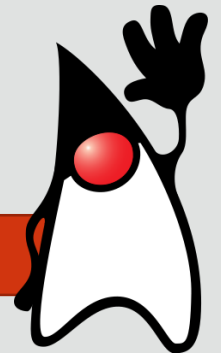
- This lesson covers the following topics:
 - Describe the JDBC
 - Introduce the Oracle JDBC Driver
 - Outline the steps in JDBC programming
 - Describe the JDBC Statement



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

JDBC stands for Java Database Connectivity.



JDBC Packages

- This lesson introduces the application developers interface
- 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)

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

ODBC stands for Open Database Connectivity.



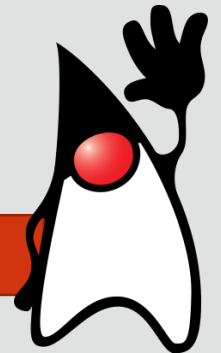
Oracle JDBC Drivers

- 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 4 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

In this course, we will use the JDBC Thin driver.

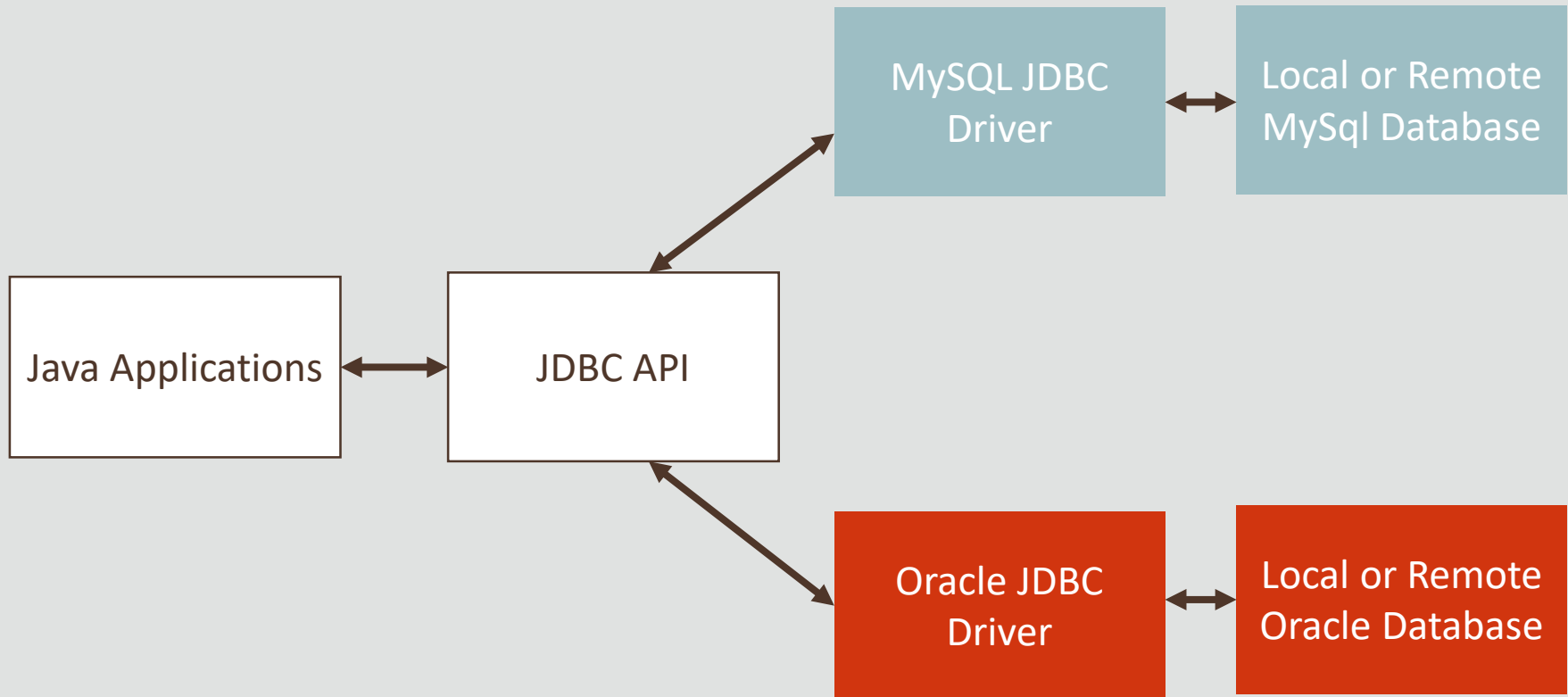


Oracle JDBC Thin Driver

- 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

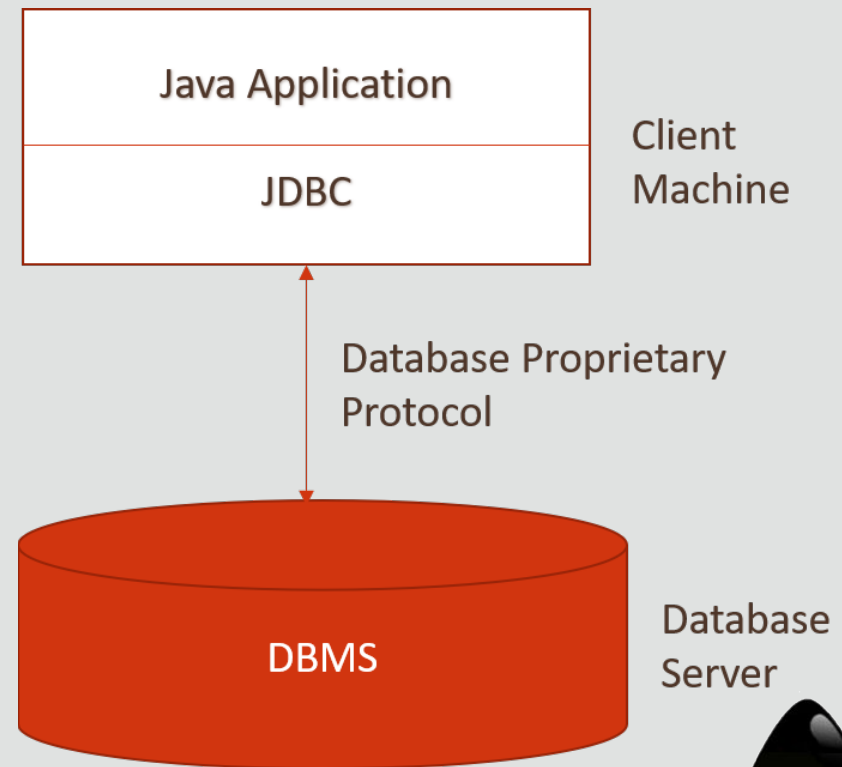


Java Application and JDBC Driver

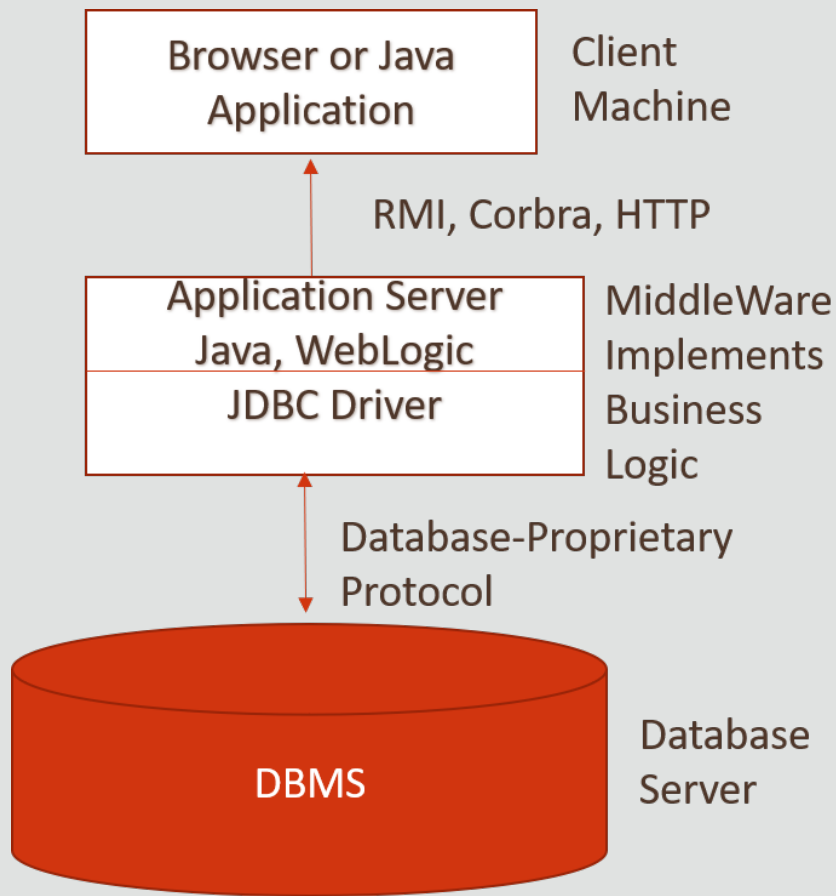


JDBC Architecture (Two-Tier)

- 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)

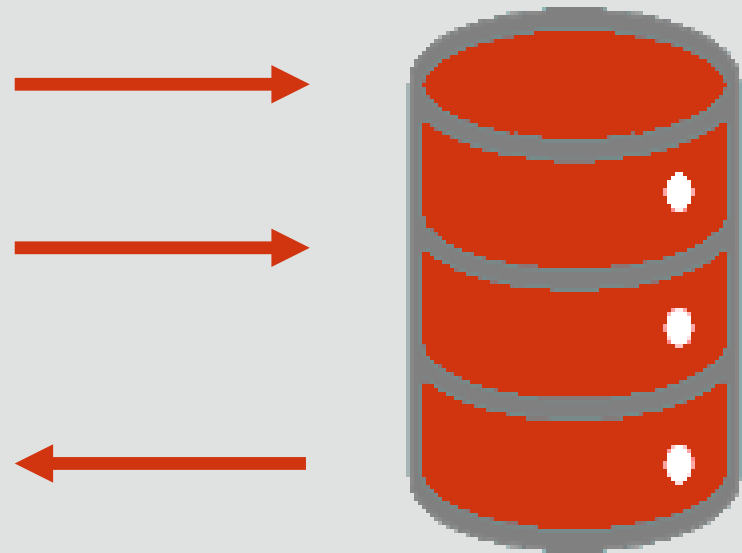


- Requests are sent to the middle tier server, which then sends commands to the data source



Developing a Database Application Using JDBC

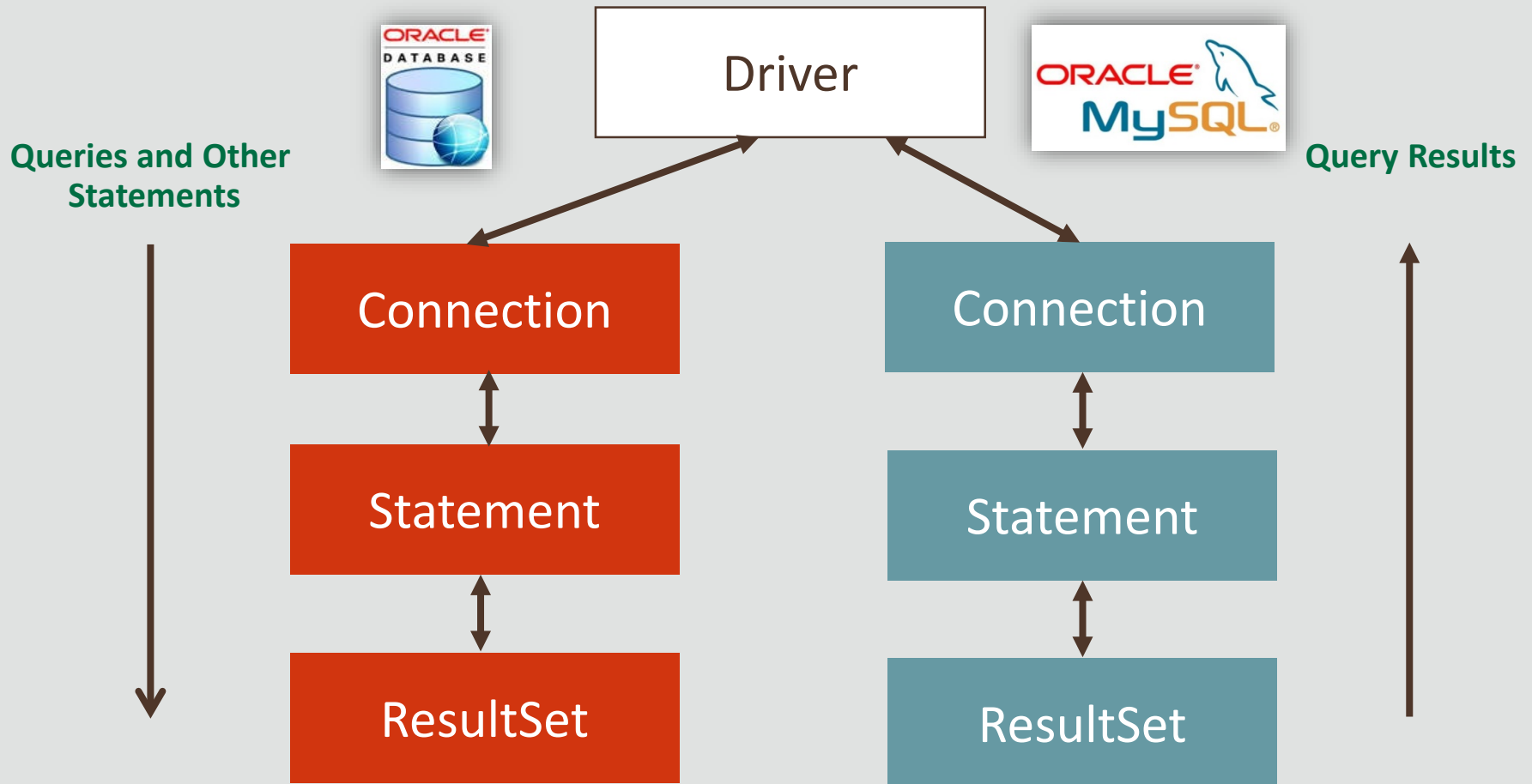
- JDBC helps you to write Java applications that manage these three programming activities:
- Connect to a data source, like a database
- Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to your query



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 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

- DataSource interface
 - The DataSource interface is preferred over the DriverManager class 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



Connecting to a MySQL database

- Open **MySQL_Tutorial.zip** that will take you through the following four activities, explaining how to connect a MySQL database to a Java program using Eclipse
 1. Downloading a MySQL database
 2. Installing a MySQL database
 3. Adding database tables and data to a MySQL database
 4. Using JDBC to connect to a MySQL database



Loading and registering an Oracle Database

- In order to load and register an Oracle Database, some or all of the following packages may have to 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
 - Username: Registered account on the database
 - Password: Associated password for the registered account
 - `Database_specifier`: Oracle Database SID



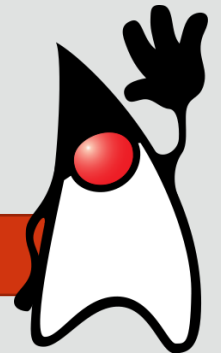
OracleDataSource

- This OracleDataSource class provides advanced connection caching extensions
 - Oracle implements the `javax.sql.DataSource` interface with the `oracle.jdbc.pool.OracleDataSource` class in the `oracle.jdbc.pool` package
 - 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`

More information on all these classes can be found in the Java API.



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**
- These statements are explained in more detail on the next slide

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:

- Sample Code:

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

- An instance of an OracleDataSource object (ods) is created to manage the connection to the database

Statement interface:

- Sample Code:

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

- The setURL method is used to identify the connection criteria to the database including type of **jdbc** being invoked, type of driver used and the connection values required

Statement interface:

- Sample Code:

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

- A Connection object (conn) uses the URL provided through the getConnection() method to connect to the database

Statement interface:

- Sample Code:

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


- An instance of a Statement object (stmt) is obtained through the invocation of the createStatement() method on the Connection object
- This allows SQL statements to be processed on the database

Running a Query and retrieving a ResultSet Object:

- To query the database, use the `executeQuery()` method of the Statement object

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

- This method takes a SQL statement as input and returns a JDBC ResultSet object containing the returned rows of information



The SQL Query
must be enclosed
in quotation marks

Processing the Result Set Object:

- Once you run your query, use the `next()` method of the `ResultSet` object (`rset`) to iterate through the results

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

- The `next()` method steps through the result set row by row, detecting the end of the result set when reached
- `rset.getString(1)` returns the value held in column 1 of the result set

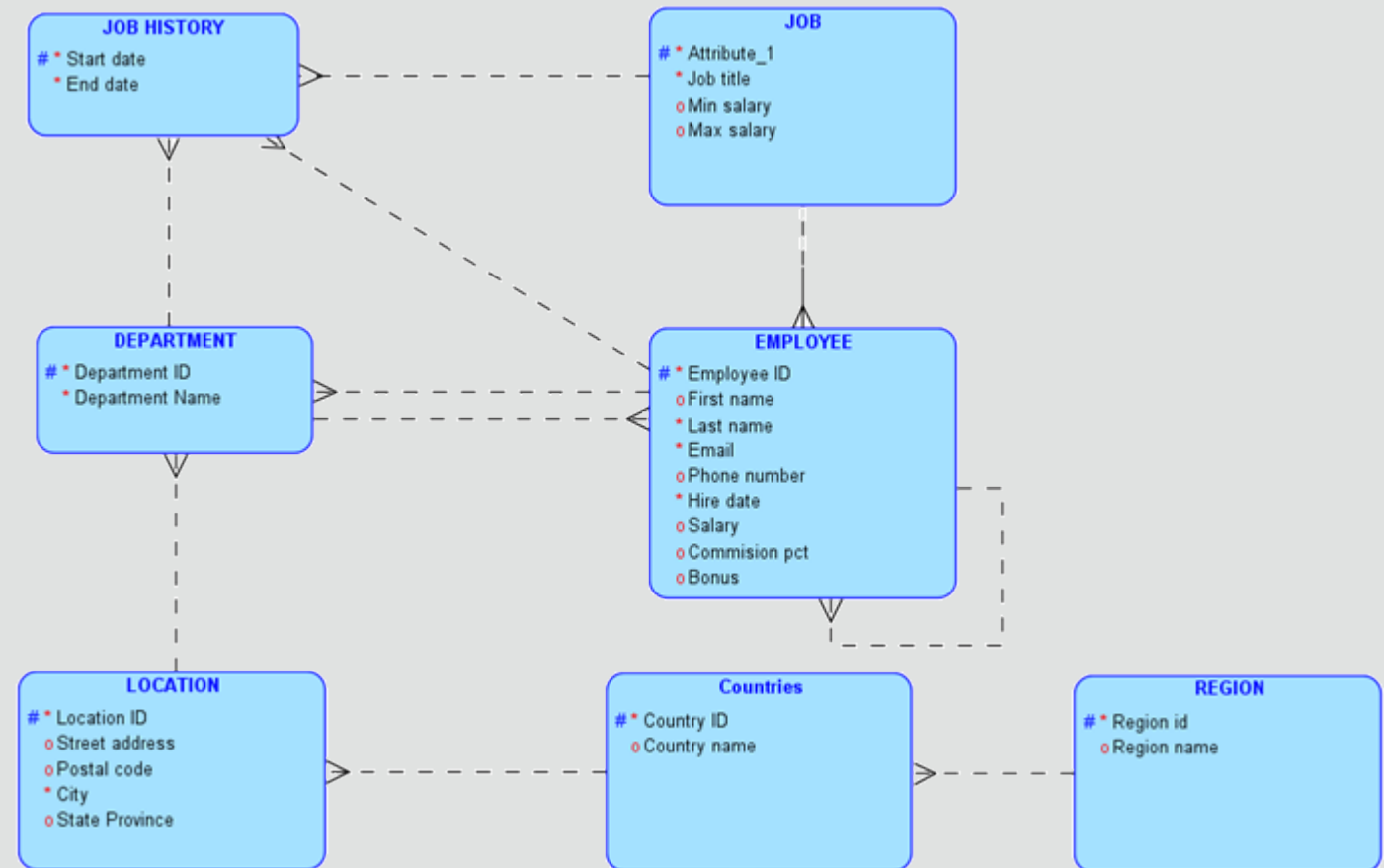
Closing the Result Set and Statement Objects:

- 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 employees table:

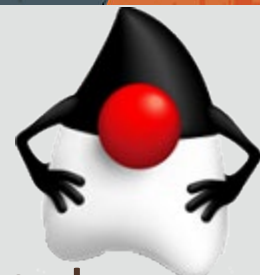
| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID | BONUS |
|---|-------------|------------|-----------|----------|--------------------|-----------|------------|--------|----------------|------------|---------------|--------|
| 1 | 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | (null) | (null) | 90 | (null) |
| 2 | 001 | 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 | WGIEZT | 515.123.8181 | 07-JUN-94 | AC_ACCOUNT | 8300 | (null) | 205 | 110 | (null) |
| 7 | 149 | Eleni | Zlotkey | EZLOTKEY | 0.1144.1344.429018 | 29-JAN-00 | SA_MAN | 10500 | 0.2 | 100 | 80 | 1500 |

Connecting to an Oracle Cloud Database

- Open **OracleCloudDB_Tutorial.zip** that will take you through the steps to Create, configure an Oracle database. An Oracle Database is required to complete the practical activities for the rest of the course

1. Creating an Oracle Cloud Account
2. Setting up a database using APEX in the cloud
3. Downloading the Cloud connection credentials





Setting Up JDBC Task - Cloud

- **Quick Task:** Connect to a Cloud based Oracle database using JDBC (Instructions for a local database will be shown later in this section)
 - a) Go to the JDBC downloads page of the Oracle website:
 - <https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html>
 - b) Accept the license agreement
 - c) Download the latest full JDBC file to your local machine

At time of writing the latest jar was 11

| Download | Release Notes |
|-------------------------------------|--|
| ojdbc11-full.tar.gz | This archive contains the latest 21.1 JDBC Thin driver (ojdbc11.jar), the Universal Connection Pool (ucp.jar), their Readme(s) and companion jars. (11,227,593 bytes) - (SHA1: 8f6cd8d0c743b6c8b6f9e340d8862dbf508fa9d1) |



JDBC Example - Cloud

- Create the following program that will test your connection to the database:
1. Create a new project and package in Eclipse named **oraclecloudconnection** (use the latest JDK(This exercise uses JDK 15))
 2. Click Next
 3. Do not create a module file

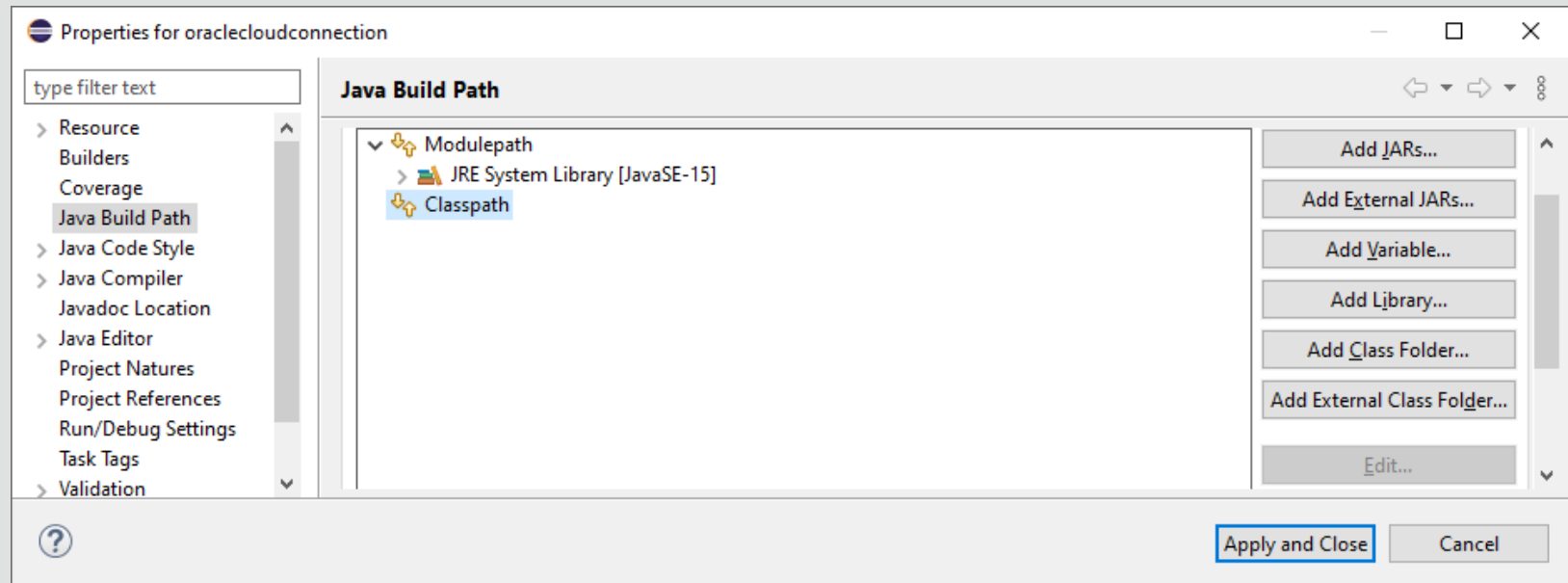
- ☐ Allow output folders for source folders
- ☐ Create module-info.java file

4. Click Finish to create the application



JDBC Example

5. Right click on the project and select properties
6. Select Java Build Path and then Libraries

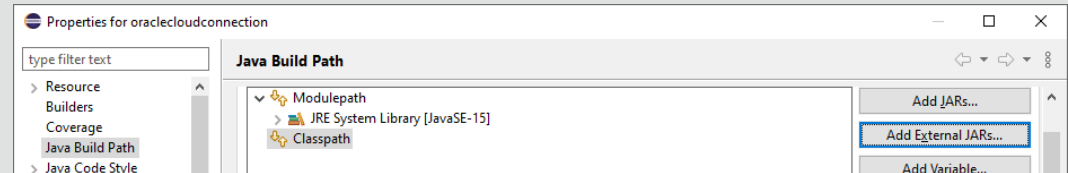


7. Click on Classpath



JDBC Example

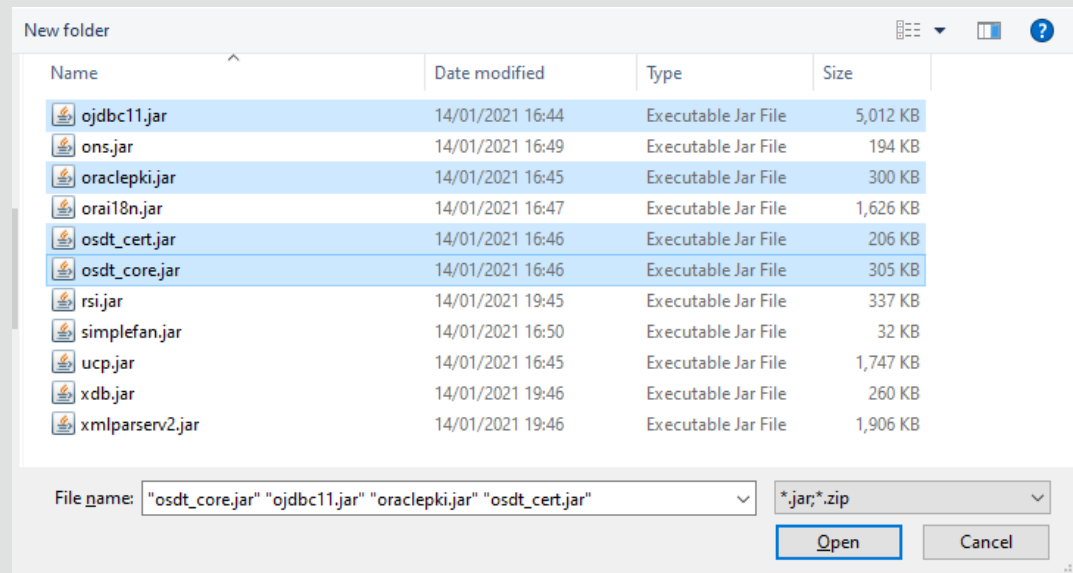
8. Click Add External JARs



9. Browse to the directory where you extracted the full Oracle JDBC jar file that you downloaded, select

- ojdbc11.jar
- oraclepki.jar
- osdt_cert.jar
- osdt_core.jar

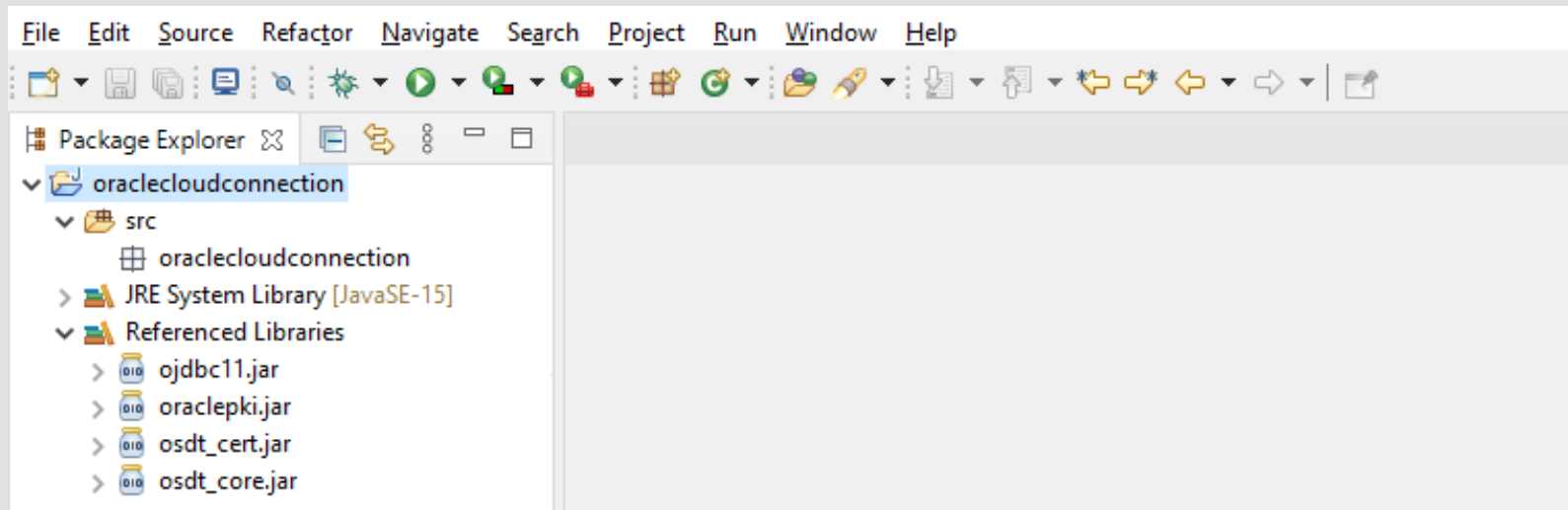
10. Click Open





JDBC Example

11. Click on Apply and Close and you will see the jar files under referenced libraries in the project explorer



12. Right click the package and create a driver class named OracleCloudConnection that includes a main method



JDBC Example - Cloud

13. You will need to setup instance variables to hold the connection details to the database

- a) If you are using the cloud database, you will need to know the following information:
- Database name: This can be found by opening the tnsnames.ora file that can be found in the extracted wallet directory (It will have the database name followed by _medium(db2021012999220_medium))
 - The directory path to the Wallet: On windows an example of this would be; drive letter:/directory path/Wallet Name e.g.
D:/cloud_connection/Wallet_DB2021012999220

final static String

```
DB_URL="jdbc:oracle:thin:@db202101290850_medium?TNS_
ADMIN=D:/cloud_connection/Wallet_DB2021012999220/";
```



JDBC Example - Cloud

13. You will need to setup instance variables to hold the connection details to the database
- b) You will require to provide valid username that has access to the required database schema
 - `final static String DB_USER = "user_01";`
 - c) You will require to provide password for that username
 - `final static String DB_PASSWORD = "UserPassword1";`



JDBC Example - Cloud

14. Add the following code above main to handle the connection to the cloud-based database:

```
public class OracleConnection {  
  
    final static String  
DB_URL="jdbc:oracle:thin:@db2021012999220_medium?TNS_ADMIN=D:/cCloud_conn/  
wallet_DB202102091440/";  
    final static String DB_USER = "user_01";  
    final static String DB_PASSWORD = "UserPassword1";  
  
    public static void main(String[] args){  
  
    }//end method main  
  
}//end class OracleConnection
```

Code continued on the next slide....



JDBC Example - Cloud

15. You now need to set up the properties of the connection, properties are configuration values managed as key/value pairs

In each pair, the key and value are both String values

```
public static void main(String[] args){  
  
    Properties info = new Properties();  
    info.put(OracleConnection.CONNECTION_PROPERTY_USER_NAME, DB_USER);  
    info.put(OracleConnection.CONNECTION_PROPERTY_PASSWORD, DB_PASSWORD);  
    info.put(OracleConnection.CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH, "20");  
  
} //end method main
```




JDBC Example - Cloud

16. You will have been required to import the following libraries with this code:

```
package oraclecloudconnection;

import java.util.Properties;
import oracle.jdbc.OracleConnection;

public class OracleCloudConnection {
    .
    .
    .

} //end class OracleConnection
```

Code continued on the next slide....



JDBC Example - Cloud

17. You now need to set up the data source connection that requires the DB URL and property information:

```
import oracle.jdbc.pool.OracleDataSource;
..
public static void main(String[] args) throws SQLException {
    ..
    info.put(OracleConnection.CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH, "20");

    OracleDataSource ods = new OracleDataSource();
    ods.setURL(DB_URL);
    ods.setConnectionProperties(info);

} //end method main
```

18. Add the required import and throws declaration



JDBC Example - Cloud

19. You are now going to create the connection to the database, you will use the `AutoCloseable` method that ensures the connection is closed automatically

```
public static void main(String[] args) throws SQLException {  
    ..  
    ods.setConnectionProperties(info);  
  
    try (OracleConnection connection = (OracleConnection) ods.getConnection())  
    {  
  
    }//end try  
  
}//end method main
```

20. You should now have a successful connection



JDBC Example - Cloud

You can now create methods under main to work with the database connection

21. To view the database details:

```
//end method main

public static void displayDatabaseDetails(Connection connection) throws SQLException
{
    // Print some connection properties
    DatabaseMetaData dbmd = connection.getMetaData();
    System.out.println("Driver Name: " + dbmd.getDriverName());
    System.out.println("Driver Version: " + dbmd.getDriverVersion());
    System.out.println("Default Row Prefetch Value is: "
        + ((OracleConnection) connection).getDefaultRowPrefetch());
    System.out.println("Database Username is: "
        + ((OracleConnection) connection).getUserName());
    System.out.println();
}
//end method displayDatabaseDetails
```

Code continued on the next slide....



JDBC Example - Cloud

22. Add a method call in the try statement to call the method, then test the code

```
try (OracleConnection connection = (OracleConnection) ods.getConnection())
{
    displayDatabaseDetails(connection);
} //end try
} //end method main

public static void displayDatabaseDetails(Connection connection) throws
SQLException {
    .
    .
} //end method displayDatabaseDetails
```

23. You should now have a successful connection



JDBC Example - Cloud

24. Now you are going to create a method that will retrieve information from the database

```
//end method main  
  
public static void displayEmployees(Connection connection)  
                                throws SQLException {  
  
} //end method displayEmployees
```



JDBC Example - Cloud

25. You are again going to use the Autocloseable way of creating both the statement and ResultSet objects, you will need to import the relevant sql libraries

```
public static void displayEmployees(Connection connection) throws
SQLException {
    // Statement and ResultSet are AutoCloseable and closed automatically.
    try (Statement statement = connection.createStatement()) {
        try (ResultSet resultSet =
statement.executeQuery("SELECT first_name, last_name FROM employees")) {

            }//end try ResultSet
        }//end try Statement
    }//end method displayEmployees
```

You place your SQL query in the brackets of the executeQuery method from the statement class



Code continued on the next slide...



JDBC Example - Cloud

- 26. The ResultSet stores the return from the query and can be used to display the results to the console
- 27. The .next method is used to identify further rows in the ResultSet
- 28. The getString value identifies the number of columns

```
try (Statement statement = connection.createStatement()) {  
    try (ResultSet resultSet =  
        statement.executeQuery("select first_name, last_name from employees")) {  
        System.out.println("First Name" + " " + "Last Name");  
        System.out.println("-----");  
        while (resultSet.next())  
            System.out.println(resultSet.getString(1) + " "  
                               + resultSet.getString(2) + " ");  
    } //end try ResultSet  
} //end try Statement
```

Code continued on the next slide....



JDBC Example - Cloud

29. Update the method calls in main to also call the display employees method

```
try (OracleConnection connection = (OracleConnection) ods.getConnection())
{
    displayDatabaseDetails(connection);
    displayEmployees(connection);
} //end try
} //end method main
```



JDBC Example (Cont)

30. The code should produce the following output:

```
Driver Name: Oracle JDBC driver
Driver Version: 21.1.0.0.0
Default Row Prefetch Value is: 20
Database Username is: USER_01
```

```
FIRST_NAME  LAST_NAME
-----
```

```
Ellen Abel
Curtis Davies
Lex De Haan
Bruce Ernst
Pat Fay
William Gietz
Kimberely Grant
Michael Hartstein
Shelley Higgins
Alexander Hunold
Steven King
Neena Kochhar
Diana Lorentz
Randall Matos
Kevin Mourgos
Trenna Rajs
Jonathon Taylor
Peter Vargas
Jennifer Whalen
Eleni Zlotkey
```



JDBC Example - Cloud

31. Add the following SQL statement that uses a join and concatenation

```
public static void displayManagers(Connection connection) throws
SQLException {
    // Statement and ResultSet are AutoCloseable and closed automatically.
    try (Statement statement = connection.createStatement()) {
        try (ResultSet resultSet = statement.executeQuery("select
e.first_name||' '||e.last_name||' is managed by '||m.first_name||'
'||m.last_name FROM employees e JOIN employees m ON (e.manager_id =
m.employee_id)")) {
            System.out.println("\nManagement Information");
            System.out.println("-----");
            while (resultSet.next())
                System.out.println(resultSet.getString(1));
        } //end try ResultSet
    } //end try Statement
} //end method displayEmployees
```

Code continued on the next slide....



JDBC Example - Cloud

32. Confirm that this is your code in main and test that your program can talk to the database

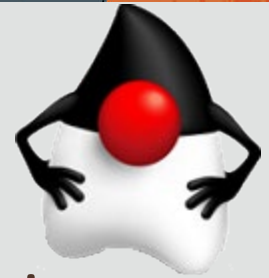
```
public static void main(String[] args) throws SQLException {  
  
    Properties info = new Properties();  
    info.put(OracleConnection.CONNECTION_PROPERTY_USER_NAME, DB_USER);  
    info.put(OracleConnection.CONNECTION_PROPERTY_PASSWORD, DB_PASSWORD);  
    info.put(OracleConnection.CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH, "20");  
  
    OracleDataSource ods = new OracleDataSource();  
    ods.setURL(DB_URL);  
    ods.setConnectionProperties(info);  
  
    try (OracleConnection connection = (OracleConnection) ods.getConnection()) {  
        displayDatabaseDetails(connection);  
        displayEmployees(connection);  
        displayManagers(connection);  
    } //end try  
} //end method main
```



Connecting to a local Oracle database

- Open **OracleDB_Tutorial.zip** that will take you through the steps to download, install and configure an Oracle database. An Oracle Database is required to complete the practical activities for the rest of the course
1. Downloading Oracle Database Express Edition(XE)
 2. Installing Oracle Database Express Edition(XE)
 3. Configuring Oracle Database Express Edition(XE)





Setting Up JDBC Task

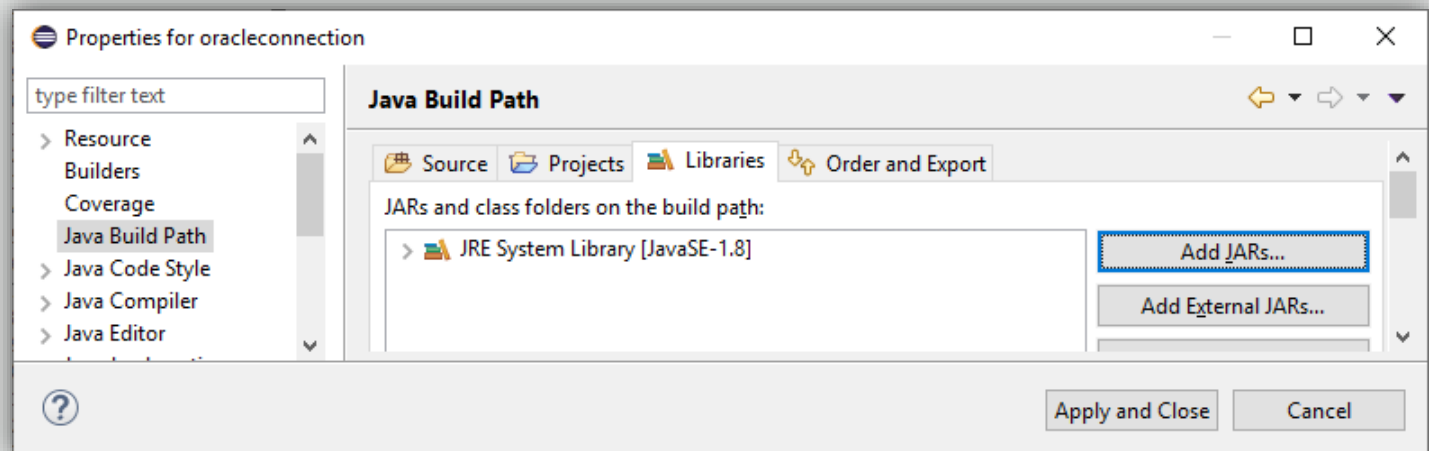
- **Quick Task:** To connect to an Oracle database using JDBC you will require the correct JAR file that contains all the connection instructions
 - a) Go to the JDBC downloads page of the Oracle website:
 - <https://www.oracle.com/technetwork/database/application-development/jdbc/downloads/jdbc-ucp-183-5013470.html>
 - b) Accept the license agreement
 - c) Download the latest JDBC jar file to your local machine

📄 **ojdbc8.jar** the Oracle JDBC driver except classes for NLS support in Oracle Object and Collection types.
(4,161,744 bytes - SHA1: 4acaa9ab2b7470fa80f0a8ec416d7ea86608ac8c)



JDBC Example

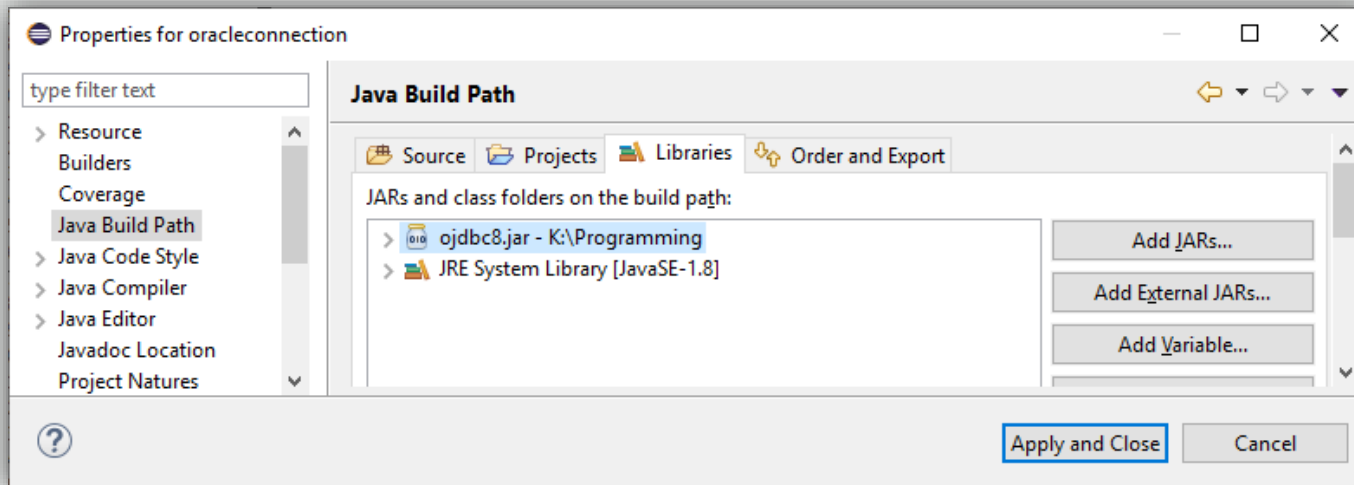
- Create the following program that will test your connection to the database:
1. Create a new project and package in Eclipse named oracleconnection
 2. Right click on the project and select properties
 3. Select Java Build Path and then Libraries





JDBC Example

4. Click Add External JARs and browse to the Oracle JDBC jar file that you downloaded and click Open



5. Click on Apply and Close and you will see the jar file under referenced libraries in the project explorer



JDBC Example

6. Add the following code in main to handle the connection to the XEPDB1 pluggable database:

```
public class OracleConnection {  
    public static void main(String[] args){  
        // Connect to a database  
        OracleDataSource ods;  
        try {  
            ods = new OracleDataSource();  
            ods.setURL("jdbc:oracle:thin:orcluser/jdbcuser@localhost:1521/xepdb1");  
            Connection conn = ods.getConnection();  
            Statement stmt = conn.createStatement();  
  
            stmt.close();  
        } catch (SQLException e) {  
            System.out.println(e);  
        } //end try catch  
    } //end method main  
} //end class OracleConnection
```

Code continued on the next slide....



JDBC Example

7. You will have been required to import the following libraries with this code:

```
package oracleconnection;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import oracle.jdbc.pool.OracleDataSource;

public class OracleConnection {

    public static void main(String[] args){
        .
        .
    }//end method main
}//end class OracleConnection
```

Code continued on the next slide....



JDBC Example (Cont)

8. Add the following code that utilizes the statement object to execute a SQL query and display the results to the console

```
Statement stmt = conn.createStatement();
// Execute a statement
ResultSet rset = stmt.executeQuery("SELECT last_name,
                                   first_name FROM employees");
//Iterate through the result and print the employee last and
first names
while (rset.next()) {
    System.out.println(rset.getString(1) + "-" +
                       rset.getString(2));
} //end while
rset.close();
stmt.close();
```



JDBC Example (Cont)

9. The code should produce the following output:

```
Abel-Ellen  
Davies-Curtis  
De Haan-Lex  
Ernst-Bruce  
Fay-Pat  
Gietz-William  
Grant-Kimberely  
Hartstein-Michael  
Higgins-Shelley  
Hunold-Alexander  
King-Steven  
Kochhar-Neena  
Lorentz-Diana  
Matos-Randall  
Mourgos-Kevin  
Rajs-Trenna  
Taylor-Jonathon  
Vargas-Peter  
Whalen-Jennifer  
Zlotkey-Eleni
```



JDBC Example (Cont) - local

10. Add the column method you used in the MySQL example under the main method

```
public static int getColumnNames(ResultSet rs) throws SQLException {  
    int numberOfColumns = 0;  
    if (rs != null) {  
        //create an object based on the Metadata of the result set  
        ResultSetMetaData rsMetaData = rs.getMetaData();  
        //Use the getColumn method to get the number of columns returned  
        numberOfColumns = rsMetaData.getColumnCount();  
        //get and print the column names, column indexes start from 1  
        for (int i = 1; i < numberOfColumns + 1; i++) {  
            String columnName = rsMetaData.getColumnName(i);  
            System.out.print(columnName + ", ");  
        } //endfor  
    } //endif  
    System.out.println(); //place the cursor on a new line in the console  
    return numberOfColumns;  
} //end method getColumnNames
```

ORACLE

Academy



JDBC Example (Cont) - local

11. Update the ResultSet code to utilize the method

```
// Execute a statement
ResultSet rset = stmt.executeQuery("SELECT last_name, first_name
                                   FROM employees");

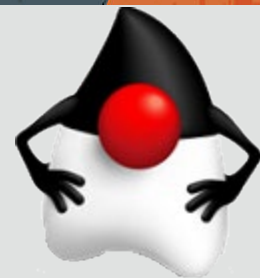
int colNum = getColumnNames(rset);
if(colNum>0)
    while(rset.next()) {
        for(int i =0; i<colNum; i++) {
            if(i+1 == colNum)
                System.out.println(rset.getString(i+1));
            else
                System.out.print(rset.getString(i+1)+ ", ");
        }//endfor
    }//endwhile
//endif
rset.close();
```



JDBC Example (Cont) - local

12. The code should produce the following output:

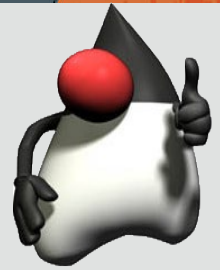
```
LAST_NAME, FIRST_NAME,  
Abel, Ellen  
Davies, Curtis  
De Haan, Lex  
Ernst, Bruce  
Fay, Pat  
Gietz, William  
Grant, Kimberly  
Hartstein, Michael  
Higgins, Shelley  
Hunold, Alexander  
King, Steven  
Kochhar, Neena  
Lorentz, Diana  
Matos, Randall  
Mourgos, Kevin  
Rajs, Tenna  
Taylor, Jonathon  
Vargas, Peter  
Whalen, Jennifer  
Zlotkey, Eleni
```



Using JDBC Task - Local

- **Quick Task:** Manipulate the query to display the following:
 - a) Create variables for the username, password and query
 - b) Use the variable names in your code instead of the current hard coded values
 - c) Change the query to display the department_id, department_name and location_id for every department
 - d) Change the query to display the location_id, postal_code, city and state_province for every location

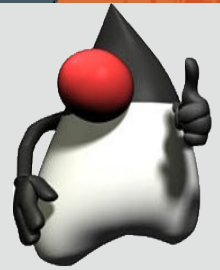




Using JDBC Suggested Solution

- a) Create variables for the username, password and query

```
public static void main(String[] args){  
    String username = "orcluser";  
    String password = "jdbcuser";  
    String query = "SELECT last_name, first_name FROM employees";  
}
```



Using JDBC Suggested Solution

- b) Use the variable names in your code instead of their values

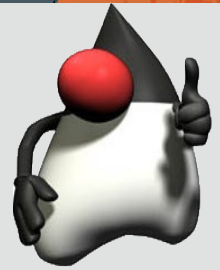
```
public static void main(String[] args){
    String username = "orcluser";
    String password = "jdbcuser";
    String query = "SELECT last_name, first_name FROM employees";
    OracleDataSource ods;
    try {
        ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:"+username+"/"
            +password+"@localhost:1521/xepdb1");
        Connection conn = ods.getConnection();
        Statement stmt = conn.createStatement();
        // Execute a statement
        ResultSet rset = stmt.executeQuery(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



Using JDBC Suggested Solution

- c) Change the query to display the department_id, department_name and location_id for every department

```
public static void main(String[] args){  
    // Connect to a database  
    String username = "orcluser";  
    String password = "jdbcuser";  
    String query = "SELECT department_id, department_name,  
                                     location_id FROM departments";  
    OracleDataSource ods;
```



Using JDBC Suggested Solution

- d) Change the query to display the location_id, postal_code, city and state_province for every location

```
public static void main(String[] args){  
    // Connect to a database  
    String username = "orcluser";  
    String password = "jdbcuser";  
    String query = "SELECT location_id, postal_code, city,  
                                     state_province FROM locations";  
    OracleDataSource ods;
```



JDBC Example (Cont) - local

13. In the previous task you had separate queries that got department and location information
14. As the tables are joined by the `location_id` field you can return the information in a single query

```
// Create a SQL statement using two tables with a join
String query = ("SELECT * FROM departments JOIN locations USING
                (location_id)");
```

- `SELECT *` - returns all columns from the specified tables
- `departments join locations` – Return information when there is a match between both tables (in this case when the `location_id` value is the same in both tables)
- `using (location_id)` – specifies what field is to be used as the join in both tables



JDBC Example (Cont) - local

15. The code should produce the following output:

```
LOCATION_ID, DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID,  
1400, 60, IT, 103, 2014 Jabberwocky Rd, 26192, Southlake, Texas, US  
1500, 50, Shipping, 124, 2011 Interiors Blvd, 99236, South San Francisco, California, US  
1700, 190, Contracting, null, 2004 Charade Rd, 98199, Seattle, Washington, US  
1700, 90, Executive, 100, 2004 Charade Rd, 98199, Seattle, Washington, US  
1700, 110, Accounting, 205, 2004 Charade Rd, 98199, Seattle, Washington, US  
1700, 10, Administration, 200, 2004 Charade Rd, 98199, Seattle, Washington, US  
1800, 20, Marketing, 201, 460 Bloor St. W., ON M5S 1X8, Toronto, Ontario, CA  
2500, 80, Sales, 149, Magdalen Centre, The Oxford Science Park, OX9 9ZB, Oxford, Oxford, UK
```

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





ORACLE

Academy

