BACHKHOA EDUCATION APTECH

Bài 7

JDBC

Mục tiêu

- Giới thiệu JDBC
- Giới thiệu JDBC Driver
- Statement
- Gọi hàm thủ tục Store Procedure
- Thực hiện lệnh theo lô (batch)

JDBC

Trong ứng dụng Java có sử dụng CSDL:

- Ứng dụng cần mở một kết nối tới **database**, giao tiếp, thực hiện lệnh **SQL** và nhận dữ liệu trả về (nếu có).
- Để kết nối ứng dụng Java tới database, cần phần mềm cho kết nối csdl mà thường gọi chung là **Application Programming Interface (APIs)**.
- Phần mềm đó có tên là **Java Database Connectivity (JDBC)**.
- Phần mềm là tập hợp thư viện và driver csdl thực thi **độc lập** với ngôn ngữ lập trình, hệ thống csdl và hệ điều hành.

JDBC
Java DataBase Connectivity

JDBC

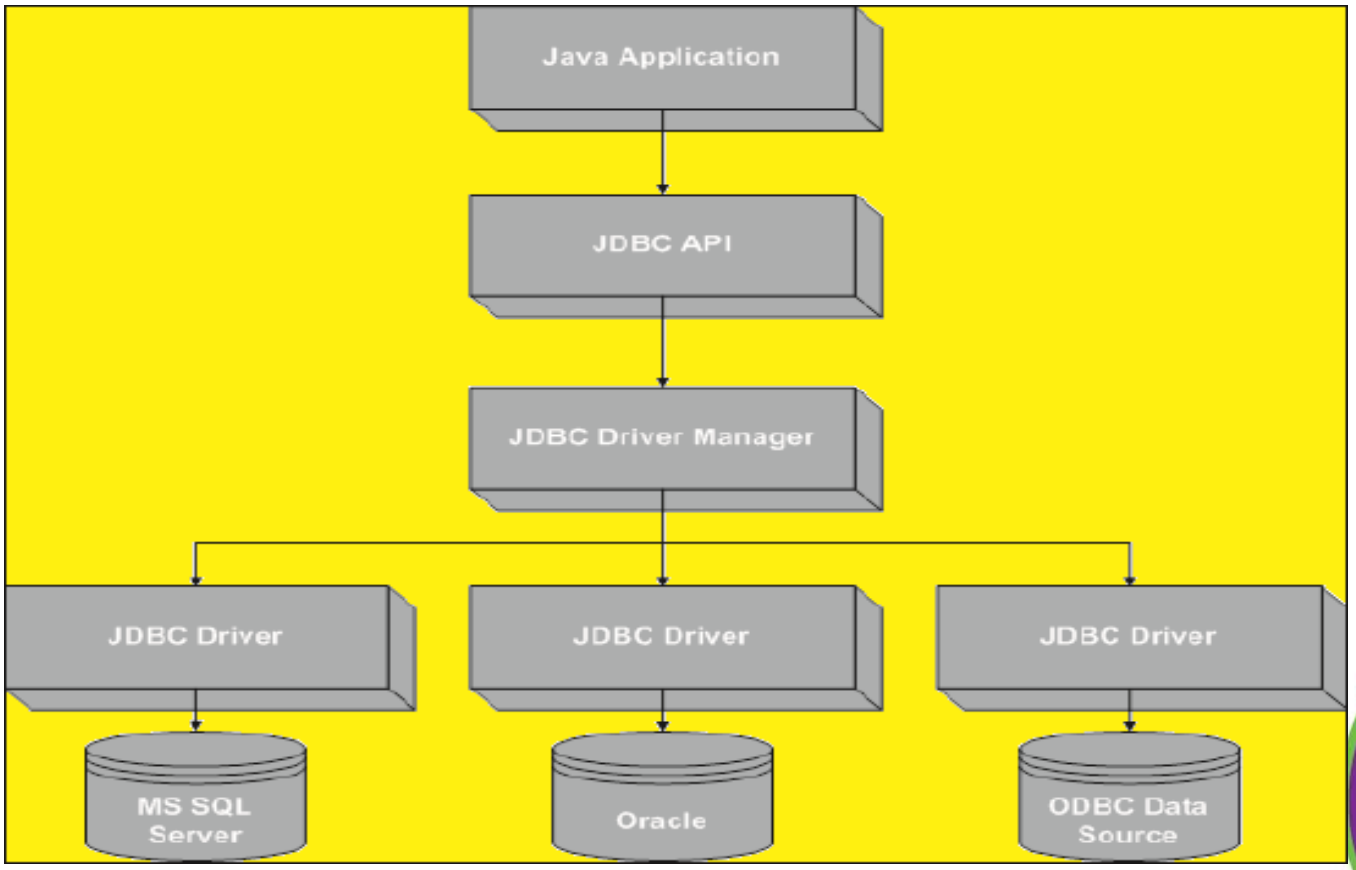
- **JDBC API** là một Java API được cung cấp bởi hãng SUN.
- JDBC API cung cấp **lớp, interface** sử dụng cho truy xuất dữ liệu dạng bảng.
- Nó được viết bằng **ngôn ngữ Java**, cung cấp API chuẩn cho lập trình viên.
- Ưu điểm của việc dùng JDBC API là có thể truy cập database và chạy **độc lập trên bất kỳ nền tảng** nào sử dụng JVM.
- Sự kết hợp JDBC API và nền tảng Java mang lại **lợi thế và tính linh hoạt**.

JDBC

- Một chương trình đơn lẻ với thực thi JDBC có thể gửi lệnh SQL (hoặc lệnh khác) tới cơ sở dữ liệu.
- Có 3 bước để bắt đầu tạo ứng dụng sử dụng JDBC:
    1. Mở kết nối tới nguồn dữ liệu
    2. Gửi truy vấn/lệnh cập nhật tới nguồn dữ liệu
    3. Xử lý kết quả trả về.

JDBC

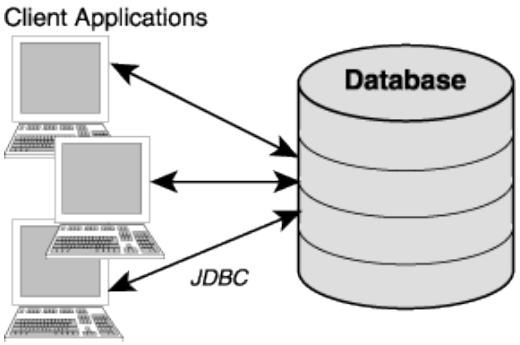
Kiến trúc JDBC

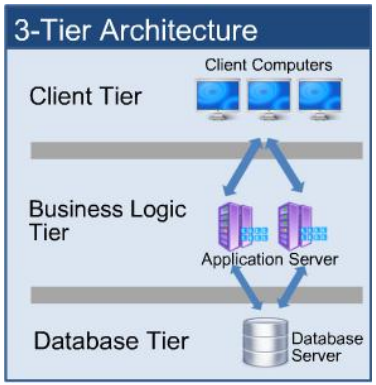


JDBC

Lợi thế ứng dụng Java có sử dụng JDBC:

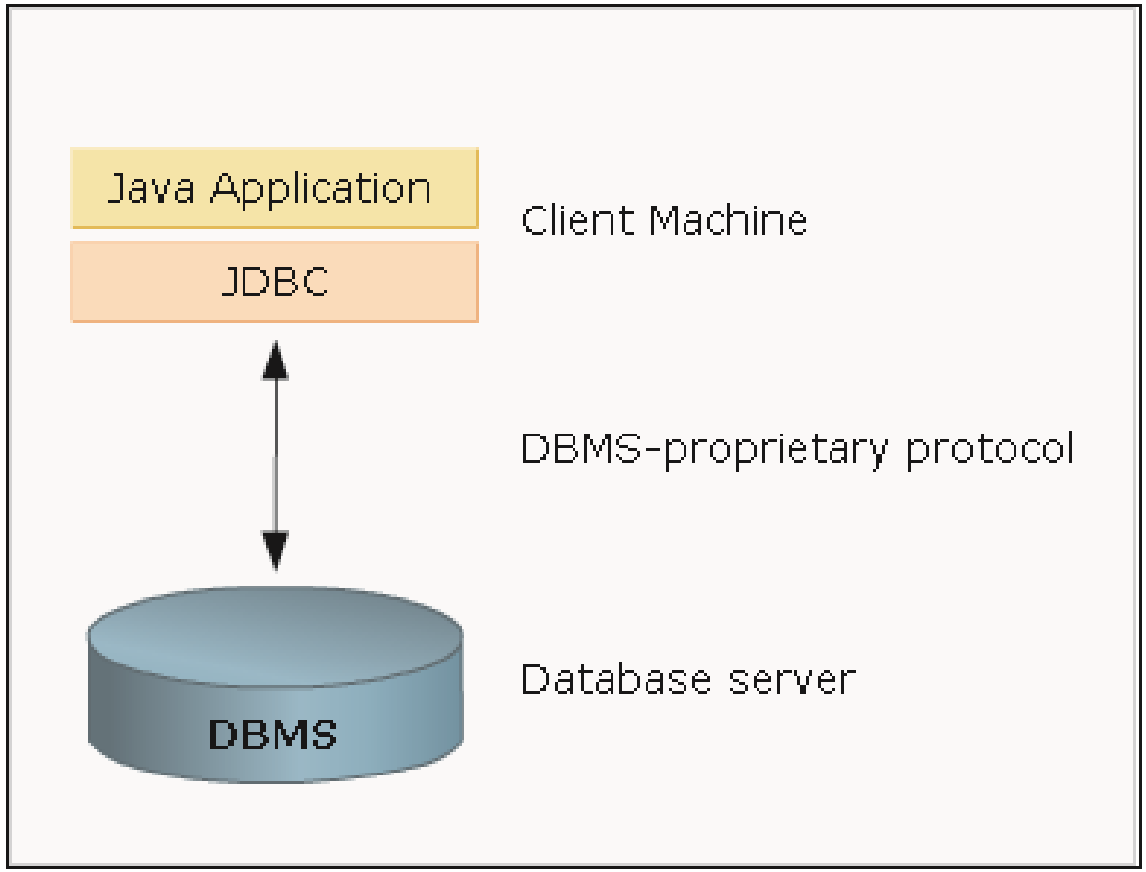
- Sử dụng dữ liệu ngay cả tình huống **nguồn dữ liệu** ở trên các hệ thống quản lý CSDL khác nhau.
- Dữ liệu có thể chuyển đổi sang nhà cung cấp khác mà **không cần** phải sửa chữa mã nguồn.
- Độc lập nền tảng giúp xây dựng ứng dụng ứng dụng 2 tầng, 3 tầng.
- Sự phức tạp của việc kết nối với CSDL được ẩn đi, giúp dễ dàng triển khai, tiết kiệm chi phí bảo trì.





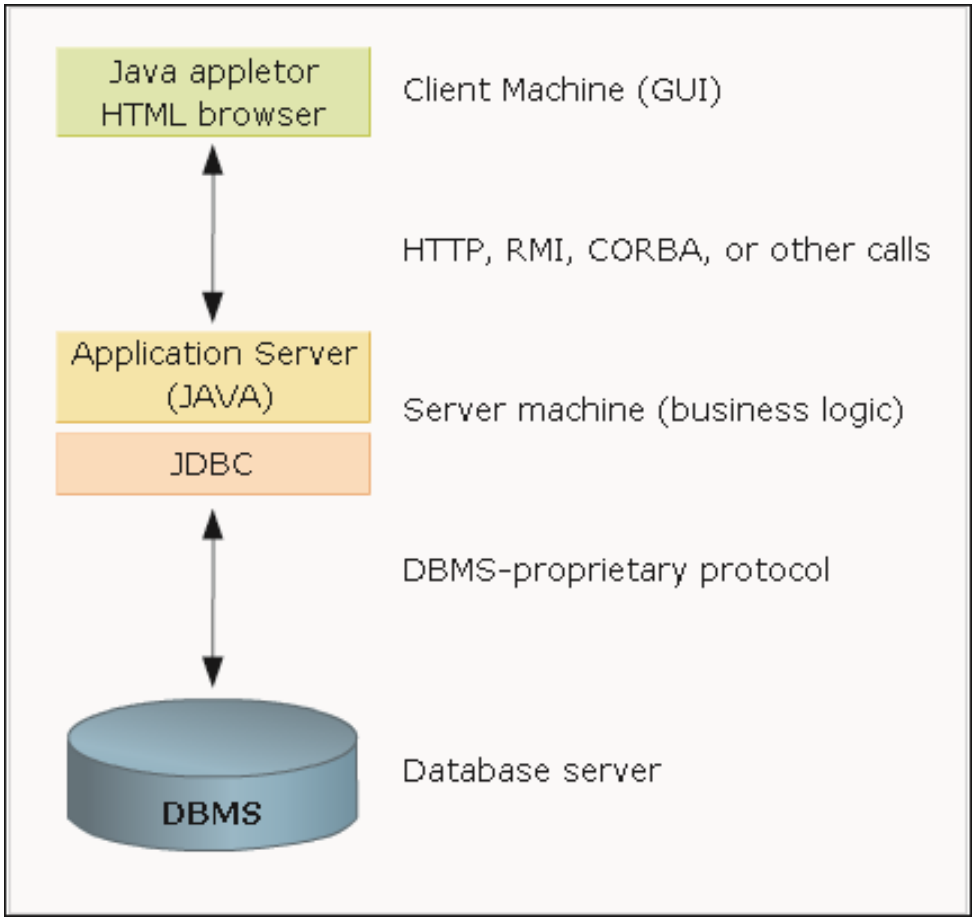
JDBC

Kiến trúc 2 tầng:



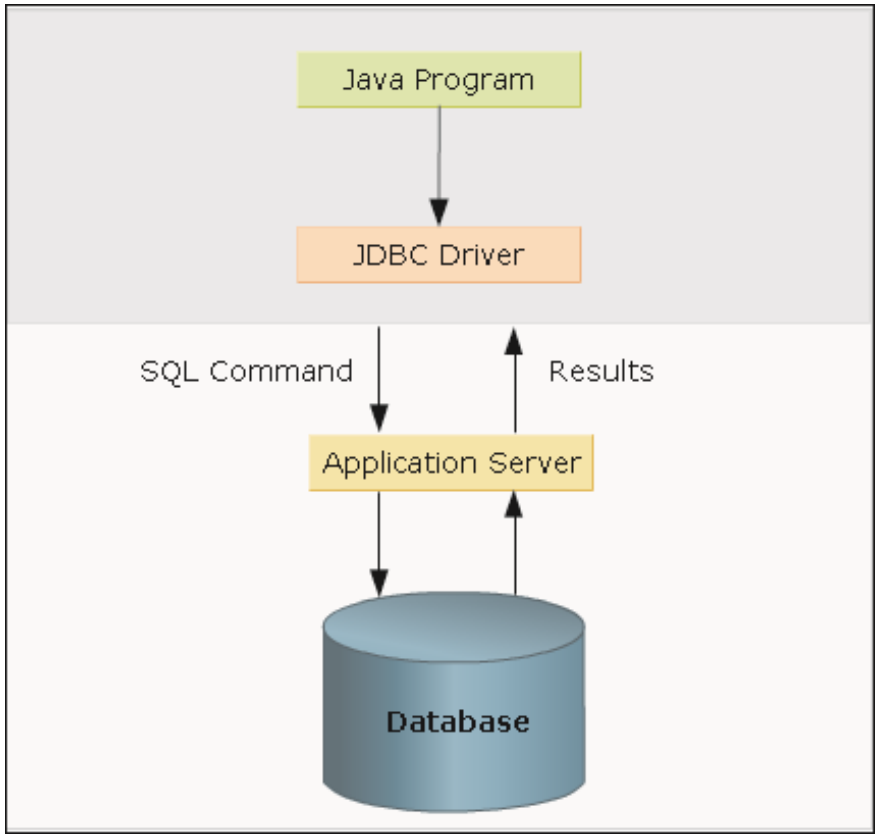
JDBC

Kiến trúc 3 tầng:



JDBC

JDBC API xây dựng tầng giữa trong kiến trúc 3 tầng:



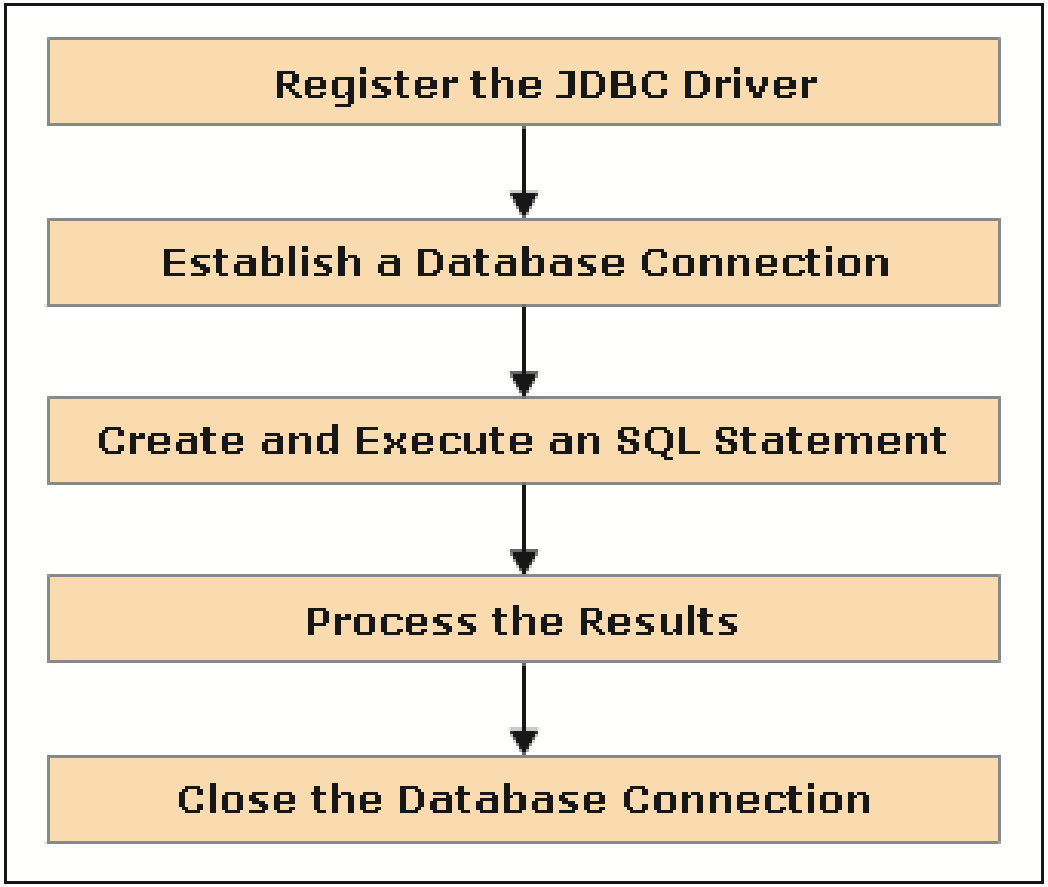
JDBC

JDBC API định nghĩa tập hợp lớp, interface hỗ trợ giao tiếp với CSDL trong **package java.sql**:

- **Connection**: duy trì, giám sát phiên kết nối. Dữ liệu truy cập có thể được kiểm soát bằng cách sử dụng cơ chế khóa giao dịch.
- **DatabaseMetaData**: Interface cung cấp các thông tin csdl như số phiên bản, tên bảng, chức năng hỗ trợ. Lớp có các phương thức giúp lấy thông tin kết nối hiện tại, bảng có sẵn, schema và các danh mục.
- **Driver**: Interface này sử dụng để tạo đối tượng Connection.
- **PreparedStatement**: Sử dụng để thực thi câu lệnh SQL được biên soạn trước.
- **ResultSet**: Interface này cung cấp phương thức nhận dữ liệu trả về từ lệnh SQL.
- **ResultSetMetaData**: Interface này sử dụng để thu thấp thông tin metadata liên quan đến đối tượng `ResultSet`.
- **Statement**: Sử dụng để thực thi lệnh SQL và nhận dữ liệu `ResultSet` trả về.

JDBC

Các bước phát triển:



JDBC Driver

- Tải JDBC Driver sử dụng `Class.forName()`.

Syntax

```
Class.forName(<protocol>)
```

- Kết nối được thiết lập bằng phương thức `DriverManager.getConnection()`
- Các thành phần sử dụng cho kết nối:
  - `Connection URL`
  - `DriverManager.getConnection()`

JDBC Driver

Ngoại lệ:

Code Snippet

```
try {
      Class.forName(“SpecifiedDriverClassName”);
   } catch(java.lang.ClassNotFoundException e) {
      System.err.print(“ClassNotFoundException: “);
      System.err.println(e.getMessage());
     }
```

Statement

- Đối tượng **Statement** sử dụng để thực hiện truy vấn
- Đối tượng Statement được tạo bằng cách gọi phương thức **Connection.createStatement()**.
- Đối tượng Statement có thể được phân làm 3 loại dựa trên kiểu câu lệnh SQL được gửi tới CSDL.
- Statement và **PreparedStatement** kế thừa interface **Statement**.
- **CallableStatement** kế thừa từ interface **PreparedStatement** sử dụng để gọi tới hàm thủ tục **Store Procedure**.
- **PreparedStatement** có thể được thực thi với tham số IN.

Statement

Syntax

```
public Statement createStatement() throws SQLException
```

Code Snippet

```
Connection cn = DriverManager.getConnection(“jdbc:odbc:demo”, “sa”,
“playware”);
Statement st = cn.createStatement();
```

Statement

- Phương thức **executeQuery()** nhận dữ liệu trả về từ lệnh truy vấn.
- Phương thức này nhận một lệnh **SELECT** trong **SQL** như là tham số, dữ liệu trả về là dòng csdl có thể lưu trong đối tượng **ResultSet**.
- Thực thi truy vấn database như sau:
  - **executeQuery()**: phương thức thực thi lệnh SQL "SELECT" với các mệnh đề, nó trả về kết quả truy vấn dưới dạng tập hợp dữ liệu.
  - **ResultSet object**: Đối tượng ResultSet nhận và lưu trữ dữ liệu trả về từ truy vấn SQL. Đối tượng ResultSet được tạo bằng cách gọi phương thức **executeQuery()**.

Statement

Code:

Syntax

```
public ResultSet executeUpdate(String sql) throws SQLException
```

Code Snippet

```
ResultSet rs = st.executeQuery("SELECT * FROM Department");
```

Statement

- Phương thức **executeUpdate()** method thực hiện các lệnh INSERT, DELETE, UPDATE, và các lệnh SQL DDL (Data Definition Language) như là CREATE TABLE, DROP TABLE, và tương tự vậy….
- Phương thức trả về một số nguyên cho biết số row dữ liệu thực thi thành công.

Syntax

```
public int executeUpdate(String sql) throws SQLException
```

- Phương thức **execute()** thực hiện lệnh SQL trả về nhiều hơn một tập kết quả, trả về TRUE nếu thành công

Syntax

```
public boolean execute (String sql) throws SQLException
```

Statement

Truy vấn SQL có tham số

Code Snippet

```
. . .
// Create PreparedStatement object
   String sqlStmt = “UPDATE Employees SET Dept_ID = ? WHERE Dept_Name LIKE ?”;
   PreparedStatement pStmt = cn.preparedStatement(sqlStmt);

// Passing parameters
   pStmt.setInt(1, 25);
   pStmt.setString(2, “Production”);

// Executes the executeUpdate() method
   pStmt.executeUpdate();
```

Statement

Ngoại lệ truy vấn

Code Snippet

```
try {
// Code that could generate an exception goes here.
// If an exception is generated, the catch block below will print out
// information about it.
          }
catch(SQLException ex)
{
                System.err.println(“SQLException: “ + ex.getMessage());
                System.out.println(“ErrorCode: “ + ex.getErrorCode ());
}
```

Statement

Duyệt từng dòng dữ liệu trả về trong **ResultSet**:

Code Snippet

```
  ResultSet rs1 = st1.executeQuery("SELECT Employee_Name FROM
Employees");
 while (rs1.next()) {
    String name=rs1.getString("Employee_Name");
    System.out.println(name);
 }
```

Statement

Dữ liệu trong **ResultSet**:

`getString()`	Lấy về dữ liệu kiểu chuỗi (SQL type VARCHAR).
`getInt()`	Lấy về kiểu dữ liệu số nguyên.
`getFloat()`	Lấy về kiểu dữ liệu số thực.
`getObject()`	Lấy về kiểu dữ liệu dạng đối tượng.

Call Store Procedure

Demo SQL

Code Snippet

```
create procedure recalculatetotal
      @a int, @inc_a int OUT
as
select @inc_a = max(salary) from Employee
set @inc_a = @a * @inc_a;
```

Call Store Procedure

Demo

Code Snippet

```
import java.sql.*;
import java.util.*;
class CallOutProc {
	Connection con;
	String url;
	String serverName;
	String instanceName;
	String databaseName;
	String userName;
	String password;
	String sql;
```

Call Store Procedure

```
   CallOutProc() {
            url = “jdbc:sqlserver://”;
            serverName = “10.2.1.51”;
            instanceName = “martin”;
            databaseName = “DeveloperApps”;
            userName = “sa”;
            password = “playware”;
   }
 private String getConnectionUrl() {
  // Constructing the connection string
  return url + serverName + ”;instanceName = ” +instanceName +”
;DatabaseName = ” +databaseName;
 }
```

Call Store Procedure

```
private java.sql.Connection getConnection() {
  try {
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    // Establishing the connection
          con = DriverManager.getConnection(getConnectionUrl(), userName,
password);
         if(con != null)
             System.out.println("Connection Successful!");
  } catch(Exception e) {
              e.printStackTrace();
              System.out.println("Error Trace in getConnection(): "
              + e.getMessage());
  }
      return con;
}
```

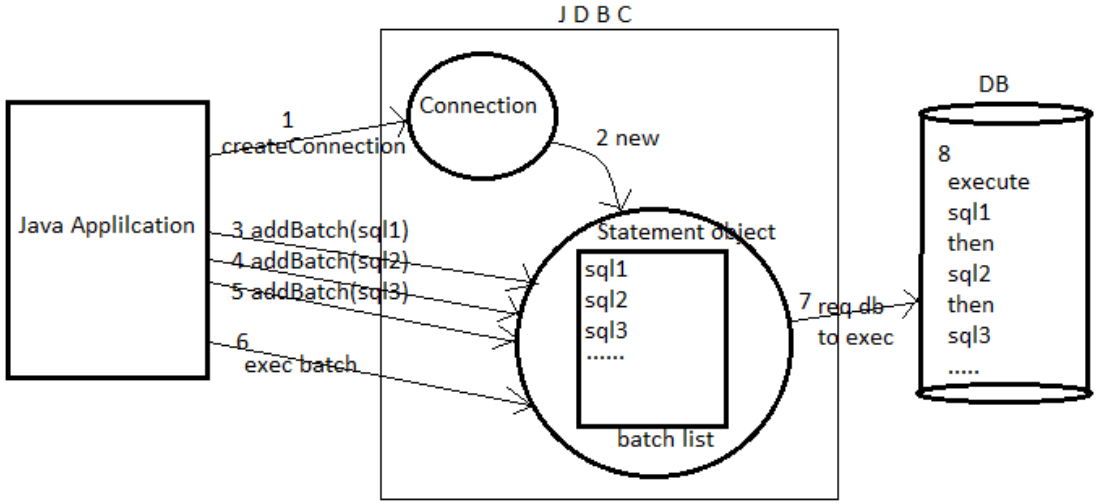
Call Store Procedure

```
public void display(){
  try {
       con = getConnection();
       CallableStatement cstmt = con.prepareCall("{call recalculatetotal (?, ?)}");
      cstmt.setInt(1,2500);
      cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
      cstmt.execute();
      int maxSalary = cstmt.getInt(2);
      System.out.println(maxSalary);
  } catch(SQLException ce) {
              System.out.println(ce);
  }
 }
 public static void main(String args[]) {
      CallOutProc proObj = new CallOutProc();
       proObj.display();
 }
}
```

Batch Update

Thực hiện lệnh theo lô (batch) sử dụng với **PreparedStatement**

- Phương thức **addBatch()** sử dụng để thêm đối tượng Statement vào danh sách chờ thực thi.
- Phương thức **executeBatch()** sử dụng để thực thi danh sách chờ.



Batch Update

Demo:

Code Snippet

```
// Turn off auto-commit
      cn.setAutoCommit(false);
// Creating an instance of Prepared Statement
      PreparedStatement pst = cn.prepareStatement("INSERT INTO
EMPLOYEES VALUES (?, ?)");
// Adding the calling statement batches
    pst.setInt(1, 5000);
    pst.setString(2, "Roger Hoody");
```

Batch Update

Demo:

Code Snippet

```
    pst.addBatch();
    pst.setInt(1, 6000);
    pst.setString(2, “Kelvin Keith”);
    pst.addBatch();
// Submit the batch for execution
    int[] updateCounts = pst.executeBatch();
// Enable auto-commit mode
     cn.commit();
```

Batch Update

Thực hiện lệnh theo lô (batch) sử dụng với **CallableStatement**

Code Snippet

```
// Creating an instance of Callable Statement
CallableStatement cst = cn.prepareCall("{call updateProductDetails(?, ?)}");
// Adding the calling statement batches
      cst.setString(1, "Cheese");
      cst.setFloat(2, 70.99f);
      cst.addBatch();
      cst.setString(1, "Almonds");
      cst.setFloat(2, 80.99f);
      cst.addBatch();
// Submitting the batch for execution
      int [] updateCounts = cst.executeBatch();
// Enabling auto-commit mode
      cn.commit();
```

Tóm tắt bài học

- ✓ **JDBC** là API cho phép ứng dụng Java kết nối tới CSDL.
- ✓ Các thư viện **driver JDBC** khác nhau cho phép kết nối tới các cơ sở dữ liệu khác nhau.
- ✓ Gói **java.sql** chứa các thư viện hỗ trợ truy vấn tới csdl bằng lệnh SQL.
- ✓ Đối tượng **Statement** giúp gửi và thực thi lệnh SQL.
- ✓ Dữ liệu lấy về từ truy vấn được lưu trong đối tượng **ResultSet**.

HẾT
XIN CẢM ƠN!