JDBC

CallableStatement and Transactions

CallableStatement and Transactions

<u>Objectives</u>

At the end of this module, you will be able to:

- Invoke stored procedures through CallableStatement object

- Understand the function of commit and roll back in transactions

The CallableStatement Object

- A CallableStatement object is used for calling the stored procedure from JDBC program

- A callable statement can contain variables that you supply each time you execute the call

- When the stored procedure returns, computed values (if any) are retrieved through the CallableStatement object

The CallableStatement Object

- The way to access stored procedures using JDBC is through the CallableStatement class which is inherited from the PreparedStatement class. CallableStatement is like PreparedStatement in that you can specify parameters using the question mark (?) notation, but it contains no SQL statements.

- Both functions and procedures take parameters represented by identifiers. A function executes some procedural logic and it returns a value that can be any data type supported by the database. The parameters supplied to the function do not change after the function is executed.

- A procedure executes some procedural logic but does not return any value. However, some of the parameters supplied to the procedure may have their values changed after the procedure is executed.

Note: Calling a stored procedure is the same whether the stored procedure was written originally in Java or in any other language supported by the database, such as PL/SQL. Indeed, a stored procedure written in Java appears to the programmer as a PL/SQL stored procedure.

How to Create a CallableStatement?

- Register the driver and create the database connection

- On connection object prepareCall() method is used to call the stored procedure

- Create the callable statement, identifying variables with a question mark (?)

```
CallableStatement cstmt =
        conn.prepareCall("{call " + ADDITEM + "(?,?,?)}");
        cstmt.registerOutParameter(2,Types.INTEGER);
        cStmt.registerOutParameter(3,Types.DOUBLE);
```

How to Create a CallableStatement?

Creating a Callable Statement

- First you need an active connection to the database in order to obtain a CallableStatement object.

- Next, you create a CallableStatement object using the prepareCall() method of the Connection class. This method typically takes a string as an argument. The syntax for the string has two forms. The first form includes a result parameter and the second form does not:

- {? = call proc (…) }   // A result is returned into a variable
    {call proc (…) }   // Does not return a result

- In the example in the slide, the second form is used, where the stored procedure in question is ADDITEM.

How to Create a CallableStatement?

- Note that the parameters to the stored procedures are specified using the question mark notation used earlier in PreparedStatement. You must register the data type of the parameters using the registerOutParameter() method of CallableStatement if you expect a return value, or if the procedure is going to modify a variable (also known as an OUT variable). In the example in the slide, the second and third parameters are going to be computed by the stored procedure, whereas the first parameter is an input (the input is specified in the next slide). Parameters are referred to sequentially, by number. The first parameter is 1.

- To specify the data type of each OUT variable, you use parameter types from the Types class. When the stored procedure successfully returns, the values can be retrieved from the CallableStatement object.

How to execute a CallableStatement?

1. To pass the input parameters

```
cstmt.setXXX(index, value);
```

2. CallableStatement should be executed, as:

```
cstmt.execute();
```
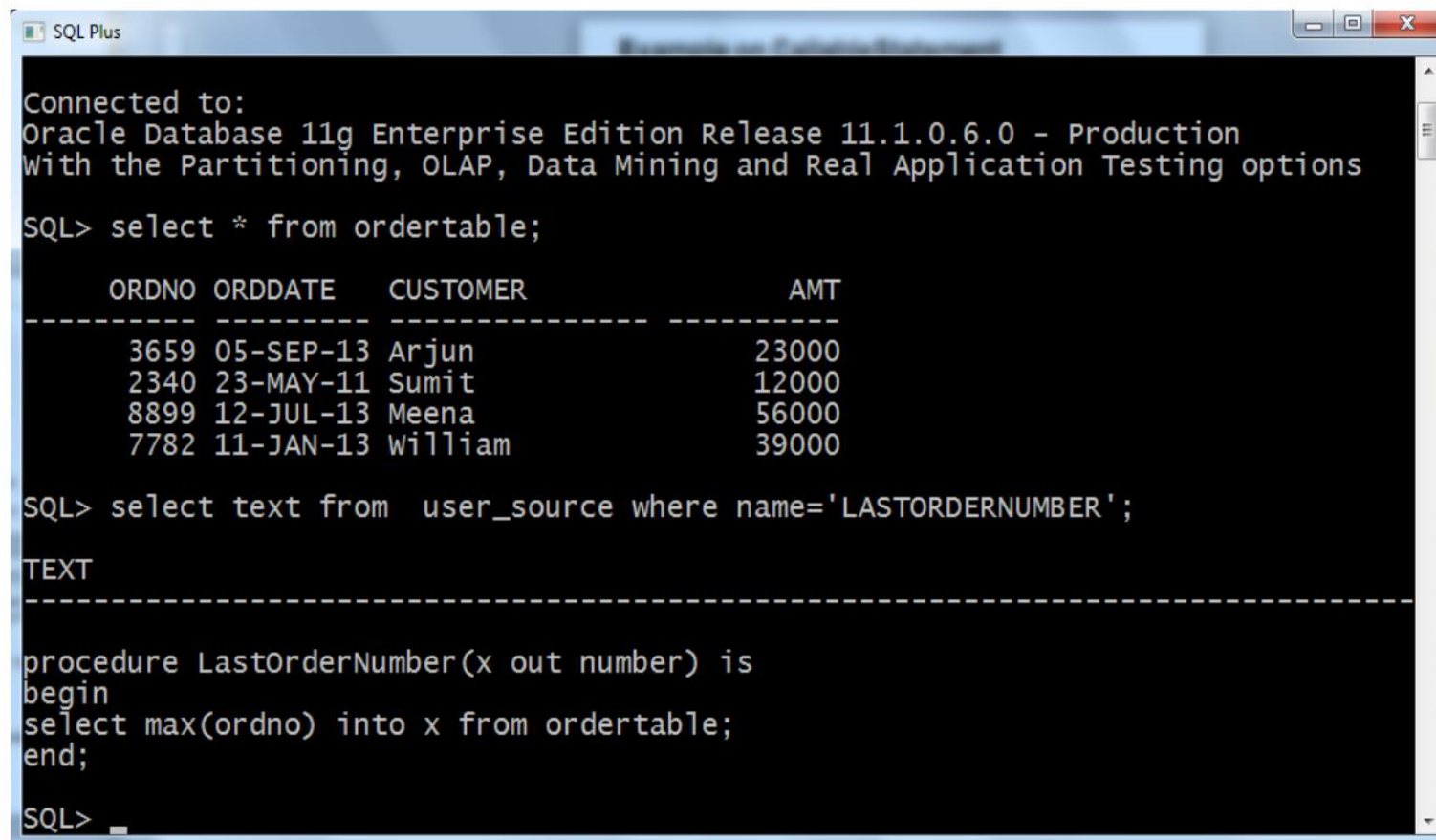
3. To get the output parameters

```
var = cstmt.getXXX(index);
```

How to execute a CallableStatement?

There are three steps in executing the stored procedure after you have registered the types of the OUT variables:

1. Set the IN parameters - Use the set*XXX*() methods to supply values for the IN parameters. There is one setXXX() method for each Java type: setString(), setInt(), and so on. You must use the setXXX() method that is compatible with the SQL type of the variable. You can use setObject() with any variable type. Each variable has an index. The index of the first variable in the callable statement is 1, the index of the second is 2, and so on. If there is only one variable, its index is 1.

2. Execute the call to the stored procedure - Execute the procedure using the execute() method.

3. Get the OUT parameters - Once the procedure is completed, you retrieve OUT variables, if any, using the getXXX() methods. Note that these methods must match the types you registered in the previous slide.

Stored Procedure and the ordertable

```
SQL Plus

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> select * from ordertable;

    ORDNO ORDDATE   CUSTOMER                AMT
---------- --------- --------------- ----------
     3659 05-SEP-13 Arjun                 23000
     2340 23-MAY-11 Sumit                 12000
     8899 12-JUL-13 Meena                 56000
     7782 11-JAN-13 William               39000

SQL> select text from  user_source where name='LASTORDERNUMBER';

TEXT
--------------------------------------------------------------------------------

procedure LastOrderNumber(x out number) is
begin
select max(ordno) into x from ordertable;
end;

SQL>
```

Example on CallableStatement

```java
import java.sql.*;
class ExCallable{
  public static void main (String args[]){
  new ExCall();
  }
}
class ExCall {
  ExCall() {
    int lastOrderNumber;
    Connection conn;
    try {
        ConnectionClass x = new ConnectionClass();
        conn=x.connectionFactory();
        String query ="{ CALL LastOrderNumber (?)}" ;
```
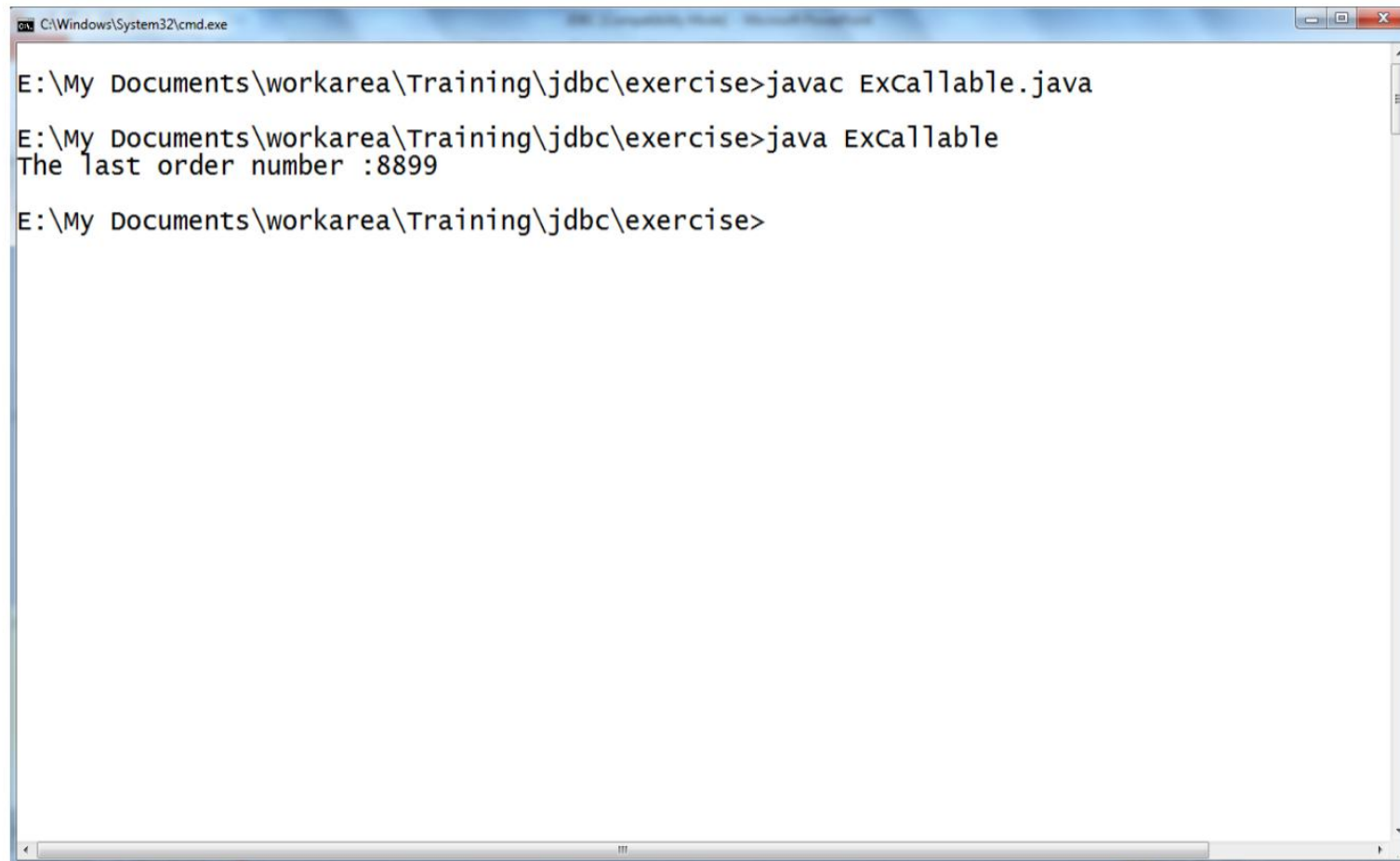
Example on CallableStatement (Contd..)

```
CallableStatement cstatement = conn.prepareCall(query);

cstatement.registerOutParameter(1, Types.INTEGER);

cstatement.execute();

lastOrderNumber = cstatement.getInt(1);

System.out.println("The last order number :"+lastOrderNumber);

cstatement.close();

conn.close();

  }

  catch (SQLException error) {

    System.out.println("Cannot connect to database "+ error);

  }

 }

}
```

Example on CallableStatement (Contd.).

```
C:\Windows\System32\cmd.exe

E:\My Documents\workarea\Training\jdbc\exercise>javac ExCallable.java

E:\My Documents\workarea\Training\jdbc\exercise>java ExCallable
The last order number :8899

E:\My Documents\workarea\Training\jdbc\exercise>
```

Using Transactions

With JDBC drivers new connections are, by default , in auto commit mode

- Turn Auto Commit Off :
    - Use *conn.setAutoCommit(false);*

- To control transactions, when you are not in autocommit mode:
    - *conn.commit();* Commit a transaction
    - *conn.rollback();* Roll back a transaction

- Turn Auto Commit On :
    - Use *conn.setAutoCommit(true);*

Using Transactions

Transactions with JDBC

- With JDBC, database transactions are managed by using the Connection object. When you create a Connection object, by default in autocommit mode, meaning that, after executing every statement by default it is committed.

- You can change the autocommit mode by using setAutoCommit() method. Here is a full description of autocommit mode:

- If a connection is in autocommit mode, all its SQL statements will be executed and committed as individual transactions.

- If a statement returns a result set, the statement completes when the last row of the result set has been retrieved, or the result set has been closed.

Using Transactions

- If autocommit mode has been disabled, its SQL statements are grouped into transactions, which must be terminated by calling either commit() or rollback(). commit() makes permanent all changes since the previous commit or rollback and releases any database locks held by the connection.

- rollback() drops all changes since the previous commit or rollback and releases any database locks. commit() and rollback() should only be called when in non-autocommit mode.

Quiz

1. Which of the following method has to be invoked on the Connection object to get CallableStatement object ?

    a) prepareCall

    b) prepareStatement
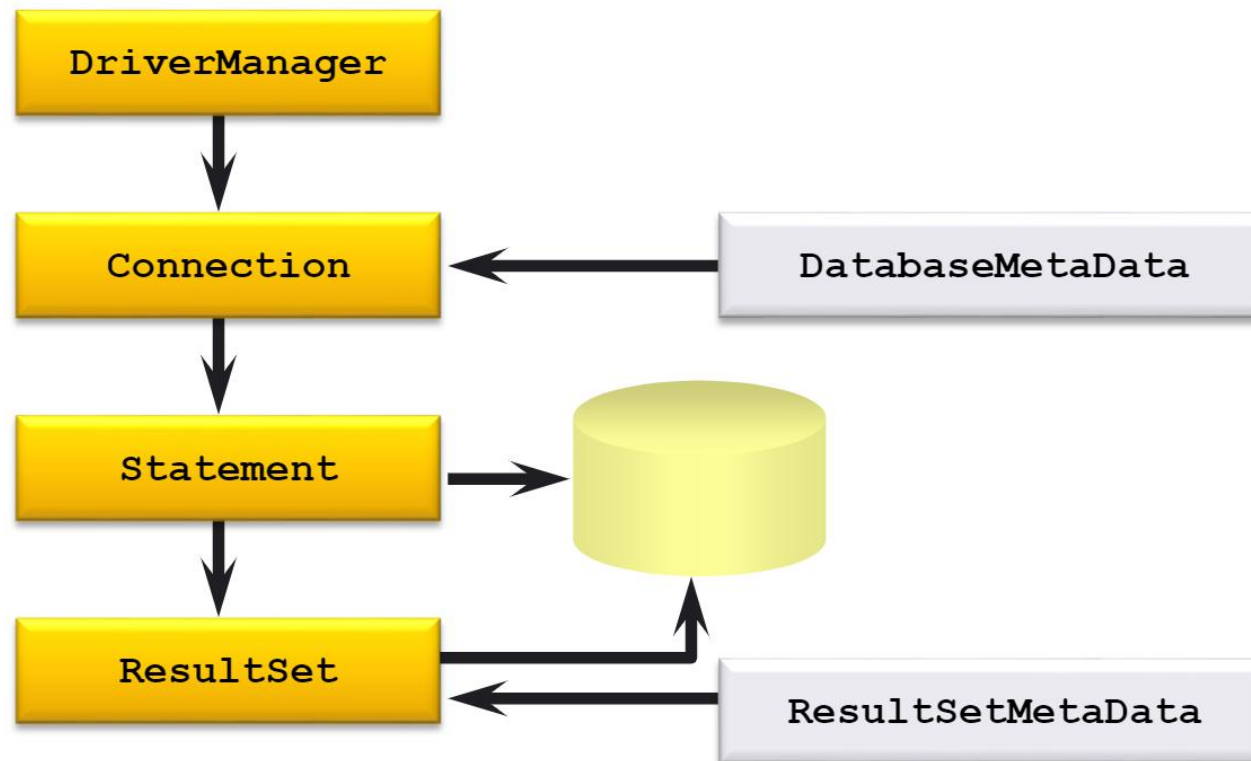
    c) createCallable

    d) prepareCallable

2. Which of the following statements can be used to turn autocommit off (where conn is the reference to the Connection object) ?

    a) conn.setAutoCommit("true");

    b) conn.setAutoCommit(true);

    c) conn.setAutoCommit("false");

    d) conn.setAutoCommit(false);

    .

Answers :

1 : a
2 : d

Summary of JDBC Classes

Summary of JDBC Classes

- **DriverManager** - DriverManager provides access to registered JDBC drivers. DriverManager hands out connections to a specified data source through its getConnection() method.

- **Connection** - The Connection class is provided by the JDBC driver, as are all subsequent classes mentioned. A Connection object represents a session with a database and is used to create a Statement object, using Connection.createStatement().

- **Statement** - The Statement class executes SQL statements. For example, queries can be executed using the executeQuery() method and the results are wrapped up in a ResultSet object.

Summary of JDBC Classes

- **ResultSet -** JDBC returns the results of a query in a ResultSet object. A ResultSet object maintains a cursor pointing to its current row of data. The next() method moves the cursor to the next row. The ResultSet class has getXXX() methods to retrieve the columns in the current row.

- **DatabaseMetaData and ResultSetMetaData -** The DatabaseMetaData and ResultSetMetaData classes return metadata about the database and ResultSet, respectively. Call getMetaData() on the Connection object or the ResultSet object.

Summary

In this module, you were able to:

- Invoke stored procedures through CallableStatement object

- Understand the function of commit and roll back in transactions

Thank You