Advanced Programming

Week 7

Database Programming

- JDBC Classes and Interfaces
- Transaction Management in JDBC
- Batch Processing in JDBC
- JDBC RowSet

Tilahun Melak(PhD)

October, 2025

Objectives

At the end of this lecture, students will be able to:

- Manipulate database data using JDBC.
- Explain Atomicity, Consistency, Isolation, and Durability—that ensure reliable transaction management.
- Explain the steps required for batch processing in JDBC.
- Explain JDBC RowSet.

JDBC Classes and Interfaces Cntd…

- **Statement interface**

Methods of Statement interface

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

Oracle. (n.d.). JDBC basics. Oracle.

JDBC Classes and Interfaces Cntd…

- **ResultSet interface**
  - The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row.
  - **By default, ResultSet object can be moved forward only and it is not updatable.**

Oracle. (n.d.). Interface ResultSet (Java SE Documentation). Oracle.

JDBC Classes and Interfaces Cntd…

- **ResultSet interface**
  - But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in **createStatement(int,int**) method as well as we can make this object as updatable by:
  - Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

Oracle. (n.d.). *Interface ResultSet (Java SE Documentation)*. Oracle.

Example: Fetching Records

```
import java.sql.*;
  class FetchRecord
  {
  public static void main(String args[])throws Exception{
    try
    {
  Class.forName("com.mysql.jdbc.Driver");
  Connection
   con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","r
   oot","root");
```

Example: Fetching Records

```
Statement
    stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.C
    ONCUR_UPDATABLE);
   ResultSet rs=stmt.executeQuery("select * from stud");


   //getting the record of 3rd row
   rs.absolute(3);
   System.out.println("ID\tName\t\tGender\tDept");
   System.out.print("---------------------------------\n");
```

Example: Fetching Records

```
System.out.println(rs.getString(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"
    \t"+rs.getString(4));
   System.out.println("----------------------------------");
    con.close();
     }
     catch(Exception e)
     {
        e.printStackTrace();
     }
   }
   }
```

JDBC Classes and Interfaces Cntd…

- **ResultSet interface**

`ResultSet` static type constant	Description
`TYPE_FORWARD_ONLY`	Specifies that a `ResultSet`'s cursor can move only in the forward direction (i.e., from the first row to the last row in the `ResultSet`).
`TYPE_SCROLL_INSENSITIVE`	Specifies that a `ResultSet`'s cursor can scroll in either direction and that the changes made to the `ResultSet` during `ResultSet` processing are not reflected in the `ResultSet` unless the program queries the database again.
`TYPE_SCROLL_SENSITIVE`	Specifies that a `ResultSet`'s cursor can scroll in either direction and that the changes made to the `ResultSet` during `ResultSet` processing are reflected immediately in the `ResultSet`.

`ResultSet` constants for specifying `ResultSet` type.

Oracle. (n.d.). *Interface ResultSet (Java SE Documentation)*. Oracle.

JDBC Classes and Interfaces Cntd…

- **ResultSet interface**

`ResultSet` static concurrency constant	Description
`CONCUR_READ_ONLY`	Specifies that a `ResultSet` cannot be updated (i.e., changes to the `ResultSet` contents cannot be reflected in the database with `ResultSet`'s update methods).
`CONCUR_UPDATABLE`	Specifies that a `ResultSet` can be updated (i.e., changes to the `ResultSet` contents can be reflected in the database with `ResultSet`'s update methods).

`ResultSet` constants for specifying result properties

Oracle. (n.d.). *Interface ResultSet (Java SE Documentation)*. Oracle.

JDBC Classes and Interfaces Cntd…

- **PreparedStatement interface**
  - The PreparedStatement interface is a subinterface of Statement.
  - It is used to execute parameterized query.
  - Example: String sql="insert into emp values(?,?,?)";
  - As you can see, we are passing parameter (?) for the values.
  - Its value will be set by calling the setter methods of PreparedStatement.

Oracle. (n.d.). PreparedStatement interface (Java SE Documentation). Oracle.

Example: Prepared Statement

```
import java.sql.*;
  class PreparedInsrt{
  public static void main(String args[]){
  try
  {
  Class.forName("com.mysql.jdbc.Driver");
  Connection
   con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","r
   oot","root");
   PreparedStatement stmt=con.prepareStatement("insert into stud
   values(?,?,?,?)");
```

Example: Prepared Statement

```
stmt.setString(1,"r/109/08");//1 specifies the first parameter in the query
stmt.setString(2,"Sagni");
stmt.setString(3,"Male");
stmt.setString(4,"Marketing");
int i=stmt.executeUpdate();
System.out.println(i+" records inserted");
con.close();
}
catch(Exception e)
{
    e.printStackTrace();
} }}
```

JDBC Classes and Interfaces Cntd…

- **ResultSetMetaData Interface**
  - The metadata means data about data i.e. we can get further information from the data.
  - If you have to get metadata of a table like total number of column, column name, column type etc. **, ResultSetMetaData** interface is useful because it provides methods to get metadata from the ResultSet object.
  - The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData.

Syntax: public ResultSetMetaData getMetaData()throws SQLException

Oracle. (n.d.). ResultSetMetaData interface (Java SE Documentation). Oracle.

JDBC Classes and Interfaces Cntd…

- **ResultSetMetaData Interface**

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.

Oracle. (n.d.). JDBC overview. Oracle Help Center.

Example: ResultSet MetaData

```
import java.sql.*;
class rsmd{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
 Connection
    con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sys","root","root");

PreparedStatement ps=con.prepareStatement("select * from sys_config");
ResultSet rs=ps.executeQuery();
```

Example: ResultSet MetaData

```
ResultSetMetaData rsmd=rs.getMetaData();
  int x=rsmd.getColumnCount();
  System.out.println("column count="+x);
  for(int i=1;i<=x;i++){
      System.out.println("Column Name of column: "+rsmd.getColumnName(i));
System.out.println("Column Type Name of column: "+rsmd.getColumnTypeName(i));
  }
con.close();
  }
catch(Exception e)
{
    e.printStackTrace();
}} }
```

Example: Select

```
import java.sql.*;
class Select{
    public static void main(String[] args) {
        try{
        Class.forName("com.mysql.jdbc.Driver");
        Connection
      con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM stud");
            ResultSetMetaData rsmd = rs.getMetaData();
```

Example: Select

```
int noCol = rsmd.getColumnCount();
for(int i = 1; i <= noCol; i++)
{
   if(i==2)
      System.out.print("\t");
   System.out.print(rsmd.getColumnName(i)+"\t");
}

System.out.print("\n-----------------------------------\n");
```

Example: Select

```
        while(rs.next()){
            for(int i = 1; i <= noCol; i++)
            {
                System.out.print( rs.getString(i)+"\t");
            }
            System.out.println();
           }
        System.out.println("-----------------------------------");
      }
      catch(Exception e){e.printStackTrace();}
   }
}
```

JDBC Classes and Interfaces Cntd…

- **DatabaseMetaData interface**
  - DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.
  - The getMetaData() method of Connection interface returns the object of DatabaseMetaData.

Syntax: public DatabaseMetaData getMetaData()throws SQLException

Oracle. (n.d.). *DatabaseMetaData (Java Platform SE 8)*. Oracle.

JDBC Classes and Interfaces Cntd…

- **methods of DatabaseMetaData interface**

- ✓ **public String getDriverName()throws SQLException:** it returns the name of the JDBC driver.
- ✓ **public String getDriverVersion()throws SQLException:** it returns the version number of the JDBC driver.
- ✓ **public String getUserName()throws SQLException:** it returns the username of the database.

Oracle. (n.d.). *DatabaseMetaData (Java Platform SE 8)*. Oracle.

JDBC Classes and Interfaces Cntd…

- **methods of DatabaseMetaData interface**

- ✓ **public String getDatabaseProductName()throws SQLException:** it returns the product name of the database.
- ✓ **public String getDatabaseProductVersion()throws SQLException:** it returns the product version of the database.
- ✓ **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:** it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

Oracle. (n.d.). *DatabaseMetaData (Java Platform SE 8)*. Oracle.

Example: Database Metadata

```
import java.sql.*;
class DBMD{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection
  con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");

DatabaseMetaData dbmd=con.getMetaData();
```

Example: Database Metadata

```
System.out.println("Driver Name: "+dbmd.getDriverName());
    System.out.println("Driver Version: "+dbmd.getDriverVersion());
    System.out.println("UserName: "+dbmd.getUserName());
    System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
    System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
    con.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    }
    }
```

JDBC Classes and Interfaces Cntd…

- **Callable Statement Interface**
  - To call the **stored procedures and functions**, CallableStatement interface is used.
  - We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
  - Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output

Oracle. (n.d.). Java™ Platform, Standard Edition 8 API Specification: CallableStatement Interface. Oracle.

JDBC Classes and Interfaces Cntd…

- **How to get the instance of CallableStatement?**

The **prepareCall()** method of Connection interface returns the instance of CallableStatement.

Syntax:

public CallableStatement prepareCall("{ call procedurename(?,?...?)}");

The example to get the instance of CallableStatement is given below:

- CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
- It calls the procedure myprocedure that receives 2 arguments.

Oracle. (n.d.). Java™ Platform, Standard Edition 8 API Specification: CallableStatement Interface. Oracle.

JDBC Classes and Interfaces Cntd…

- **Example of CallableStatement**

To call the stored procedure, you need to create it in the database. Here, we are assuming that stored procedure looks like this.

```
create or replace procedure "INSERTrecord"
(id IN VARCHAR, Name VARCHAR,Gender IN VARCHAR,Department IN
VARCHAR)
is
begin
insert into stud values(id,Name,Gender,Department);
end;
```

Oracle. (n.d.). Java™ Platform, Standard Edition 8 API Specification: CallableStatement Interface. Oracle.

Transaction Management in JDBC

- Transaction represents **a single unit of work**.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

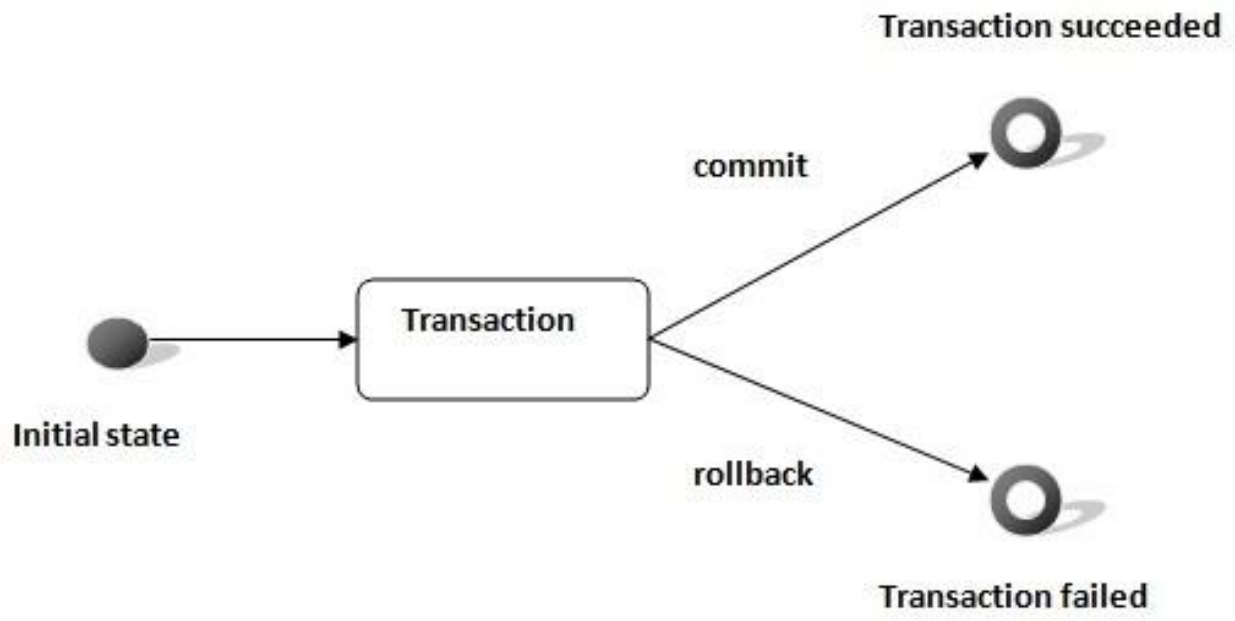
- **Atomicity** means either all successful or none.
- **Consistency** ensures bringing the database from one consistent state to another consistent state.
- **Isolation** ensures that transaction is isolated from other transaction.
- **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (7th ed.). Pearson.

Transaction Management in JDBC…

- **Advantage of Transaction Management**

fast performance : it makes the performance fast because database is hit at the time of commit.



Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (7th ed.). Pearson.

Transaction Management in JDBC…

- In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (7th ed.). Pearson.

Example: Transaction management

```
mport java.sql.*;
class TCommit{
public static void main(String args[])throws Exception{
    try
    {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
      con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
 con.setAutoCommit(false);
```

Example: Transaction management

```
Statement stmt=con.createStatement();
stmt.executeUpdate("insert into stud values('r/110/08','Lami','Male','ICT')");
stmt.executeUpdate("insert into stud values('r/111/08','Abebe','Male','Electrical')");

con.commit();
con.close();
   }
 catch(Exception e)
  {
     e.printStackTrace();
  }
}
}
```

Transaction Management in JDBC…

- **Batch Processing in JDBC**

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.

The **java.sql.Statement** and **java.sql.PreparedStatement** interfaces provide methods for batch processing

- **Advantage:** Fast Performance

Methods of Statement interface

Method	Description
void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

Oracle. (2023). *The JDBC API: Executing SQL statements*. Oracle.

Transaction Management in JDBC…

Let's see the simple example of batch processing in jdbc.

It follows following steps:

1. Load the driver class
2. Create Connection
3. Create Statement
4. Add query in the batch
5. Execute Batch
6. Close Connection

Oracle. (2023). *The JDBC API: Executing SQL statements*. Oracle.

Transaction Management in JDBC…

- **JDBC RowSet**

The instance of **RowSet** is the java bean component because it has properties and java bean notification mechanism. It is the wrapper of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.

The implementation classes of RowSet interface are as follows:

- JdbcRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet

Oracle. (n.d.). The RowSet interface (JDBC). Oracle.

Transaction Management in JDBC…

- **Steps to create and execute RowSet**.

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:mysql://localhost:3306/student","root","root");
rowSet.setUsername(“root");
rowSet.setPassword(“root");

rowSet.setCommand("select * from stud");
rowSet.execute();
```

- Advantage of RowSet
  - ✓ It is easy and flexible to use
  - ✓ It is Scrollable and Updatable by default

Oracle. (n.d.). The RowSet interface (JDBC). Oracle.

Summary

- In today's lecture we have discussed about;
  - JDBC Classes and Interfaces.
  - Transaction: a single unit of work ensuring ACID—Atomicity, Consistency, Isolation, and Durability.
  - JDBC batch processing allows multiple SQL statements to be grouped and executed together, improving efficiency by reducing database round-trips.
  - JDBC RowSet: a flexible JavaBean wrapper for tabular data.
  - Examples

References

- Oracle. (n.d.). JDBC basics. Oracle.
- Oracle. (n.d.). *Interface ResultSet (Java SE Documentation)*. Oracle.
- Oracle. (n.d.). *PreparedStatement interface (Java SE Documentation)*. Oracle.
- Oracle. (n.d.). *ResultSetMetaData interface (Java SE Documentation)*. Oracle.
- Oracle. (n.d.). *DatabaseMetaData (Java Platform SE 8)*. Oracle.
- Oracle. (n.d.). *Java™ Platform, Standard Edition 8 API Specification: CallableStatement Interface*. Oracle.
- Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (7th ed.). Pearson.
- Oracle. (2023). *The JDBC API: Executing SQL statements*. Oracle.
- Oracle. (n.d.). *The RowSet interface (JDBC)*. Oracle.