
ORACLE
Academy

Java Programming

6-2
JDBC Basics

ORACLE
Academy

Objectives

- This lesson covers the following topics:
  - JDBC Data Types
  - Programming with JDBC PreparedStatement
  - Programming with JDBC CallableStatement
  - Reading MetaData from Database

JDBC Data Types:

- The Java Language has a data type system, for example:
  - boolean
  - int
  - long
  - float
  - double
  - String
- Oracle Databases systems also have a data type system, for example:
  - integer
  - char
  - varchar2
  - text
  - blob
  - clob

JDBC Data Types:

- The Oracle JDBC Drivers support standard JDBC data type as well as Oracle-specific datatypes
- The JDBC driver can convert a Java data type to the appropriate database type and vice versa
- The JDBC type system controls the conversion between Oracle data types and Java language types and objects
- The JDBC types are modeled on the SQL-92 and SQL-99 version types

JDBC Data Types:

- Some sample data types:

SQL Datatypes	JDBC Typecodes STANDARD JDBC 1.0 TYPES:	Standard Java Types	Oracle Extension Java Types
CHAR	java.sql.Types.CHAR	java.lang.String	oracle.sql.CHAR
VARCHAR2	java.sql.Types. VARCHAR	java.lang.String	oracle.sql.CHAR
LONG	java.sql.Types. LONGVARCHAR	java.lang.String	oracle.sql.CHAR
NUMBER	java.sql.Types. NUMERIC	java.math. BigDecimal	oracle.sql. NUMBER
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE

A full list of JDBC Data Types can be found at https://docs.oracle.com/en/database/oracle/oracle-database/21/sqirf/Data-Types.html

The PreparedStatement Object

- For a Database, calling a precompiled SQL statement is more efficient than repeatedly calling the same SQL statement
- It extends the Statement interface
- The setter methods (setShort, setString and so on) for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter
- Consult the table on the previous slide to compare Java/SQL data types

The PreparedStatement Object

- A prepared statement is a precompiled SQL statement
  - The SQL statement is parsed only once
  - The PreparedStatement interface extends the Statement interface to add the capability of passing parameters inside of a statement
  - If the same SQL statements are executed multiple times, use a PreparedStatement object

The PreparedStatement Object

- Syntax for assigning a query to prepared statement

PreparedStatement pstm =
conn.prepareStatement(sqlString);

- Example 1:

```
PreparedStatement pstmt =
        conn.prepareStatement("UPDATE employees "+
                              "SET salary = ? " +
                              "WHERE id = ?");
pstmt.setBigDecimal(1, 100000.00);
pstmt.setInt(2, 200);
ResultSet rs = pstmnt.executeQuery();
```

This prepared statement takes 2 parameters (signified by the question marks (?). They are numbered based on the order of their appearance in the SQL query, salary is 1, id is 2.

The PreparedStatement Object

- Parameter 1
  - pstmt.setBigDecimal() is used as the decimal parameter for the salary (100000.00) as it converts to a SQL NUMERIC value when it sends it to the database
- Parameter 2
  - pstmt.setInt() is used to pass the integer value (200) for the id to the database

```
PreparedStatement pstmt =
        conn.prepareStatement("UPDATE employees "+
                              "SET salary = ? " +
                              "WHERE id = ?");
pstmt.setBigDecimal(1, 100000.00);
pstmt.setInt(2, 200);
ResultSet rs = pstmnt.executeQuery();
```

The PreparedStatement Object

- Syntax for assigning a query to prepared statement

PreparedStatement pstm =
conn.prepareStatement(sqlString);

- Example 2

```
PreparedStatement prepStmnt =
    conn.prepareStatement("SELECT department_id "+
                          "FROM departments "+
                          "WHERE department_name = ?");
prepStmnt.setString(1, "Marketing");
ResultSet rs = prepStmnt.executeQuery();
```

This prepared statement takes 1 parameter (signified by the question mark (?), department_name is parameter 1.

The PreparedStatement Object

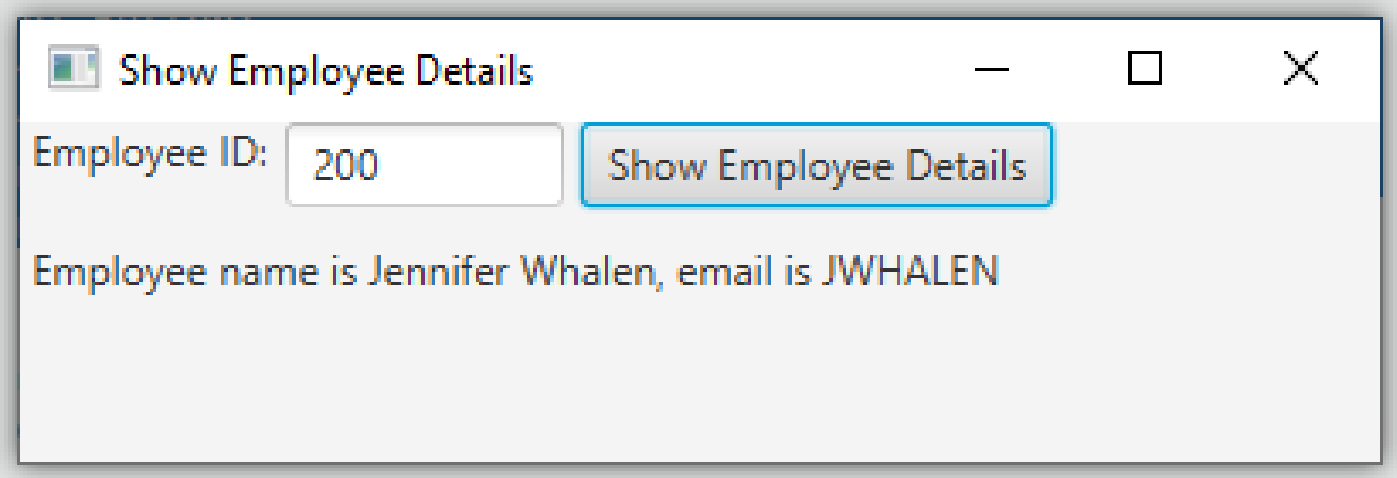
- Parameter 1
  - prepStmnt.setString() is used to set the value of parameter 1 to Marketing

```
PreparedStatement prepStmnt =
    conn.prepareStatement("SELECT department_id "+
                          "FROM departments "+
                          "WHERE department_name = ?");
prepStmnt.setString(1, "Marketing");
ResultSet rs = prepStmnt.executeQuery();
```

Remember that text values must be enclosed in quotes!

PreparedStatement: Retrieving Data

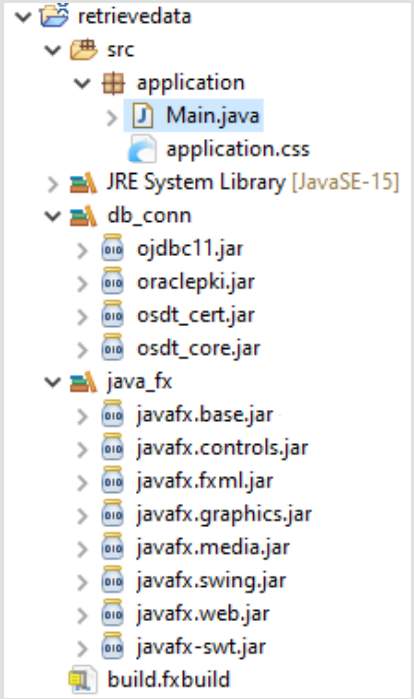
- The following JavaFX application establishes a database connection
- The Program prompts the user to enter the ID of an employee to find the employee's full name and email address



PreparedStatement: Retrieving Data

- This exercise uses the either a local or Cloud Oracle database (steps 7 and 8 are specific)

1. Open Eclipse and install the FX plugin if you haven't already (there is a tutorial on this in section 5.3)
2. Create a JavaFX Project named retrievedata
3. Add the FX Jar files to your build path and update the run configuration
4. Add the Oracle JDBC jar files to your build path



PreparedStatement: Retrieving Data

```
public class Main extends Application {
   @Override
   public void start(Stage primaryStage) {
      try {
         BorderPane root = new BorderPane();
         Scene scene = new Scene(root,400,400);

         primaryStage.setScene(scene);
         primaryStage.show();
      } catch(Exception e) {
           e.printStackTrace();
      }//end trycatch
   }//end method start

   public static void main(String[] args) {
      launch(args);
   }//end method main
}//end class Main
```

PreparedStatement: Retrieving Data

6. Create the following instance fields at the top of the Main class:

```
public class Main extends Application {

    private Connection conn;
    private PreparedStatement pstmt;
    private ResultSet rset;
    private TextField tfid = new TextField();
    private Label lblResult = new Label();

    @Override
     public void start(Stage primaryStage) {
```

When prompted to import libraries select the java.sql or the javafx option from the pop up menu!

Database Type

- If you are using the Always Free Cloud instance, then use the instructions on the next 2 slides (18 and 19) to initialise the database connection

- If you are using a local database (Oracle XE) then use slides 20 and 21 to initialise the database connection

PreparedStatement: Retrieving Data

7. Create the following method between the start and main methods:


```
    }//end method start

    public void initializeDatabase() {
        final String DB_URL =
"jdbc:oracle:thin:@db202102091440_medium?TNS_ADMIN=D:/cloud_conn/wallet_D
B202102091440/";
        final String DB_USER = "user_01";
        final String DB_PASSWORD = "UserPassword1";
        String query = "SELECT email, first_name, last_name "
                     +"FROM employees "
                     +"where employee_id = ?";
    }//end method initializeDatabase
```

This sets up the login information and query that will be used to create the ResultSet to hold the returned data from the database.

PreparedStatement: Retrieving Data

8. Update the initializeDatabase method to create a connection to the database:


```
try {
  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);
  conn = ods.getConnection();
  pstmt = conn.prepareStatement(query);
}catch (Exception e) {
     e.printStackTrace();
}//end try catch
}//end method initializeDatabase
```

Stores the prepared statement query in the **pstmt** instance field.

PreparedStatement: Retrieving Data

7. Create the following method between the start and main methods:


```
}//end method start

public void initializeDatabase() {
    String user = "orcluser";
    String password = "jdbcuser";
    String query = "SELECT email, first_name, last_name "
                  +"FROM employees "
                  +"where employee_id = ?";
}//end method initializeDatabase

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

This sets up the login information and query that will be used to create the ResultSet to hold the returned data from the database.

PreparedStatement: Retrieving Data

8. Update the initializeDatabase method to create a connection to the database:


```
                +"where employee_id = ?";
    try {
       OracleDataSource ods = new OracleDataSource();
       ods.setURL("jdbc:oracle:thin:" + user + "/"
                  + password + "@localhost:1521/xepdb1");
       conn = ods.getConnection();
       pstmt = conn.prepareStatement(query);
    }catch (Exception e) {
       e.printStackTrace();
    }//end try catch
}//end method initializeDatabase

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

Stores the prepared statement query in the **pstmt** instance field.

PreparedStatement: Retrieving Data

9. Create the following method between the start and initializeDatabase methods:

```
}//end method start

private void showResult() {
    String employeeID = tfid.getText();
    try {
    } catch (SQLException e) {
        e.printStackTrace();
    }//end try catch
}//end method showResult

public void initializeDatabase() {
```

This method will use the text entered in the text field of the interface for the prepared statement to find the employee in the database table.

PreparedStatement: Retrieving Data

10. Add the following code to the try statement in the showResults method:

```
String employeeID = tfid.getText();
try {
    pstmt.setBigDecimal(1, new BigDecimal(employeeID));
    rset = pstmt.executeQuery();
} catch (SQLException e) {
```

- Use the employee id value entered as parameter 1 in the prepared statement
- Execute the query stored in the pstmt instance field that now includes the parameter value
- Import the java.math.BigDecimal library at the top

PreparedStatement: Retrieving Data

11. Add the following if statement that get the String values from the query results and displays them in the label of the JavaFX interface:

```
            rset = pstmt.executeQuery();
            if (rset.next()) {
                String email = rset.getString(1);
                String firstName = rset.getString(2);
                String lastName = rset.getString(3);
                lblResult.setText(" Employee name is " + firstName
                                  + " " + lastName
                                  + ", email is " + email);
            } //endif
        } catch (SQLException e) {
```

PreparedStatement: Retrieving Data

12. Add an else statement that displays an error message to screen if the employee id is not found:

```
                        +", email is " + email);
        } else {
              lblResult.setText("Try again !No Employee "
                              + "information for the ID "
                              + employeeID);
        }//endif
    } catch (SQLException e) {
```

PreparedStatement: Retrieving Data

- Now that the functionality of connecting to the database and displaying the results is in place it is time to code the JavaFX interface

13. In the start method delete the content of the try statement. Add a call to the initializeDatabase() method to create the connection at program start

```
@Override
public void start(Stage primaryStage) {
   try {
      initializeDatabase();
   } catch(Exception e) {
      e.printStackTrace();
   }//end try catch
}//end method start
```

PreparedStatement: Retrieving Data

14. Add a button object that displays "**Show Employee details**"
15. Add an actionlistener using a lambda expression to call showResults when the button is clicked

```
            initializeDatabase();
            //create a button that will call the showResult method
            Button btShowID = new Button("Show Employee Details");
            btShowID.setOnAction(e -> showResult());
        } catch(Exception e) {
            e.printStackTrace();
        }//end trycatch
    }//end method start
```

PreparedStatement: Retrieving Data

16. Use the horizontal box layout manager to space the interface components 5 spaces apart horizontally
17. Use the `getChildren().addAll()` method to add from left to right a label that displays "**Employee ID:**", the text box for input and the button

```
    btShowID.setOnAction(e -> showResult());
    //create a horizontal box that displays the label, text box
    //and button
    HBox hBox = new HBox(5);
    hBox.getChildren().addAll(new Label(" Employee ID"), tfid,
                                        btShowID);
} catch(Exception e) {
```

PreparedStatement: Retrieving Data

18. Use the vertical box layout manager to space the interface components 10 spaces apart vertically
19. Use the `getChildren().addAll()` method to add from top to bottom hBox components and then the result label

```
  hBox.getChildren().addAll(new Label(" Employee ID"), tfid,
                                        btShowID);
  //create a vertical box that displays all components on the
  //interface
  VBox vBox = new VBox(10);
  vBox.getChildren().addAll(hBox, lblResult);
} catch(Exception e) {
```

PreparedStatement: Retrieving Data

20. Set the width of the text box to have a maximum size of 6 characters
21. Create a scene object that holds the vBox component and has a size of 400 wide and 100 high

```
    vBox.getChildren().addAll(hBox, lblResult);
    //set the width of the text box to be 6 characters wide
    tfid.setPrefColumnCount(6);
    //create a scene object that contains the contents of the
    //vBox
    Scene scene = new Scene(vBox, 400, 100);
} catch(Exception e) {
```

PreparedStatement: Retrieving Data

22. Set the title of the window to “Show Employee Details”
23. Set the scene using the scene object
24. Show the primary stage to make the application visible

```
        Scene scene = new Scene(vBox, 400, 100);
        //set up the primary stage
        primaryStage.setTitle("Show Employee ID");
        primaryStage.setScene(scene);
        primaryStage.show();
    } catch(Exception e) {
```

PreparedStatement: Retrieving Data

25. Check the imported libraries, they should look like this:

```
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javafx.application.Application;
import javafx.stage.Stage;
import oracle.jdbc.pool.OracleDataSource;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
```

Run and test your JavaFX application!

PreparedStatement: Retrieving Data

- Testing the application is important!
- What happens when you do the following?

a) Add a valid Employee ID

b) Add an invalid Employee ID

c) Add alphabetic characters instead of numeric ones in the text field

d) Click the button with no data in the text field

PreparedStatement: Retrieving Data

- Testing Results!

a) Add a valid Employee ID
   - **Displays the employee id to screen**

b) Add an invalid Employee ID
   - **Displays no employee message**

c) Add alphabetic characters instead of numeric ones in the text field
   - **Causes a NumberFormatException**

d) Click the button with no data in the text field
   - **Causes a NumberFormatException**

PreparedStatement: Retrieving Data

26. To deal with the NumberFormatException (Tests 3 and 4) an additional catch statement can be added to the bottom of the showResult() method
27. The catch statement will call a method named displayError() that accepts String values for the title of the window and the message to be displayed

```
        } catch (SQLException e) {
            e.printStackTrace();
        }catch(NumberFormatException e) {
        displayError("ID Error",
                     "No value entered for Employee ID");
    }//end try catch
}//end method showResult
```

PreparedStatement: Retrieving Data

28. Create the following method under the showResults() method

```
}//end method showResult

private void displayError(String title, String message) {
   Alert alert = new Alert(AlertType.INFORMATION);
   alert.setTitle(title);
   alert.setHeaderText(null);
   alert.setContentText(message);
   alert.showAndWait();
}//end method displayError
```

29. The alert object will create an alert box. The title is displayed at the top of the box, the header text specifies if the text is to be displayed in the header of the box and the content is the message

PreparedStatement: Retrieving Data

- Future Recommendations:
  - A useful feature in this application would be the ability to clear the displayed information (label and text field)

30. Create the following clear() method under the showResults() method
    - This will hide the results label and set the text in the field to an empty String

```
}//end method start

private void clear() {
   lblResult.setVisible(false);
   tfid.setText("");
}//end method clear
```

PreparedStatement: Retrieving Data

31. Add a button directly under where you created the btShowID button in the start method
    - The button text should display "**Clear**" and it should call the clear() method when clicked

```
btShowID.setOnAction(e -> showResult());
//create a button that will call the clear method
Button btClearInfo = new Button("Clear");
btClearInfo.setOnAction(e -> clear());
//create a horizontal box that displays the label, text box
```

Remember: Even though you have created a button component it will not be displayed until it is added to a layout manager!

PreparedStatement: Retrieving Data

32. Update the HBox layout manager so that the clear button will be added to the end of the horizontal box on the right

```
//create a horizontal box that displays the label, text box,
search button and clear button
HBox hBox = new HBox(5);
hBox.getChildren().addAll(new Label(" Employee ID:"),tfid,
                                        btShowID, btClearInfo);
//create a vertical box that displays all components on the
//interface
```

PreparedStatement: Retrieving Data

33. Finally you need to update the showResults() method so that the label will be made visible after the text has been updated

```
   lblResult.setText(" Employee name is " + firstName
                   + " " + lastName
                    +", email is " + email);
 } else {
   lblResult.setText("Try again !No Employee information for the
                          ID "+ employeeID);
 }//endif
 lblResult.setVisible(true);
} catch (SQLException e) {
```

Run and test your code, it should now display an error message and have a clear facility!

PreparedStatement: DML Statement

- The PreparedStatement interface is an extended Statement interface
  - We would use the Statement interface to execute static SQL statements that do not contain any parameters

```
SELECT * FROM employees WHERE employee_id = 100;
```

  - We can use the PreparedStatement interface to execute a precompiled SQL statement with or without parameters

```
SELECT * FROM employees WHERE employee_id = 100;
SELECT * FROM employees WHERE employee_id = ?;
```

PreparedStatement: DML Statement

- A PreparedStatement object is created using the prepareStatement method in the Connection interface.
  - For example, the following code creates a PreparedStatement for a SQL
  - delete statement:

```
DELETE FROM employees WHERE employee_id = ?
```

  - select statement:

```
SELECT job_title FROM jobs WHERE job_id = ? ORDER BY ?
```

  - insert statement:

```
INSERT INTO regions(region_id, region_name) VALUES (?,?)
```

PreparedStatement: SQL Insert

- The Insert statement has two question marks which work as the placeholders for the parameters region_id and region_name in the regions table

```
INSERT INTO regions(region_id, region_name)
VALUES (?,?)
```

- Use the setter methods setX(setShort, setString etc.) for setting IN parameter values

```
setX(int columnIndex, X value);
```

  - Where the columnIndex is the index of the parameter in the statement and X is a parameter value of the correct type

PreparedStatement : SQL Insert

- Excute the PrepareStatement by selecting the most appropriate execute statement:
  - `prepstmt.execute();`
  - `prepstmt.executeUpdate();`
  - `prepstmt.executeBatch();`

PreparedStatement: Update Data

- The following example uses the prepared statement to pass parameters
- After setting the parameters, you can execute the prepared statement by invoking the execute() or the executeUpdate() method
- In the example, the program passes the salary and employee_id parameters to the update sql statement
- The sql statements will be compiled before they are sent to the database

PreparedStatement: Update Data

- This exercise uses the Oracle database that was installed and configured in the previous section
- This exercise will update the existing employee JavaFX application so that you can view the employees current salary as well as having the option to modify the salary

1. Open the RetrieveData class in Eclipse
2. Update the main class file to be named ManageData

PreparedStatement: Update Data

3. Add an additional TextField instance field, this will be used for the input of the modified salary value
4. Add a new label reference (it will be instantiated later)

```
public class ManageData extends Application {
    private Connection conn;
    private Statement stmt;
    private PreparedStatement pstmt;
    private ResultSet rset;
    private TextField tfid = new TextField();
    private TextField tfSalary = new TextField();
    private Label lblResult = new Label();
    private Label lblSalary;
```

PreparedStatement: Update Data

5. Update the initializeDatabase() method so that the query also returns the salary value

```
String query = "SELECT email, first_name, last_name, salary "
             +"FROM Employees "
             +"where employee_id = ?";
```

6. Update the showResults() method so that it can display the new salary value in the label

```
String lastName = rset.getString(3);
int empSalary = rset.getInt(4);
lblResult.setText(" Employee name is " + firstName
                + " " + lastName
                + ", email is " + email
                + ", salary: $" + empSalary);
```

PreparedStatement: Update Data

7. Add a button directly under where you created the btClearInfo button in the start method.
   - The button text should display "**Update Salary**" and it should call the updateSalary() method when clicked.
8. Instantiate the lblSalary label with a value of "**New Salary**"

```
btClearInfo.setOnAction(e -> clear());
//create a button that will call the updateSalary method
Button btModSalary = new Button("Update Salary");
btModSalary.setOnAction(e -> updateSalary());
btModSalary.setVisible(false);
lblSalary = new Label("New Salary");
```

PreparedStatement: Update Data

9. Add a new method named updateVisibility() directly under start() that takes a single Boolean parameter
10. The parameter value will be used to set the visibility of some of the components

```
private void updateVisibility(boolean val) {
   lblSalary.setVisible(val);
   tfSalary.setVisible(val);
   btModSalary.setVisible(val);
}//end method updateVisibility
```

When you try to set the visiblilty of the button it will not allow you to do it here. This is because the btModSalary button was declared locally in start()

PreparedStatement: Update Data

11. Create a reference for the button as an instance field and update the code in start so that it does not create a second instance of the button!
    - Instance field

```
private Label lblSalary;
private Button btModSalary;
```

    - Declaration

```
//create a button that will call the updateSalary method
btModSalary = new Button("Update Salary");
```

    - Update method – name is now shown in blue as it is recognized as an instance field

```
    btModSalary.setVisible(val);
}//end method updateVisibility
```

PreparedStatement: Update Data

12. Under the code that sets up the btModSalary button and the label add a method call to the updateVisibility() method
    - The components are being set to not visible as they will be displayed on successful entry of a valid employee id

```
//create a button that will call the updateSalary method
btModSalary = new Button("Update Salary");
btModSalary.setOnAction(e -> updateSalary());
lblSalary = new Label("New Salary");
updateVisibility(false);
```

PreparedStatement: Update Data

13. Add a new Horizontol box layout manager under the existing one
    - It will contain the salary label
    - The text box to allow the input of the modified salary
    - The button to allow the modification to take place

```
hBox.getChildren().addAll(new Label(" Employee ID:"), tfid,
                                        btShowID, btClearInfo);
//create an hbox that displays a label, text box and modify
button
HBox hBox2 = new HBox(5);
hBox2.getChildren().addAll(lblSalary, tfSalary, btModSalary);
```

PreparedStatement: Update Data

14. Update the vBox layout manager to include the hBox2 layout as a third component
15. Set the preferred width of the salary box to match that of the employee id box

```
//create a vertical box that displays all components on the
//interface
VBox vBox = new VBox(10);
vBox.getChildren().addAll(hBox, lblResult, hBox2);
//set the width of the text box to be 6 characters wide
tfid.setPrefColumnCount(6);
tfSalary.setPrefColumnCount(tfid.getPrefColumnCount());
```

PreparedStatement: Update Data

16. The components in the updateVisibility() method should be displayed with the result label when a valid employee id is entered
    - Add the results label to the updateVisibilty() method

```
private void updateVisibility(boolean val) {
   lblSalary.setVisible(val);
   tfSalary.setVisible(val);
   btModSalary.setVisible(val);
   lblResult.setVisible(val);
}//end method updateVisibility
```

PreparedStatement: Update Data

17. When the clear() method is called it now has to work with multiple components
    - Remove the visible label statement from the clear() method and replace it with a call to updateVisibility()
    - Set the tfSalary text box to have an empty String

```
private void clear() {
    updateVisibility(false);
    tfSalary.setText("");
    tfid.setText("");
}//end method clear
```

PreparedStatement: Update Data

18. In the showResults() method find the line that shows the results label and delete it
19. There are now two separate outcomes based on the employee id entered
    - If it **is valid**, the results label and the modify components should be displayed
    - If it is **not valid** then only the results label should be displayed

PreparedStatement: Update Data

20. To achieve this the updateVisibilty() method should be called from the if statement
21. The results label alone should be shown from the else statement

```
     lblResult.setText(" Employee name is " + firstName
                     + " " + lastName
                     + ", email is " + email
                     + ", salary: $" + empSalary);
     updateVisibility(true);
  } else {
    lblResult.setText("Try again! No Employee information for
                                          the ID " + employeeID);
       lblResult.setVisible(true);
    }//endif
```

PreparedStatement: Update Data

22. The final stage in this exercise is to create a method that will allow the fields in the database to be updated
    - Create the following method under start()

```
private void updateSalary() {
   try{

   }catch (NumberFormatException e){
      displayError("Input Error", "Incorrect value entered");
   } catch (SQLException e) {
      e.printStackTrace();
   }//end try catch
}//end method updateSalary
```

PreparedStatement: Update Data

23. In the try statement:
    - Create 2 local int variables to store the values from the text boxes as numbers
    - Create a prepared statement object that includes the SQL update statement
    - Set the value of the parameters for salary and id
    - Execute the prepared statement to carry out the update statement
    - Close the prepared statement

The completed code is on the next slide….

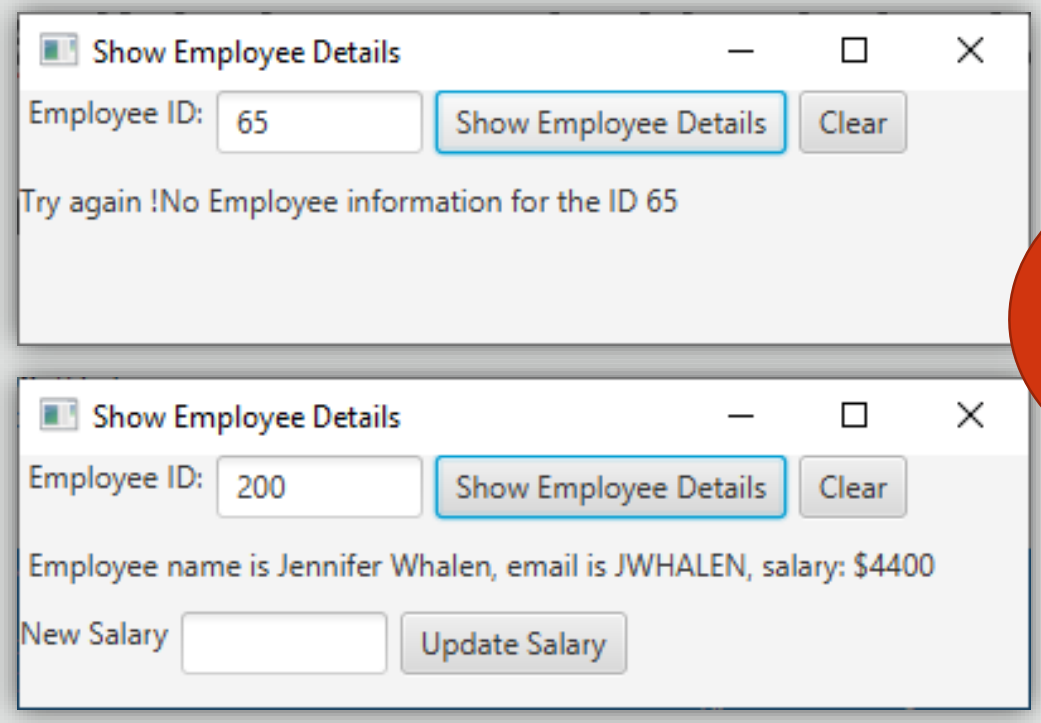
PreparedStatement: Update Data

- Your completed should look like this:

```
private void updateSalary() {
   try{
      int id = Integer.parseInt(tfid.getText());
      int salary=Integer.parseInt(tfSalary.getText());
      PreparedStatement prepstmt = conn.prepareStatement(
            "UPDATE employees SET salary = ? WHERE employee_id =?");
      prepstmt.setInt(1,salary);
      prepstmt.setInt(2,id);
      prepstmt.execute();
      prepstmt.close();
   }catch (NumberFormatException e){
      displayError("Input Error", "Incorrect value entered");
   } catch (SQLException e) {
      e.printStackTrace();
   }//end try catch
}//end method updateSalary
```

PreparedStatement: Update Data

24. Run and test your application



What happens if you provide a negative salary?

PreparedStatement: Update Data

- The Oracle_Schema_HR.sql script included a check constraint that enforces a database rule that the salary must be greater than zero

```
CONSTRAINT "EMP_SALARY_MIN" CHECK
(salary > 0) ENABLE,
```

- If a check constraint is violated then a SQLIntegrityConstraintViolationException is thrown

PreparedStatement: Update Data

25. Add another catch statement (above the SQLException one as that catches all SQL errors)
    - Handle a SQLIntegrityConstraintViolationException
    - It will call the displayError() method
    - It will clear the new salary text box

What happens if you provide a negative salary now?

```
    displayError("Input Error", "Incorrect value entered");
}catch(SQLIntegrityConstraintViolationException e) {
    displayError("Salary Error",
                 "Negative value for salary entered");
    tfSalary.setText("");
}catch (SQLException e) {
```

Oracle Stored Procedures

- Callable statements are statements that are stored within the database and can be called from the JDBC application
  - The CallableStatement interface is designed to execute SQL stored procedures
  - The Oracle JDBC Drivers support Java and PL/SQL stored procedures
  - The procedures may have IN, OUT or IN/OUT parameters

Oracle Stored Procedures

- CallableStatement Object
  - An IN parameter receives a value passed to the procedure when it is called
  - An OUT parameter returns a value after the procedure is completed, but it doesn't contain any value when the procedure is called
  - An IN/OUT parameter contains a value passed to the procedure when it is called and returns a value after it is completed

Oracle Stored Procedures

- As with PreparedStatement objects, input parameters must be set with the setTYPE method
  - Where TYPE is the appropriate data type for the parameter
- Output parameters are defined with the registerOutParameter method of a CallableStatement object
  - They must be registered before the stored procedure can be executed

Oracle Stored Procedures

- CallableStatement Object:
  - A JDBC CallableStatement object is created by invoking the callableStatement method with the name of the procedure and question marks (“?”) to represent the input and output parameters
  - The CallableStatement interface is designed to execute SQL-stored procedures
  - The result can be retrieved via get methods

Oracle Stored Procedure Example

1. Open APEX(Cloud) or SQLPlus(local) and log in using the database user account
2. Add the following code to create the stored procedure in your database
   - The forward slash ends the statement

```
CREATE OR REPLACE PROCEDURE getEmployeeSalary(emp_id IN NUMBER,  sal OUT
NUMBER, l_name OUT VARCHAR2)
AS
BEGIN
SELECT last_name, salary  INTO l_name, sal
FROM employees
WHERE employee_id = emp_id;
END getEmployeeSalary;
/
```

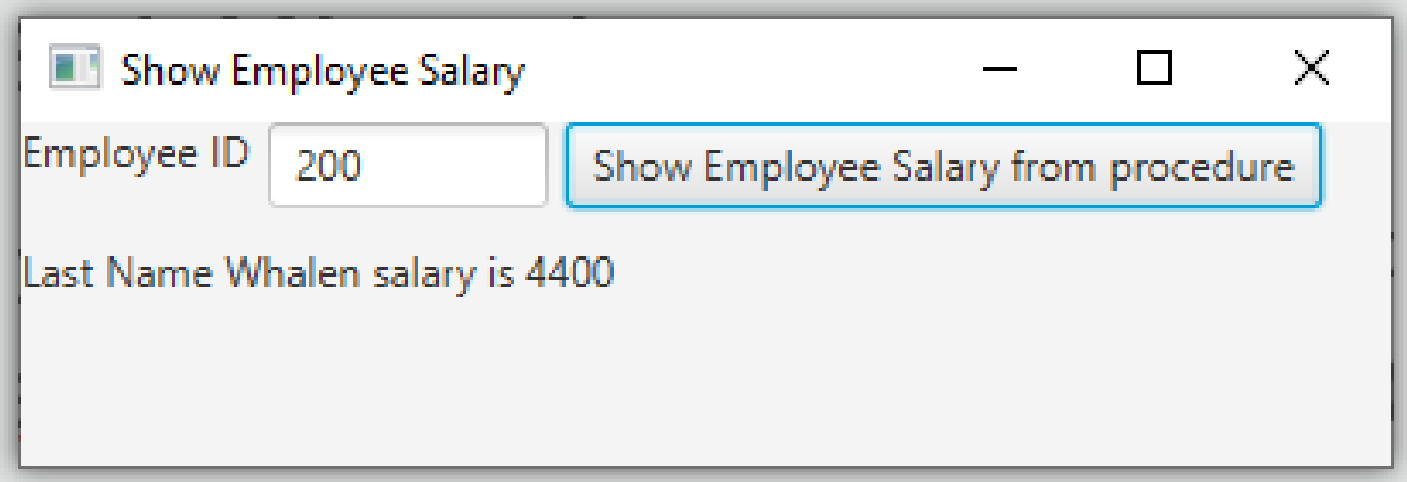
There is an explanation of this on the next slide..

Oracle Stored Procedure Explanation

- **CREATE OR REPLACE PROCEDURE**:
- creates a new procedure or overwrites an existing one with the same name
- **getEmployeeSalary**: the name of the stored procedure
- **()**: The parameter list identifies the name status (IN, OUT, IN/OUT and datatype of the parameters
- **AS**: returns the result of the following static SQL statement to the parameter list
- **BEGIN** and **END**: encloses the PL/SQL statement to be executed

CallableStatement Object

- You will create the following JavaFX example using the Oracle PL/SQL stored procedure (getEmployeeSalary(emp_id IN NUMBER, sal OUT NUMBER, l_name OUT VARCHAR2)) that you previously added to your database



Oracle Stored Procedure Example

1. Open Eclipse and setup and configure the FX plugin
2. Create a JavaFX Project named storedproceduresexample
3. Update the main class file to be named CallableEmployee
4. You can remove the stylesheets line as you won't be using them in this exercise
5. Add the Oracle JDBC jar file to your build path

Oracle Stored Procedure Example

6. Add the following Instance fields to your CallableEmployee class

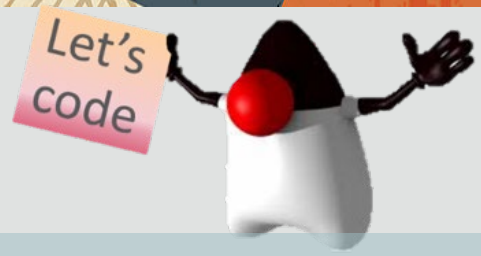
```
import java.sql.CallableStatement;
import java.sql.Connection;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;

public class CallableEmployee extends Application {
    private Connection conn;
    private CallableStatement cstmt;
    private TextField tfid = new TextField();
    private Label lblResult = new Label();
```

Database Type

7. Add the initialiseDatabase() method to create the connection to the database

- If you are using the Always Free Cloud instance, then use the instructions on the next slide (75) to create the connection to the database
- If you are using a local database (Oracle XE) then use slide 76 to create the connection to the database



Oracle Stored Procedure Example

```
 public void initializeDatabase() {
    final String DB_URL=
"jdbc:oracle:thin:@db202102091440_medium?TNS_ADMIN=D:/cloud_conn/wallet_DB2021
02091440/";
    final String DB_USER = "user_01";
    final String DB_PASSWORD = "UserPassword1";
  try {
    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);
    conn = ods.getConnection();
  }catch (Exception e) {
     e.printStackTrace();
  }//end try catch
}//end method initializeDatabase
```

Oracle Stored Procedure Example


```
public void initializeDatabase() {
   String user = "orcluser";
   String password = "jdbcuser";
   try {
      OracleDataSource ods = new OracleDataSource();
      ods.setURL("jdbc:oracle:thin:" + user + "/"
                + password + "@localhost:1521/xepdb1");
      conn = ods.getConnection();
   } catch (Exception e) {
      e.printStackTrace();
   }//end try catch
}//end method initializeDatabase
```

Oracle Stored Procedure Example

8. Add showResults() to pull the information from the stored procedure in the database
9. Set the result label to empty at the start of the method and display the error in it from the catch statements

```
private void showResult() {
   lblResult.setText("");
   try {

   }catch (SQLException e) {
      lblResult.setText("wrong Employee id, please try again!");
   }catch(NumberFormatException e){
      lblResult.setText("wrong Employee id, please try again!");
   }//end try catch
}//end method sowResult
```

Oracle Stored Procedure Example

10. Add the following String variable named spQuery (sp -stored procedure) that includes the call to the stored procedure in the database and uses the ? Mark placeholders for the parameter list
11. Within the try statement store the employee id value from the id text box in an integer id variable

```
private void showResult() {
    lblResult.setText("");
    String spQuery = "{call getEmployeeSalary(?,?,?)}";
    try {
        int id = Integer.parseInt(tfid.getText());
```

Oracle Stored Procedure Example

12. Set the callable statement (cstmt) to hold the connection information and the call to the stored procedure
13. Set the first parameter (IN) to hold the value of the employee id field
14. Register the two OUT parameters before executing the callable statement

```
int id = Integer.parseInt(tfid.getText());
cstmt = conn.prepareCall(spQuery);
cstmt.setInt(1, id);
cstmt.registerOutParameter(2, Types.DOUBLE);
cstmt.registerOutParameter(3, Types.VARCHAR);
cstmt.execute();
```

Oracle Stored Procedure Example

15. Set the salary to the value returned by parameter number 2 of the callable statement (cstmt)
16. Set the salary to the value returned by parameter number 2 of the callable statement (cstmt)
17. Display the results in the results label

```
        cstmt.execute();
        String salary = cstmt.getBigDecimal(2).toString();
        String lastName = cstmt.getString(3);
        lblResult.setText("Last Name " + lastName + " salary
                          is " + salary);
    }catch (SQLException e) {
```

Oracle Stored Procedure Example

18. Create the interface in the try statement in start():

```
public void start(Stage primaryStage) {
   try {
      initializeDatabase();
      Button btShowID = new Button("Show Employee Salary from procedure");
      HBox hBox = new HBox(5);
      hBox.getChildren().addAll(new Label("Employee ID"), tfid, btShowID);
      VBox vBox = new VBox(10);
      vBox.getChildren().addAll(hBox, lblResult);
      tfid.setPrefColumnCount(6);
      btShowID.setOnAction(e -> showResult());
      Scene scene = new Scene(vBox, 400, 100);
      primaryStage.setTitle("Show Employee Salary");
      primaryStage.setScene(scene);
      primaryStage.show();
   } catch(Exception e) {
```

Run and test your code!

Reading Metadata

- In some cases, the application may require to read the metadata
- For example, the DatabaseMetaData and ResultSetMetaData
- JDBC provides the DatabaseMetaData interface for retrieving comprehensive information about the database as a whole

Reading Metadata

- DatabaseMetaData is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the driver based on JDBC™ technology ("JDBC driver") that is used with it
- The database metadata such as username, table name and database URL can be accessed by the DatabaseMetaData interface

Database MetaData

- The Connection object provides access to database metadata information that describes the capabilities of the Oracle Database
- To get an instance of the DatabaseMetaData for a database, use the getMetaData method defined on a Connection Object

```
DatabaseMetaData dbmd = conn.getMetaData();
```

- Requires the import of:

```
import java.sql.DatabaseMetaData;
```

Database MetaData

- You can invoke the methods defined in DatabaseMetaData by using the get methods
- getDriverName()
  - Retrieves the name of this JDBC driver
- getSchemas()
  - Retrieves the schema names available in this database
- getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
  - Retrieves a description of the tables available in the given catalog

Database MetaData Example

1. Add the displayDatabaseDetails () method to the bottom of the CallableEmployee class

```
public void displayDatabaseDetails() throws SQLException {
   DatabaseMetaData dbmd = conn.getMetaData();
   System.out.println("Database Version: "
                      + dbmd.getDatabaseProductVersion());
   System.out.println("Driver name    : "
                      + dbmd.getDriverName());
   System.out.println("URL            : " + dbmd.getURL());
   System.out.println("Username       : "
                      + dbmd.getUserName());
   ResultSet rsTables = dbmd.getTables(null, null, "EMP%",
                                       new String[] {"TABLE"});
   while (rsTables.next()) {
      System.out.println(rsTables.getString("TABLE_NAME"));
   }//end while
}//end method displayDatabaseDetails
```

Consult the API for other get methods.

Database MetaData Example

2. Add a method call to displayDatabaseDetails() in start() after the database has been initialized

```
public void start(Stage primaryStage) {
   try {
      initializeDatabase();
      displayDatabaseDetails();
```

3. This should produce the following console output for a local database instance:

```
CallableEmployee [Java Application] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe
Databse Version: Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0
Driver name     : Oracle JDBC driver
URL             : jdbc:oracle:thin:orcluser/jdbcuser@localhost:1521/xepdb1
Username        : ORCLUSER
EMPLOYEES
EMPLOYEES
```

Result MetaData

- The ResultMetaData object can be used to get information about the types and properties of the columns in a ResultSet object
- An instance of the ResultMetaData is created from the running of query on a database

```
ResultSet rs = stmt.executeQuery(
        "SELECT job_id, job_title FROM jobs");
```

- Requires the import of:

```
import java.sql.ResultSet;
```

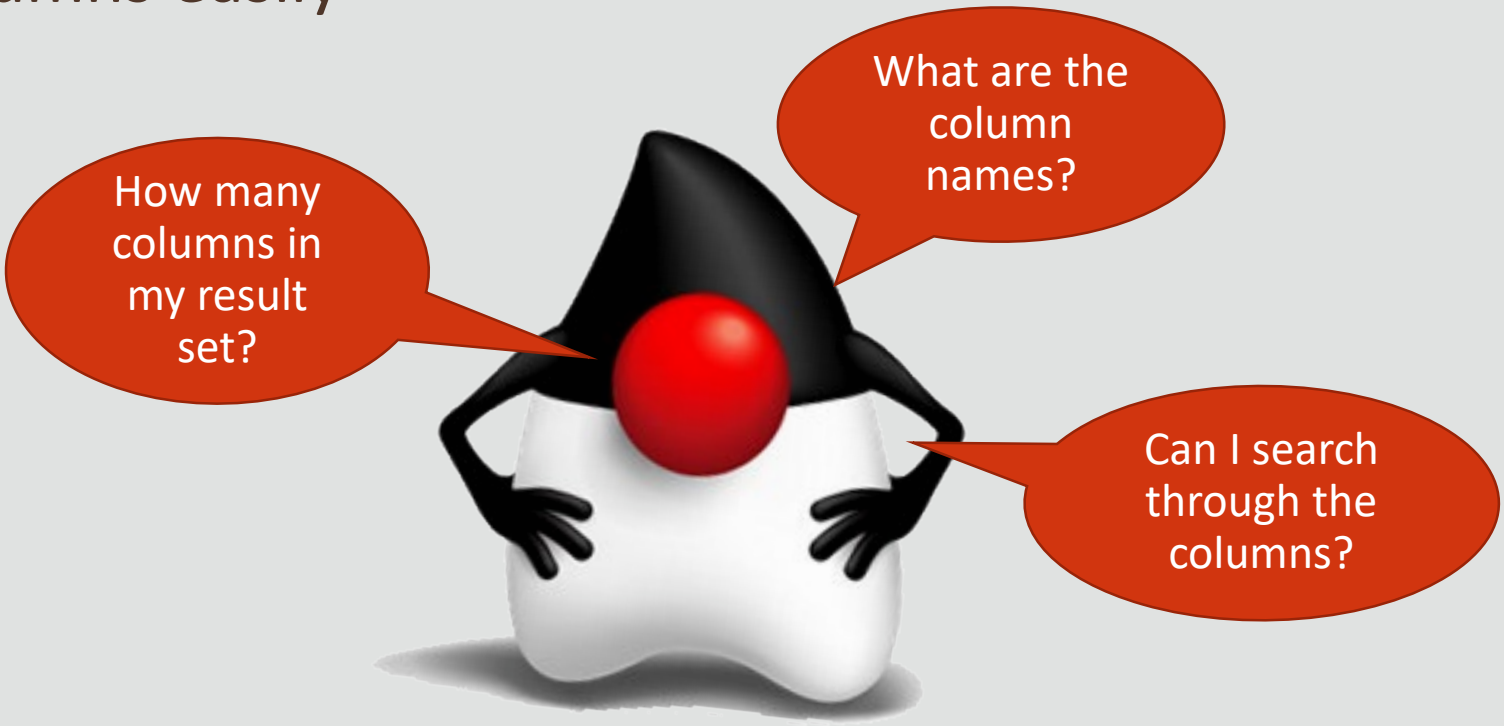
Result MetaData

- The following code fragment:
  - creates the ResultSet object rs
  - creates the ResultSetMetaData object rsmd
  - uses rsmd to find out how many columns rs has, and whether the first column in rs can be used in a WHERE clause

```
ResultSet rs = stmt.executeQuery(
                        "SELECT job_id, job_title FROM jobs");
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
boolean b = rsmd.isSearchable(1);
```

Result MetaData

- The ResultMetaData object can be used to help you create code that works with any number of returned columns easily



Reading Metadata

- You have already used the ResultSetMetaData object to get information about the types and properties of the columns in the resultSet of a query

```
public static int getColumnNames(ResultSet rs) throws SQLException {
   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
```

Reading Metadata

- In the previous example you used the getColoumn Count() method to return the number of columns in the current result set
- This allows the code to always work despite varying numbers of columns being returned

```
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)+ ", ");
      //endif
  }//endfor
}//endwhile
```

Summary

- In this lesson, you should have learned:
  - JDBC Data Types
  - Programming with JDBC PreparedStatement
  - Programming with JDBC CallableStatement
  - Reading MetaData from Database


ORACLE
Academy