# JDBC

The slides adapted from different resources including:
Hitha Paulson, Assistant Professor, Dept. of Computer Science LF College, Guruvayoor
MET CS 667: 9. JDBC, Servlets, JSP Zlateva
Sanjay Goel

# Reading from the Properties File

```
Properties props = new Properties();
String fileName = "QueryDB.properties";
FileInputStream in = new FileInputStream(fileName);
props.load(in);
in.close();
String drivers = props.getProperty("jdbc.drivers");
if (drivers != null)
        System.setProperty("jdbc.drivers", drivers);
String url = props.getProperty("jdbc.url");
String username = props.getProperty("jdbc.username");
String password = props.getProperty("jdbc.password");
```

# Statement - Types

- Statements in JDBC abstract the SQL statements
- Primary interface to the tables in the database
- Used to create, retrieve, update & delete data (CRUD) from a table
  - Syntax: Statement statement = connection.createStatement();
- Three types of statements each reflecting a specific SQL statements
  - Statement
  - PreparedStatement allows to write SQL queries with placeholders for input parameters (using ? symbols), which will later be replaced by actual values.
    - This eliminates the need to concatenate strings when including values in SQL queries, which is important for security reasons, particularly to prevent SQL injection attacks.
  - CallableStatement used for executing **stored procedures** or **functions**

# Statement - Syntax

- Statement used to send SQL commands to the database
  - Case 1: ResultSet is non-scrollable and non-updateable

```
public Statement createStatement() throws SQLException
Statement statement = connection.createStatement();
```

  - Case 2: ResultSet is non-scrollable and/or non-updateable

```
public Statement createStatement(int, int) throws
    SQLException
Statement statement = connection.createStatement();
```

  - Case 3: ResultSet is non-scrollable and/or non-updateable and/or holdable

```
public Statement createStatement(int, int, int) throws
    SQLException
Statement statement = connection.createStatement();
```

- PreparedStatement

```
public PreparedStatement prepareStatement(String sql) throws
    SQLException
PreparedStatement pstatement = prepareStatement(sqlString);
```

- CallableStatement used to call stored procedures

```
public CallableStatement prepareCall(String sql) throws
    SQLException
```

# Scrollable ResultSet

- ResultSet obtained from the statement created using the no argument constructor is:
  - Type forward only (non-scrollable)
  - Not updateable
- To create a scrollable ResultSet the following statement constructor is required
  - `Statement createStatement(int resultSetType, int resultSetConcurrency)`
- ResultSetType determines whether it is scrollable. It can have the following values:
  - `ResultSet.TYPE_FORWARD_ONLY`
  - `ResultSet.TYPE_SCROLL_INSENSITIVE` (Unaffected by changes to underlying database)
  - `ResultSet.TYPE_SCROLL_SENSITIVE` (Reflects changes to underlying database)
  - resultSetHoldability - one of the following ResultSet constants: ResultSet.HOLD_CURSORS_OVER_COMMIT or ResultSet.CLOSE_CURSORS_AT_COMMIT
- ResultSetConcurrency determines whether data is updateable. Its possible values are
  - `CONCUR_READ_ONLY`
  - `CONCUR_UPDATEABLE`
- Not all database drivers may support these functionalities

# Scrollable ResultSet

- TYPE_FORWARD_ONLY : The constant indicating the type for a ResultSet object whose cursor may move only forward.
- TYPE_SCROLL_INSENSITIVE : The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes to the data that underlies the ResultSet.
- TYPE_SCROLL_SENSITIVE : The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes to the data that underlies the ResultSet.
- CONCUR_READ_ONLY : The constant indicating the concurrency mode for a ResultSet object that may NOT be updated.
- CONCUR_UPDATABLE : The constant indicating the concurrency mode for a ResultSet object that may be updated.
- CLOSE_CURSORS_AT_COMMIT : The constant indicating that open ResultSet objects with this holdability will be closed when the current transaction is committed.
- HOLD_CURSORS_OVER_COMMIT : The constant indicating that open ResultSet objects with this holdability will remain open when the current transaction is committed.

# Scrollable ResultSet

- absolute(int row): Moves the cursor to the given row number in this ResultSet object.
- relative(int rows): Moves the cursor a relative number of rows, either positive or negative.
- afterLast(): Moves the cursor to the end of this ResultSet object, just after the last row.
- beforeFirst(): Moves the cursor to the front of this ResultSet object, just before the first row.
- first(): Moves the cursor to the first row in this ResultSet object.
- last(): Moves the cursor to the last row in this ResultSet object.
- isAfterLast(): Retrieves whether the cursor is after the last row in this ResultSet object.
- isBeforeFirst(): Retrieves whether the cursor is before the first row in this ResultSet object.
- isClosed(): Retrieves whether this ResultSet object has been closed.
- isFirst(): Retrieves whether the cursor is on the first row of this ResultSet object.
- isLast(): Retrieves whether the cursor is on the last row of this ResultSet object.
- findColumn(String columnLabel): Maps the given ResultSet column label to its ResultSet column index.
- moveToCurrentRow(): Moves the cursor to the remembered cursor position, usually the current row.
- next(): Moves the cursor forward one row from its current position.
- previous(): Moves the cursor to the previous row in this ResultSet object.

# JDBC 2 – Scrollable Result Set

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

String query = “select students from class where type=‘not sleeping’ “;
ResultSet rs = stmt.executeQuery( query );

rs.previous();  // go back in the RS
rs.relative(-5); // go 5 records back
rs.relative(7); // go 7 records forward
rs.absolute(100); // go to 100th record
…
```

# Updateable ResultSet

- moveToInsertRow(): Moves the cursor to the insert row.
  - The insert row is a special row associated with an updatable result set. It is essentially a buffer where a new row may be constructed by calling the updater methods prior to inserting the row into the result set.
- refreshRow(): Refreshes the current row with its most recent value in the database
- getStatement(): Retrieves the Statement object that produced this ResultSet object.
- rowDeleted(): Retrieves whether a row has been deleted.
- rowInserted(): Retrieves whether the current row has had an insertion.
- rowUpdated(): Retrieves whether the current row has been updated.
- cancelRowUpdates(): Cancels the updates made to the current row in this ResultSet object.
- deleteRow(): Deletes the current row from this ResultSet object and from the underlying database.
- insertRow(): Inserts the contents of the insert row into this ResultSet object and into the database
- updateRow(): Updates the underlying database with the new contents of the current row of this ResultSet object.
- updateRowId(int columnIndex, RowId x): Updates the designated column with a RowId value.
- updateRowId(String columnLabel, RowId x): Updates the designated column with a RowId value.
- updateDouble(String columnLabel, double x): Updates the designated column with a double value.

# JDBC 2 – Updateable ResultSet

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                    ResultSet.CONCUR_UPDATABLE);
String query = " select students, grade from class
             where type='really listening this presentation☺' ";
ResultSet rs = stmt.executeQuery( query );
…
while ( rs.next() )
{
  int grade = rs.getInt("grade");
  rs.updateInt("grade", grade+10);
  rs.updateRow();
}
```

# MetaData

- Meta Data means data about data.
- Retrieves a DatabaseMetaData object that contains metadata about the database to which this Connection object represents a connection.
- Two kinds of meta data in JDBC
  - Database Metadata: To look up information about the database (here)
  - ResultSet Metadata: To get the structure of data that is returned (later)
- Example
  - `connection.getMetaData().getDatabaseProductName()`
  - `connection.getMetaData().getDatabaseProductVersion()`
- Sample Code:

```
private void showInfo(String driver,String url,String user,String
password,
  String table,PrintWriter out) {
Class.forName(driver);
   Conntection con = DriverManager.getConnection(url, username, password);
   DatabaseMetaData dbMetaData = connection.getMetaData();
   String productName = dbMetaData.getDatabaseProductName();
   System.out.println("Database: " + productName);
   String productVersion = dbMetaData.getDatabaseProductVersion();
   System.out.println("Version: " + productVersion);
   }
```

# MetaData

- getDatabaseProductName(): Retrieves the name of this database product.
- getDatabaseProductVersion(): Retrieves the version number of this database product.
- getDriverName(): Retrieves the name of this JDBC driver.
- getPrimaryKeys(String catalog, String schema, String table): Retrieves a description of the given table's primary key columns.
- getProcedures(String catalog, String schemaPattern, String procedureNamePattern): Retrieves a description of the stored procedures available in the given catalog.
- 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.

# Metadata from DB - example

```
…
Connection con = …. ;

DatabaseMetaData dbmd = con.getMetaData();

String catalog = null;
String schema = null;
String table = “sys%”;
String[ ] types = null;

ResultSet rs =
  dbmd.getTables(catalog , schema , table , types );
…
```

# Prepared Statement

**Main Idea:** Prepare query in advance with a **host variable,** denoted by **?,** that is filled with a different value every time the query is run.
The host variable must be bound to an actual value of a field, and this is performed by a **set** function similar to the get function discussed earlier.

- PreparedStatement provides a means to create a reusable statement that is precompiled by the database

```
String publisherQuery =
 "SELECT Books.Price, Books.Title " +
 "FROM Books, Publishers " +
 "WHERE Books.Publisher_Id=Publishers.Publisher_Id AND" +
 "Publishers.Name = ?";

PreparedStatement publisherQueryStmt =
   con.prepareStatement(publisherQuery);

publisherQueryStmt.setString(1, publisher);
rs = publisherQueryStmt.executeQuery();
```

1st argument is host variable we want to set, 2d argument value to be assigned to host variable;
position 1denotes first ?

# Prepared Statement

- It has three main uses
  - Create parameterized statements such that data for parameters can be dynamically substituted
  - Create statements where data values may not be character strings
  - Precompiling SQL statements to avoid repeated compiling of the same SQL statement
- If parameters for the query are not set the driver returns an SQL Exception
- Only the no parameters versions of executeUpdate() and executeQuery() allowed with prepared statements.

# Prepared Statement

- Example

```
// Creating a prepared Statement
String sqlString = "UPDATE authors SET lastname = ?
   Authid = ?";
PreparedStatement ps =
   connection.prepareStatement(sqlString);
ps.setString(1, "Allamaraju");     // Sets first
   placeholder to Allamaraju
ps.setString(2, 212);                  // Sets second
   placeholder to 212
ps.executeUpdate();           // Executes the update
```

## authorPublisherQuery

```
String authorPublisherQuery =
  "SELECT Books.Price, Books.Title " +
  "FROM Books, BooksAuthors, Authors, Publishers " +
  "WHERE Authors.Author_Id = BooksAuthors.Author_Id AND "+
  "BooksAuthors.ISBN = Books.ISBN AND " +
  "Books.Publisher_Id = Publishers.Publisher_Id AND " +
  "Authors.Name = ? AND " +
  "Publishers.Name = ?";

authorPublisherQueryStmt =
  con.prepareStatement(authorPublisherQuery);
authorPublisherQueryStmt.setString(1, author);
authorPublisherQueryStmt.setString(2,publisher);
rs = authorPublisherQueryStmt.executeQuery();
```

# Prepared Statement

```
Import java.sql.*;


public class AuthorDatabase {
  public static void main(String[] args) {
    try {
          String url = "jdbc:odbc:library";
          String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
          String user = "goel"
          String password = "password";
          // Load the Driver
          Class.forName(driver);
          Connection connection =
      DriverManager.getConnection();
          String sqlString = "UPDATE authors SET lastname = ?
      Authid = ?";
          PreparedStatement ps =
      connection.prepareStatement(sqlString);
          // Sets first placeholder to Allamaraju
ps.setString(1, "Allamaraju");
          // Sets second placeholder to 212
          ps.setString(2, 212);
          // Executes the update
          int rowsUpdated = ps.executeUpdate();
          System.out.println("Number of rows changed = " +
       rowsUpdated);
          connection.close();
          }
        catch (ClassNotFoundException cnfe) {
          System.out.println("Driver not found");
          cnfe.printStackTrace();
         }
         catch (SQLException sqle) {
          System.out.println("Bad SQL statement");
          sqle.printStackTrace();
         }
  }
 }
```

# Callable Statements & Stored Procedures

- Stored Procedures
  - Are procedures that are stored in a database.
  - Consist of SQL statements as well as procedural language statements
  - May (or may not) take some arguments
  - May (or may not) return some values
- Advantages of Stored Procedures
  - Encapsulation & Reuse
  - Transaction Control
  - Standardization
- Disadvantages
  - Database specific (lose independence)
- Callable statements provide means of using stored procedures in the database

# Callable Statements & Stored Procedures

- Stored Procedures must follow certain rules
  - Names of the stored procedures and parameters must be legal
  - Parameter types must be legal supported by database
  - Each parameter must have one of In, Out or Inout modes
- Example

  // Creating a stored procedure using SQL

  - `CREATE PROC procProductsList AS SELECT * FROM Products;`
  - `CREATE PROC procProductsDeleteItem(inProductsID LONG) AS DELETE FROM Products WHERE ProductsID = inProductsID;“`
  - `CREATE PROC procProductsAddItem(inProductName VARCHAR(40), inSupplierID LONG, inCategoryID LONG) AS INSERT INTO Products (ProductName, SupplierID, CategoryID) Values (inProductName, inSupplierID, inCategoryID);"`
  - `CREATE PROC procProductsUpdateItem(inProductID LONG, inProductName VARCHAR(40)) AS UPDATE Products SET ProductName = inProductName WHERE ProductID = inProductID;"`

  Usage: procProductsUpdateItem(1000, “My Music”)

  (Sets the name of the product with id 1000 to 16.99)

# Callable Statements & Stored Procedures

```
String sql = "{call getEmployeeSalary(?, ?)}"; // SQL calling a stored procedure with parameters

CallableStatement callableStatement = connection.prepareCall(sql);
callableStatement.setInt(1, 101);  // Input parameter: employee ID
callableStatement.registerOutParameter(2, Types.DECIMAL); // Output parameter: salary

callableStatement.execute(); // Execute the stored procedure

double salary = callableStatement.getDouble(2); // Get the output parameter (salary)
System.out.println("Employee Salary: " + salary);

CREATE FUNCTION get_salary(emp_id INT) RETURNS DECIMAL(10, 2)
BEGIN
    DECLARE total DECIMAL(10, 2);
    SELECT salary INTO total FROM employees WHERE id = emp_id;
    RETURN total;
END;

CREATE PROCEDURE get_employee_salary(IN emp_id INT, OUT total_salary DECIMAL(10, 2))
BEGIN
    SELECT salary INTO total_salary FROM employees WHERE id = emp_id;
END;
```