JDBC API

Java Database Connectivity

EHSDI

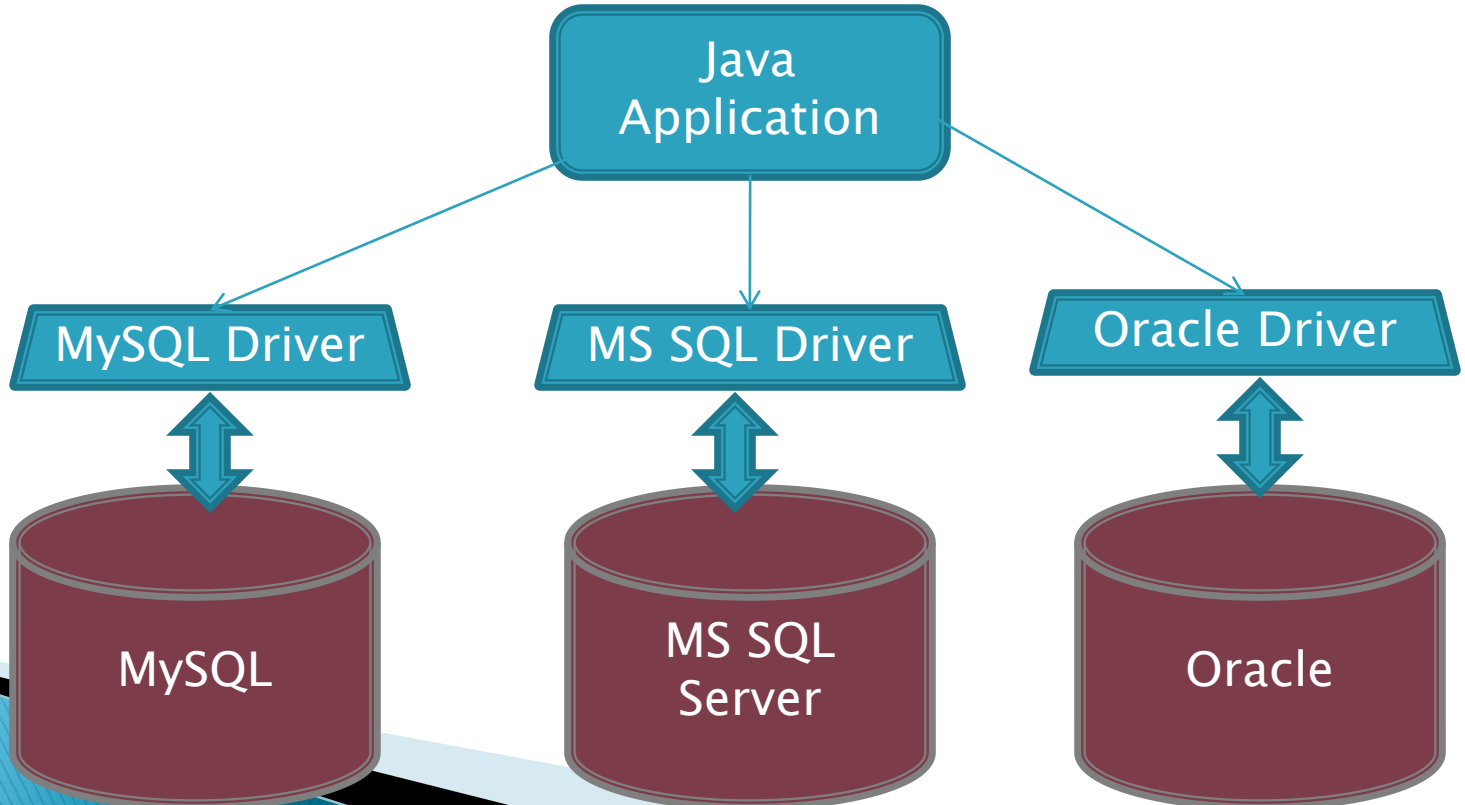
eBuzima

JDBC

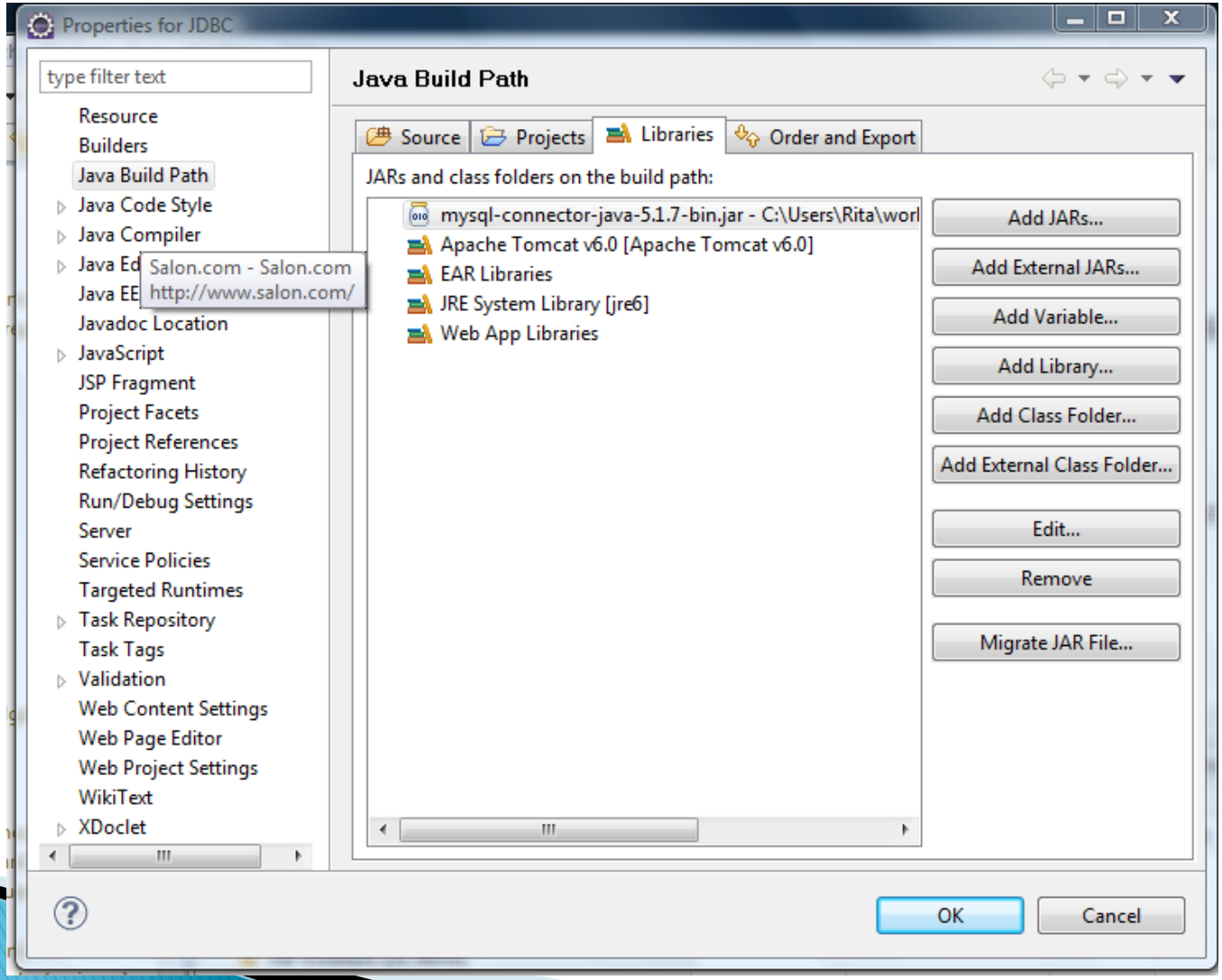
- Load Driver
- Establish Connection
- Retrieving Values
- Updating Tables
- Prepared Statements

Database Drivers

- A bridge between your Java application and the database



Add the Database Driver to Your Build Path



Load the Driver

- Loading a class dynamically by its name

```
/**
 * Dynamically loads the database driver for MySQL
 */
public void loadDatabaseDriver() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

http://cephas.net/blog/2005/07/31/java-classfornamestring-classname-and-jdbc/

Connect to the Database

```
/**
 * Creates a connection to the database.
 * @return Connection the connection object.
 */
public Connection connectToDatabase() {
    String connectionURL = "jdbc:mysql://localhost:3306/weather";
    Connection connection = null;

    try {

        connection = (Connection) DriverManager.getConnection(
                connectionURL, "devuser", "devpass");

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return connection;
}
```

Statements

- The object used for executing a static SQL statement and returning the results it produces.
- By default, only one ResultSet object per Statement object can be open at the same time. Therefore, if the reading of one ResultSet object is interleaved with the reading of another, each must have been generated by different Statement objects.

ResultSet

- ResultSet is an object containing records from the database

- `next()` - moves the cursor forward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned after the last row.
- `previous()` - moves the cursor backwards one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned before the first row.
- `first()` - moves the cursor to the first row in the `ResultSet` object. Returns true if the cursor is now positioned on the first row and false if the `ResultSet` object does not contain any rows.
- `last()` - moves the cursor to the last row in the `ResultSet` object. Returns true if the cursor is now positioned on the last row and false if the `ResultSet` object does not contain any rows.
- `beforeFirst()` - positions the cursor at the start of the `ResultSet` object, before the first row. If the `ResultSet` object does not contain any rows, this method has no effect.
- `afterLast()` - positions the cursor at the end of the `ResultSet` object, after the last row. If the `ResultSet` object does not contain any rows, this method has no effect.
- `relative(int rows)` - moves the cursor relative to its current position.
- `absolute(int row)` - positions the cursor on the row-th row of the `ResultSet` object.

Source: http://java.sun.com/docs/books/tutorial/jdbc/basics/retrieving.html

Query the Database

```
/**
 * Queries the database
 * @param conn
 */
public void runQuery(Connection conn) {
    Statement stmt;
    ResultSet srs;
    try {
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
        srs = stmt.executeQuery("SELECT * from forecast_data");
        srs.first();
        System.out.println(srs.getString(1));
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }


}
```

ResultSet Types

- `TYPE_FORWARD_ONLY (Default)`
  - Can only scroll forward through a result set
- `TYPE_SCROLL_INSENSITIVE`
  - The result set can be scrolled forward or backward for records. Changes to the database are not reflected in the record set
- `TYPE_SCROLL_SENSITIVE`
  - The result set can be scrolled forward or backward for records. Changes to the database ARE reflected in the record set

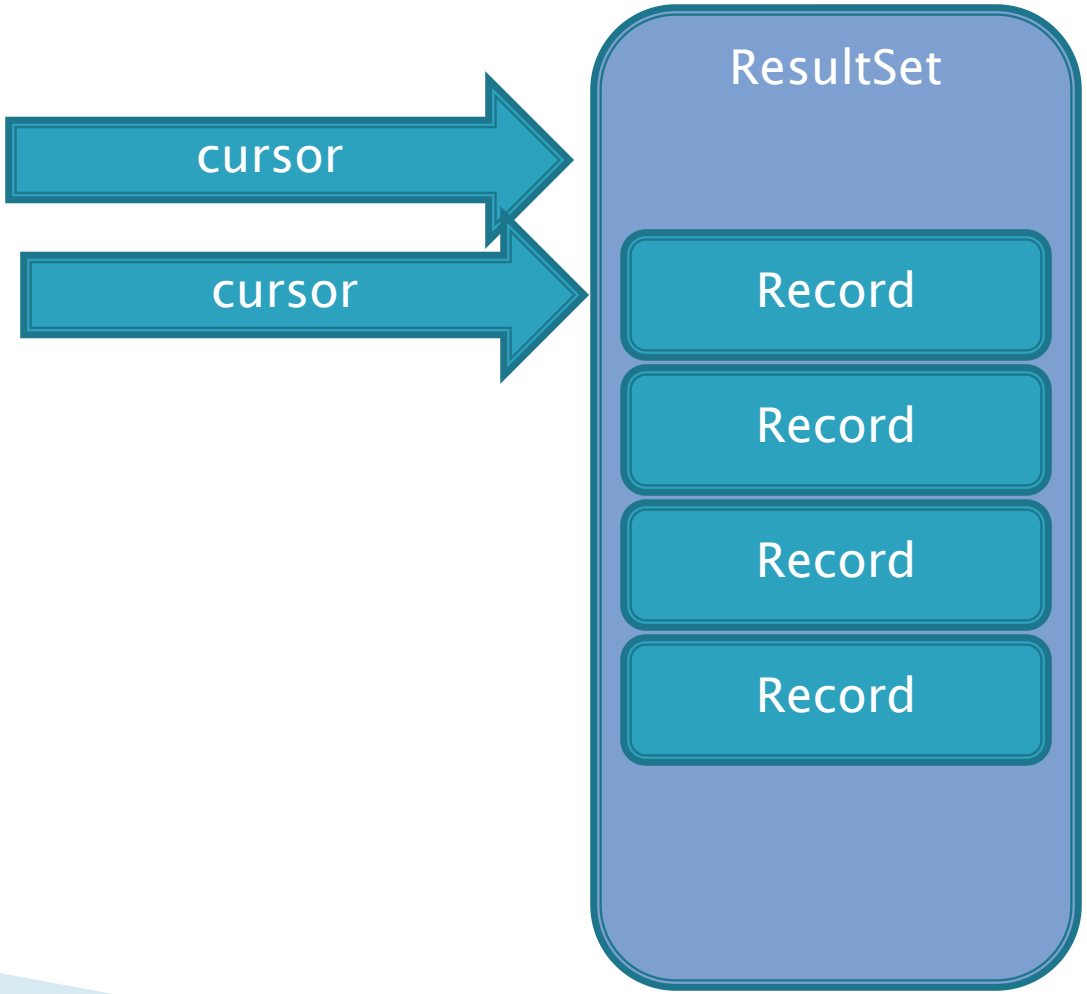
Updatable

- `CONCUR_READ_ONLY (Default)`
  - The `ResultSet` may NOT be updated

- `CONCUR_UPDATABLE`
  - The `ResultSet` may be updated

Cursor is pointing before the first record before a call to `next()`

```
resultSet.next();
```



Getting Values from the ResultSet

- Iterating forward through a result set

```
while (srs.next()) {
    String name = srs.getString("description");
    float id = srs.getInt("id");
    System.out.println(name + "      " + id);
}
```

Getting Values from the ResultSet

- Iterating backwards through a ResultSet

```
srs.afterLast();
while (srs.previous()) {
    String name = srs.getString("description");
    float id = srs.getFloat("id");
    System.out.println(name + "      " + id);
}
```

Getting Values From the ResultSet

- Can get values from the column index or column name

```
srs.afterLast();
while (srs.previous()) {
    String name = srs.getString(1); // the first column starts at 1!!
}
```

- You can use `getString()` for all the basic datatypes and then cast if desired.

Running Update Queries

- Can be run with a default ResultSet (`CONCUR_READ_ONLY`)
- Can be INSERT, UPDATE, or DELETE

```
query = "UPDATE " + tablename + " SET " + fieldname + " = 'Karen' WHERE " + fieldname + " = 'Sarah';";

// modifying data
try {
    stmt.executeUpdate(query);
} catch (SQLException e1) {
    System.out.println("There was a problem with executing query:"+query);
    e1.printStackTrace();
}
```

Update a Table via Result Set

```
srs.beforeFirst();
while (srs.next()) {
    srs.updateString("description", "new value for description");
    srs.updateRow();
}
```

Prepared Statements

- Create a prepared statement with placeholders for the values

```
ps = conn.prepareStatement("UPDATE " + tablename + " SET " + fieldname + " = ? WHERE "
        + fieldname + " = ?");
ps.setString(1, "Gary");
ps.setString(2, "Bob");
ps.executeUpdate();
```

Managing database connections

```
/**
 * Creates a connection to the database.
 *
 * @return Connection the connection object.
 */
private Connection getNewConnection() {
    String connectionURL = "jdbc:mysql://localhost:3306/projects";
    Connection connection = null;

    try {
        connection = (Connection) DriverManager.getConnection(
                connectionURL, "devuser", "devpass");

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return connection;
}

/**
 * Initializes a number of db connections
 * @param sizeOfConnectionPool
 */
public void initializeConnections(int sizeOfConnectionPool){
    int i = 0 ;
    while (i < sizeOfConnectionPool){
        connections.add(getNewConnection());
        i++;
    }
}
```

Managing database connections

```
/**
 *
 * @return
 */
public Connection getAvailableConnection(){
    Connection connectionToReturn = null;

    if(nextIndex == connections.size()){
        nextIndex = 0;
    }

    connectionToReturn = connections.get(nextIndex);

    nextIndex++;

    return connectionToReturn;
}

/**
 * Closes all connections
 */
public void closeConnections(){
    for(Connection conn:connections){
        try {
            if(conn != null){
                conn.close();
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Look at the Documentation for `java.sql`

- Connection
- Statement
- ResultSet