Core Java

Agenda

- JDBC

JDBC

Call Stored Procedure using JDBC (without OUT parameters)

- Stored Procedure - Change price of given book id.

```
DELIMITER //

CREATE PROCEDURE sp_updateprice(IN p_id INT, IN p_price DOUBLE)
BEGIN
    UPDATE books SET price=p_price WHERE id=p_id;
END;
//

DELIMITER ;
```

```
CALL sp_updateprice(22, 543.21);
```

- JDBC use CallableStatement interface to invoke the stored procedures.
- CallableStatement interface is extended from PreparedStatement interface.
- Steps to call Stored procedure are same as PreparedStatement.

  - Create connection.
  - Create CallableStatement using con.prepareCall("CALL ...").
  - Set IN parameters using stmt.setXYZ(...);
  - Execute the procedure using stmt.executeQuery() or stmt.executeUpdate().
  - Close statement & connection.

- To invoke stored procedure, in general stmt.execute() is called. This method returns true, if it is returning ResultSet (i.e. multi-row result). Otherwise it returns false, if it is returning update/affected rows count.

```
boolean isResultSet = stmt.execute();
if(isResultSet) {
    ResultSet rs = stmt.getResultSet();
    // process the ResultSet
}
else {
    int count = stmt.getUpdateCount();
    // process the count
}
```

Call Stored Procedure using JDBC (with OUT parameters)

- Stored Procedure - Get title and price of given book id.

```
DELIMITER //

CREATE PROCEDURE sp_gettitleprice(IN p_id INT, OUT p_name CHAR(40), OUT p_price DOUBLE)
BEGIN
    SELECT name INTO p_name FROM books WHERE id=p_id;
    SELECT price INTO p_price FROM books WHERE id=p_id;
END;
//
```

```
DELIMITER ;
```

```
CALL sp_gettitleprice(22, @p_name, @p_price);

SELECT @p_name, @p_price;
```

- Steps to call Stored procedure with out params.
    - Create connection.
    - Create CallableStatement using con.prepareCall("CALL ...").
    - Set IN parameters using stmt.setXYZ(...) and register out parameters using stmt.registerOutParam(...).
    - Execute the procedure using stmt.execute().
    - Get values of out params using stmt.getXYZ(paramNumber).
    - Close statement & connection.

Transaction Management

- RDBMS Transactions
    - Transaction is set of DML operations to be executed as a single unit. Either all queries in tx should be successful or all should be discarded.
    - The transactions must be atomic. They should never be partial.

```
CREATE TABLE accounts(id INT, type CHAR(30), balance DOUBLE);
INSERT INTO accounts VALUES (1, 'Saving', 30000.00);
INSERT INTO accounts VALUES (2, 'Saving', 2000.00);
INSERT INTO accounts VALUES (3, 'Saving', 10000.00);

SELECT * FROM accounts;

START TRANSACTION;
```

```
--SET @@autocommit=0;

UPDATE accounts SET balance=balance-4000 WHERE id=1;
UPDATE accounts SET balance=balance+4000 WHERE id=2;

SELECT * FROM accounts;

COMMIT;
-- OR
ROLLBACK;
```

- JDBC transactions (Logical code)

```
try(Connection con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
    con.setAutoCommit(false); // start transaction
    String sql = "UPDATE accounts SET balance=balance+? WHERE id=?";
    try(PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setDouble(1, -3000.0); // amount=3000.0
        stmt.setInt(2, 1); // accid = 1
        cnt1 = stmt.executeUpdate();
        stmt.setDouble(1, +3000.0); // amount=3000.0
        stmt.setInt(2, 2); // accid = 2
        cnt2 = stmt.executeUpdate();
        if(cnt1 == 0 || cnt2 == 0)
            throw new RuntimeException("Account Not Found");
    }
    con.commit(); // commit transaction
}
catch(Exception e) {
    e.printStackTrace();
    con.rollback(); // rollback transaction
}
```

ResultSet

- ResultSet types
  - TYPE_FORWARD_ONLY -- default type
    - next() -- fetch the next row from the db and return true. If no row is available, return false.

```
while(rs.next()) {
    // ...
}
```

  - TYPE_SCROLL_INSENSITIVE
    - next() -- fetch the next row from the db and return true. If no row is available, return false.
    - previous() -- fetch the previous row from the db and return true. If no row is available, return false.
    - absolute(rownum) -- fetch the row with given row number and return true. If no row is available (of that number), return false.
    - relative(rownum) -- fetch the row of next rownum from current position and return true. If no row is available (of that number), return false.
    - first(), last() -- fetch the first/last row from db.
    - beforeFirst(), afterLast() -- set ResultSet to respective positions.
    - INSENSITIVE -- After taking ResultSet if any changes are done in database, those will NOT be available/accessible using ResultSet object. Such ResultSet is INSENSITIVE to the changes (done externally).
  - TYPE_SCROLL_SENSITIVE
    - SCROLL -- same as above.
    - SENSITIVE -- After taking ResultSet if any changes are done in database, those will be available/accessible using ResultSet object. Such ResultSet is SENSITIVE to the changes (done externally).
- ResultSet concurrency
  - CONCUR_READ_ONLY -- Using this ResultSet one can only read from db (not DML operations). This is default concurrency.
  - CONCUR_UPDATABLE -- Using this ResultSet one can read from db as well as perform INSERT, UPDATE and DELETE operations on database.

```
String sql = "SELECT roll, name, marks FROM students";
stmt = con.prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

```
rs = stmt.executeQuery();
```

```
rs.absolute(2); // moves the cursor to the 2nd row of rs
rs.updateString("name", "Bill"); // updates the 'name' column of row 2 to be Bill
rs.updateDouble("marks", 76.32); // updates the 'marks' column of row 2 to be 76.32
rs.updateRow(); // updates the row in the database
```

```
rs.moveToInsertRow(); // moves cursor to the insert row -- is a blank row
rs.updateInt(1, 9); // updates the 1st column (roll) to be 9
rs.updateString(2, "AINSWORTH"); // updates the 2nd column (name) of to be AINSWORTH
rs.updateDouble(3, 76.23); // updates the 3rd column (marks) to true 76.23
rs.insertRow(); // inserts the row in the database
rs.moveToCurrentRow();
```

```
rs.absolute(2); // moves the cursor to the 2nd row of rs
rs.deleteRow(); // deletes the current row from the db
```

JDBC Tutorials

- Transactions: https://youtu.be/Wh6nrkB_o8c
- DAOs: https://youtu.be/TfIpk7ITCGk

HTTP protocol

- HTTP -- Hyper Text Transfer Protocol.
- Connection-less protocol.
- State-less protocol.

- Request-response model.
- Web server is program that enable loading multiple web applications in it.
- Web application is set of web pages (static or dynamic), which are served over HTTP protocol.
- Client makes request by entering URL, click submit, or click hyper-link.
- URL: http://server:port/appln/resource
    - http: protocol/scheme
    - server: machine name or IP address
    - port: default 80
    - URI: /appln/resource
- Request Headers
    - Server/Host: server name/ip + port
    - User-Agent: Browser type/version
    - URI
    - HTTP version: 1.0 or 1.1
    - Content-Type: Type of data in Request body -- application/json, text/...
    - Length: Number of bytes in Request body
    - Method:
        - GET: Get the resource from the server.
            - Request sent when URL entered in address bar, hyper-link is clicked, html form with method=get is submitted.
            - The data (in html form) is sent via URL.
            - Not secured (because data visible in URL).
            - Faster.
        - POST: Post data to the server.
            - Request sent when html form with method=post is submitted.
            - The data (in html form) is sent via request body.
            - More secure
        - HEAD: Send response headers only.
            - No response data is sent to the client.
        - PUT: Put/upload a resource on server.
        - DELETE: Delete a resource from the server.
        - TRACE: Tracing/Information logging

        - OPTIONS: To know which request methods are supported for the resource.
    - Cookies, ...
- Request Body: JSON, Form-Data, or Other.
- Response Headers
    - Status: Code/Text
        - 1xx: Information
        - 2xx: Success e.g. 200 (Ok), 201 (Created), ...
        - 3xx: Redirection e.g. 302
        - 4xx: Client errors e.g. 404 (Not found), 403 (Forbidden), ...
        - 5xx: Server errors e.g. 500 (Internal server error), ...
    - Content-Type: Type of data in Response body
        - text/... : plain, html, xml
        - image/...: png, jpeg, gif, svg
        - audio/...: mp3, wav
        - video/...: mpeg
        - application/...: json, ...
    - Length: Number of bytes in Response Body
    - Cookies, ...
    - Server Info: IP, port, server type, ...
- Quick Revision: https://youtu.be/N_cgBn2KIto

Assignments

1. Complete Election Application.
    - Top Level Menu:
        1. Sign In

```
private static void userAuthentication() {
    try(UserDao userDao = new UserDaoImpl()) {
        System.out.print("Enter email: ");
        String email = sc.next();
        System.out.print("Enter passwd: ");
```

```
            String passwd = sc.next();
            User u = userDao.findByEmail(email);
            if(u != null && u.getPassword().equals(passwd)) {
                System.out.println("Login Successful: " + u);
                curUser = u;
                if(u.getRole().equals("voter"))
                    userMenu();
                else
                    adminMenu();
            } else
                System.out.println("Login Failed");
        } // userDao.close();
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

2. Sign Up
3. Exit

- User/Voter Menu:
  1. Show all Candidates
  2. Vote
     - CandidateDao interface -- int incrVoteById(int candId);
     - CandidateDaoImpl class -- Implement above method.
       - UPDATE candidates SET votes = votes + 1 WHERE id = ?;
     - UserDao interface -- int update(User user);
     - UserDaoImpl class -- Implement above method.
       - UPDATE candidates SET first_name=?, ... WHERE id=?;
     - Voting logic

```
if(curUser.getStatus() != 0) {
  sysout("Alredy voted.");
```

```
        return;
    }
    // input candidate id to vote -- Scanner
    int count = candDao.incrVoteById(candId);
    if(count == 1) {
        curUser.setStatus(1);
        userDao.update(curUser);
    }
```

3. Show result
    - CandidateDao interface -- Candidate findByMaxVotes();
    - CandidateDaoImpl class -- Implement above method.
        - SELECT * FROM candidates ORDER BY votes DESC LIMIT 1

- Admin Menu:
    1. Add Candidate (HW)
    2. Delete Candidate (HW)
    3. Update Candidate (HW)
    4. Show all Candidates
    5. Show Full result
        - CandidateDao interface -- Candidate findAllOrderByVotes();
        - CandidateDaoImpl class -- Implement above method.
            - SELECT * FROM candidates ORDER BY votes DESC