Day 11

Driver

- It is interface declared in java.sql package.
- JDBC Driver must implement java.util.Driver interface.
- It is used to establish connection between java application and database.
- If we want to use JDBC driver then first it must be registered.
- Registration can be done using 2 ways

First Way

```
Driver driver = ...;
DriverManager.registerDriver( driver );
```

Second Way

```
String driverName = "...";
Class.forName( driverName );
```

- Since Java SE 8 this step is optional.
- The DriverManager will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

DriverManager

- It is a class declared in java.sql package.
- DriverManager is responsible for locating proper driver and establishing connection with database using driver object.
- getConnection( ) is a method of DriverManager which returns Connection object.

```
Connection c = DriverManager.getConnection( url, user, password );
```

Connection

- It is interface declared in java.sql package.
- It represents session between java application and database.
- In other words, we can perform operations on database after establising connection and before closing connection.
- Methods of Connection Interface:

1. Statement createStatement() throws SQLException

2. PreparedStatement prepareStatement(String sql) throws SQLException
3. CallableStatement prepareCall(String sql) throws SQLException
4. void setAutoCommit(boolean autoCommit) throws SQLException
5. void commit() throws SQLException
6. void rollback() throws SQLException
7. Savepoint setSavepoint() throws SQLException

Statement

- It is interface declared in java.sql package.
- Instantiation :

```
Statement stmt = con.createStatement();
```

- If we want to execute static queries then we should use Statement object
- Limitations :
    1. It can not handle special characters in query
    2. It can not avoid SQL Injection.
    3. If use statement object to execute sql statement then query gets compiled and executed per request.
- Methods:
    1. ResultSet executeQuery(String sql) throws SQLException
    2. int executeUpdate(String sql) throws SQLException

PreparedStatement

- It is sub interface of Statement interface.
- To overcome limitations of statement we should use PreparedStatement object.
- Instantiation :

```
String sql = "";
PreparedStatement stmt = con.prepareStatement( sql );
```

- It compiles query only once and execute it multiple times.
- Using PreparedStatement object, we can not execute stored procedure and function.

CallableStatement

- It is sub interface of PreparedStatement interface.
- If we want to execute stored procedure or function then we should use Callable Statement object.
- Instantiation:

```
String sql = "....";
CallableStatement stmt = con.prepareCall(sql);
```

- Syntax to call Stored procedure

```
 {call sp_procedure_name(arg1,arg2, ...)}
```

- Syntax to call Stored function

```
{?=call sf_function_name(arg1,arg2, ...)}
```

- If we want to execute stored procedure/function then we should use execute() method boolean execute() throws SQLException
- execute() method returnds boolean value. * true if the first result is a ResultSet object;
    - false if the first result is an update count or there is no result

MySQL Stored Procedure to insert book

```
DELIMITER $$
CREATE PROCEDURE sp_insert_book
(
pBookId INT,
pSubjectName VARCHAR(50),
pBookName VARCHAR(50),
pAuthorName VARCHAR(50),
pPrice FLOAT
)
BEGIN
    INSERT INTO BookTable VALUES
    ( pBookId, pSubjectName, pBookName,pAuthorName, pPrice );
END $$
DELIMITER ;
```

MySQL Stored Procedure to update book

```
DELIMITER $$
CREATE PROCEDURE sp_update_book
( pBookId INT, pPrice FLOAT )
BEGIN
    UPDATE BookTable SET price=pPrice WHERE book_id=pBookId;
END $$
DELIMITER ;
```

MySQL Stored Procedure to delete book

```
DELIMITER $$
CREATE PROCEDURE sp_delete_book
( pBookId INT )
BEGIN
   DELETE FROM BookTable WHERE book_id=pBookId;
END $$
DELIMITER ;
```

MySQL Stored Procedure to get all books

```
DELIMITER $$
CREATE PROCEDURE sp_select_book
(  )
BEGIN
  SELECT * FROM BookTable;
END $$
DELIMITER ;
```

IN and OUT parameter in Stored Procedure

```
CREATE TABLE accounts
(
    acc_number INT,
    name VARCHAR(50),
    balance FLOAT
);
```

```
INSERT INTO accounts VALUES(101, 'Rahul', 50000);

INSERT INTO accounts VALUES(102, 'Sandeep', 30000);
```

```
DELIMITER $$
CREATE PROCEDURE sp_fund_transfer
(
    IN srcAccountNumber INT,
    IN destAccountNumber INT,
    IN amount FLOAT,
    OUT srcBalance FLOAT,
    OUT destBalance FLOAT
)
BEGIN
    UPDATE accounts SET
    balance = balance - amount
```

```
    WHERE acc_number = srcAccountNumber;

    UPDATE accounts SET
    balance=balance + amount
    WHERE acc_number = destAccountNumber;

    SELECT balance INTO srcBalance
    FROM accounts
    WHERE acc_number = srcAccountNumber;

    SELECT balance INTO destBalance
    FROM accounts
    WHERE acc_number = destAccountNumber;
END $$
DELIMITER ;
```

JDBC Transaction

- If we perform any operation from java application then it is bu default auto commited.
- If we want to change that behavior then we should use setAutoCommit method.

```
con.setAutoCommit( false );
```

```
Connection connection = null;
Statement statement = null;
try
{
    connection = DBUtils.getConnection();
    statement =connection.createStatement();
    connection.setAutoCommit(false);

    String sql = "UPDATE accounts SET balance=balance - 5000 WHERE
acc_number=101";
    statement.executeUpdate(sql);

    sql = "UPDATE accounts SET balance=balance + 5000 WHERE
acc_number=102";
    statement.executeUpdate(sql);

    connection.commit();
    System.out.println("Fund transferd..");
}
catch( Exception ex )
{
    try
    {
        connection.rollback();
        ex.printStackTrace();
    }
```

```
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
```

ResultSet

- ResultSet object represents records returned by select query.
- Following are methods of java.sql.Connection interface:

1. Statement createStatement() throws SQLException
2. Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException
3. Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException

- ResultSet Fields

1. TYPE_FORWARD_ONLY
2. TYPE_SCROLL_INSENSITIVE
3. TYPE_SCROLL_SENSITIVE
4. CONCUR_READ_ONLY
5. CONCUR_UPDATABLE
6. HOLD_CURSORS_OVER_COMMIT
7. CLOSE_CURSORS_AT_COMMIT

- ResultSet Methods

1. boolean isBeforeFirst() throws SE
2. void beforeFirst() throws SE
3. boolean first() throws SE
4. boolean isAfterLast() throws SE
5. void afterLast() throws SE
6. boolean last() throws SE
7. boolean next() throws SE
8. boolean previous() throws SE
9. boolean absolute(int row) throws SE
10. boolean relative(int rows) throws SE
11. void updateRow() throws SE
12. void deleteRow() throws SE

13. void insertRow() throws SE

ResultSet Type

1. TYPE_FORWARD_ONLY
2. TYPE_SCROLL_INSENSITIVE
3. TYPE_SCROLL_SENSITIVE

ResultSet Concurrency

1. CONCUR_READ_ONLY
2. CONCUR_UPDATABLE

ResultSet Holdability

1. HOLD_CURSORS_OVER_COMMIT
2. CLOSE_CURSORS_AT_COMMIT

- By default, ResultSet object is forward only and read only.

```
Statement stmt = con.createStatement( );
```

- above statement is equivalent to

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

- How to make ResultSet Scollable?

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

- Or

```
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE );
```

- Following are methods of DatabaseMetaData interface which is used to check database support for following features:

1. boolean supportsResultSetType(int type)
2. boolean supportsResultSetConcurrency(int type, int concurrency) throws SQLException
3. boolean supportsResultSetHoldability(int holdability) throws SQLException

```
connection = DBUtils.getConnection();
DatabaseMetaData dm =
             connection.getMetaData();
```