# Stored procedures used in callable and prepared statements

Explore the execution of Stored Procedures using callable statement with the Informix JDBC driver

To improve the performance of a relational database application, it is normal to use stored procedures, which are nothing but user-defined functions containing a series of SQL statements stored in database. In this article, we are going to demonstrate how to access a stored procedure using callable statements with the Informix® JDBC driver. JDBC CallableStatement extends the PreparedStatement and sends the request to server to execute a stored procedure for complex operations, such as returning values or using input and output stored procedure parameters.

## Introduction

In a relational database application, the main advantage of using stored procedures over single SQL statements is that the query plan (or execution plan) is generated when the stored procedure is created, and the same query plan is reused with every execution of the stored procedure, saving a considerable amount of resources in the database server. When created once, they can be called by any database client, such as JDBC applications, as many times as it wants without the need for a new execution plan.

How to use stored procedures varies from one database server to another. A Database Management System (DBMS) such as Informix and DB2®, have different SQL syntax to execute stored procedures. This makes things difficult for application developers when they need to write code targeted to several DBMSes. A callable statement provides a method to execute stored procedures using the same SQL syntax in all DBMS systems.

## Why use stored procedures?

Suppose we have a JDBC application that needs to efficiently repeat a sequence of tasks again and again. We might think of using a Java™ method, but how many times do we want to do client/server communication to send and receive data? The database server will prepare and generate a query plan for every SQL statement sent by the application, which will consume some CPU time. While taking performance into consideration, using simple Java methods with single SQL statements may be a bad idea.

How about using a stored procedure, which is just a one-time task? Create a single SQL call, and you can call it from the JDBC application using a CallableStatement object, which acts as caller to the stored procedure on the server. Most of the business logic will reside on the stored procedure. This will help simplify the client code and will speed up the execution because the SQL statements included in the stored procedure were prepared and optimized when the stored procedure was created.

## Calling stored procedures in JDBC applications

The Informix JDBC driver provides the Statement, PreparedStatement, and CallableStatement methods, which can be used to execute stored procedures. Which method you use depends on the characteristic of the stored procedure. For example, if the stored procedure returns a single value, you should use a JDBC Statement object. The following table provides some guidelines for what method to use for which stored procedure type.

##### Table 1. Table shows what JDBC method to use based on the respective stored procedure type

| **Stored procedure type** | **JDBC method** |
| --- | --- |
| Stored procedure requires no IN or OUT parameters | Use a Statement object |
| Stored procedure with IN parameters | Use a PreparedStatement object |
| Stored procedure with IN and OUT parameters | Use a CallableStatement object |

We are going to provide a sample of executing a stored procedure using the Informix JDBC methods in following cases:

* Using Statement having no parameters
* Using PreparedStatement having Input parameters
* Using CallableStatement having output parameters
* Named parameters in a CallableStatement
* Overloaded stored procedures

Each topic mentioned above will be having the following details:

* Syntax used to call the stored procedure within the Informix JDBC driver procedure
* The schema of the stored procedure on the database
* JDBC driver sample program with the output

## Using Statement object having no parameters

The syntax used for executing a stored procedure without IN or OUT parameters is shown in Listing 1.

##### Listing 1. Syntax used for executing a stored procedure without IN or OUT parameters

{call procedure-name}

There is no need to use the CallableStatement object with this type of stored procedure; you can use a simple JDBC statement. The code in Listing 2 shows the definition of the stored procedure using Informix SQL. The following example, GetCustName, returns a single result set that contains one column of data, which is a combination of the first name and last name of the first five contacts in the customer table.

##### Listing 2. Definition of the stored procedure using Statement object having no parameters

CREATE PROCEDURE GETCUSTNAME() RETURNING LVARCHAR AS NAME;

DEFINE W\_NAME LVARCHAR;

FOREACH

SELECT FIRST 5 FNAME || ' ' || LNAME INTO W\_NAME

FROM CUSTOMER

RETURN W\_NAME WITH RESUME;

END FOREACH;

END PROCEDURE;

**NOTE:** This stored procedure sample (and all the others in this article) uses the tables created in the demonstration database stores\_demo, which is a selectable option during the Informix IDS installation.

The JDBC Java code for the execution of the stored procedure is as shown in Listing 3.

##### Listing 3. JDBC code using Statement object having no parameters

public static void executeStoredprocNoParams(Connection con) {

try {

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("{call GETCUSTNAME}");

while (rs.next()) {

System.out.println(rs.getString("Name"));

}

rs.close();

stmt.close();

}

catch (Exception e) {

e.printStackTrace();

}

}

Because the stored procedure returns a result set, we must use the Statement.executeQuery() method and fetch through the results using ResultSet.next(). Listing 4 shows the output of the following program.

##### Listing 4. Program output from Listing 3

$java sample\_Stored procedure\_1

Ludwig Pauli

Carole Sadler

Philip Currie

Anthony Higgins

Raymond Vector

George Watson

$

## Using PreparedStatement having input parameters

This is the most common of all stored procedures. A stored procedure requires parameters to pass data into the stored procedure for internal processing. You should use a PreparedStatement to deal with this type of stored procedure. The syntax is shown in Listing 5.

##### Listing 5. Syntax used for execution of stored procedures using prepared statements having input parameters

{call procedure-name(?,?,...)}

While you can use the Informix SQL syntax to execute the stored procedure (e.g., execute procedure procedure\_name(?,?)), it is recommended to stick with the SQL escape sequence syntax.

The question mark characters (?) correspond to each of the input parameters required by the stored procedure. It acts as a placeholder for the values passed to the stored procedure.

To specify a value for a parameter, you can use one of the setter methods of the IfxPreparedStatement class. (e.g., pstmt.setInt()). The setter method you can use is determined by the data type of the IN parameter. In addition to the value of the parameter, the setter method takes the position of the parameter in the SQL statement.

##### Listing 6. Example showing a stored procedure with a signature

UPDMANU(INT MN\_CODE, CHAR(10) MN\_NAME, DATE MN\_UPD);

##### Listing 7. Java code snippet should be used in case of example shown in Listing 6

pstmt = con.prepareStatement("{call updmanu(?,?,?)}");

pstmt.setInt(1, manu\_id);

pstmt.setString(2, manu\_code);

pstmt.setDate(3, manu\_date);

Listing 8 shows the following stored procedure is used to demonstrate how to use IN parameters.

##### Listing 8. Stored procedure example demonstrates how to use IN parameters

CREATE PROCEDURE GETORDERS(CUS\_ID INT) RETURNING INT AS ORDER\_NUM,

DATE AS ORDER\_DATE, LVARCHAR AS SHINSTRUC;

DEFINE W\_ORDERN INT;

DEFINE W\_ORDERD DATE;

DEFINE W\_SHIP LVARCHAR;

FOREACH

SELECT ORDER\_NUM,ORDER\_DATE,SHIP\_INSTRUCT

INTO W\_ORDERN,W\_ORDERD,W\_SHIP

FROM ORDERS WHERE ORDERS.CUSTOMER\_NUM=CUS\_ID

RETURN W\_ORDERN,W\_ORDERD,W\_SHIP WITH RESUME;

END FOREACH;

END PROCEDURE;

This stored procedure accepts a single input parameter named CUS\_ID, which is an integer value, and it returns a list of orders for that customer\_id. The Java code for calling this stored procedure is shown in Listing 9.

##### Listing 9. Java code for execution of stored procedure using prepared statements having input parameters

public static void executeSprocInParams(Connection con, int c\_id) {

try {

PreparedStatement pstmt = con.prepareStatement("{call getorders(?)}");

pstmt.setInt(1, c\_id);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

System.out.println("Order Number\t: "+rs.getString("order\_num"));

System.out.println("Order Date\t: "+rs.getString("order\_date"));

System.out.println("Instructions\t: "+rs.getString("shinstruc"));

System.out.println();

}

rs.close();

pstmt.close();

}

catch (Exception e) {

e.printStackTrace();

}

}

Executing the executeSprocInParams() method, we can see the output as shown in Listing 10.

##### Listing 10. Output of the program shown in Listing 9

$java sample\_Stored procedure\_2

Order Number : 1001

Order Date : 2008-05-20

Instructions : express

Order Number : 1003

Order Date : 2008-05-22

Instructions : express

Order Number : 1011

Order Date : 2008-06-18

Instructions : express

Order Number : 1013

Order Date : 2008-06-22

Instructions : express

$

## Using CallableStatement having output parameter

If the stored procedure requires the use of IN or OUT parameters, you need to use a JDBC CallableStatement to handle the parameters. Only the IfxCallableStatement class (which extends from the Java CallableStatement) can deal with IN and OUT parameters.

Next, we'll demonstrate how to call a stored procedure that returns one or more OUT parameters. These are the parameters that the stored procedure uses to return data to the calling application as single values, not as a result set as we saw earlier. The SQL syntax used for IN/OUT stored procedures is similar to what we showed before in [Listing 5](http://www.ibm.com/developerworks/data/library/techarticle/dm-1209storedprocedures/index.html?ca=drs#listing5).

##### Listing 11. Syntax used for execution of stored procedures using callable statements having output parameters

{call procedure-name(?,?,...)}

You must follow the correct order for the parameter (IN and OUT). Values for OUT parameters must be registered using the registerOutParameter() method of the CallableStatement class. Each OUT parameter must be specified in the correct order. As with the IN parameter, the first parameter of this method is the ordinal (or position) for the parameter —cstmt.registerOutParameter(2, Types.INTEGER);, for example.

The value you specify for the OUT parameter in the registerOutParameter method must be one of the Informix JDBC data types contained in java.sql.Types, which is internally converted to one of the native IDS data types.

For this example, we are going to use the following stored procedure, which uses the "items" table from the stores\_demo database, as shown in Listing 12.

##### Listing 12. Stored procedure used to demonstrate how to use OUT parameters

CREATE PROCEDURE GETTOTAL(ORDER\_ID INT, OUT TOTALPRICE MONEY);

LET TOTALPRICE=(SELECT SUM(TOTAL\_PRICE) FROM ITEMS WHERE ORDER\_NUM=ORDER\_ID);

END PROCEDURE;

This stored procedure returns a single OUT parameter (TotalPrice), which is an integer, based on the specified IN parameter (Order\_ID), which is also an integer. The value returned in the OUT parameter is the sum of all the items from a specific order number contained in the items table.

##### Listing 13. Java code for execution of stored procedure using callable statements having output parameters

public static void executeStoredProcOUTParams(Connection con,int o\_id) {

try {

CallableStatement cstmt = con.prepareCall("{call GetTotal(?, ?)}");

cstmt.setInt(1, o\_id);

cstmt.registerOutParameter(2, Types.INTEGER);

cstmt.execute();

System.out.println("Total price for order" + o\_id +"is $"+cstmt.getInt(2));

}

catch (Exception e) {

e.printStackTrace();

}

}

## Named parameters in a CallableStatement

In the previous example, we used the position to identify each one of the parameters in the stored procedure. You can identify parameters by name, making the application code cleaner and easier to read.

The following example demonstrates how to use named parameters in a Java application. Note that parameter names correspond to the parameter names in the stored procedure's definition.

##### Listing 14. How to use named parameters in a Java application

public static void executeStoredProcOUTParams(Connection con,int o\_id) {

try {

CallableStatement cstmt = con.prepareCall("{call GetTotal(?, ?)}");

cstmt.setInt("Order\_ID", o\_id);

cstmt.registerOutParameter("TotalPrice", Types.INTEGER);

cstmt.execute();

System.out.println("Total price for order"+ o\_id +"is $"+cstmt.getInt("TotalPrice"));

}

catch (Exception e) {

e.printStackTrace();

}

}

Parameters have to be indicated by index or name; you can't mix both methods. These two Java samples produce the following output printing the total price for a specified order:

##### Listing 15. Output of two java samples in Listing 13 and 14

$java sample\_Stored procedure\_3

Total price for order 1002 is $1200

$

**NOTE:** These examples use the execute() method of the CallableStatement class to run the stored procedure. This is used because the stored procedure did not return a result set. If it did, the executeQuery() method should be used, as in the following example.

##### Listing 16. Stored procedure used to illustrate when to use executeQuery() method

CREATE PROCEDURE GETTOTALBYMANU(CODE CHAR(3), OUT TOTAL MONEY)

RETURNING CHAR(3) AS MANU\_CODE, CHAR(10) AS MANU\_NAME;

DEFINE W\_MANU\_CODE CHAR(3);

DEFINE W\_MANU\_NAME CHAR(10);

LET TOTAL=(SELECT SUM(TOTAL\_PRICE) FROM ITEMS WHERE MANU\_CODE=CODE);

SELECT MANU\_CODE,MANU\_NAME

INTO W\_MANU\_CODE,W\_MANU\_NAME FROM MANUFACT WHERE MANU\_CODE=CODE;

RETURN W\_MANU\_CODE,W\_MANU\_NAME;

END PROCEDURE;

The method in Listing 17 uses the executeQuery() to invoke the GetTotalByManu stored procedure.

##### Listing 17. Java code used to illustrate use of executeQuery() method

public static void executeStoredProcOUTParamsResulset(Connection con,String manu\_id) {

try {

CallableStatement cstmt = con.prepareCall("{ call gettotalbymanu(?,?)}");

cstmt.setString(1, manu\_id);

cstmt.registerOutParameter(2, Types.CHAR);

ResultSet rs = cstmt.executeQuery();

rs.next();

System.out.println("Total for manufacturer '"+rs.getString(2).trim()+

" ("+rs.getString(1)+") ' is $"+cstmt.getInt(2));

}

catch (Exception e) {

e.printStackTrace();

}

}

The output of the program shown in Listing 17 is shown in Listing 18.

##### Listing 18. Program output for Java example in Listing 17

$java sample\_Stored procedure\_4

Total for manufacturer 'Hero (HRO)' is $2882

$

**NOTE:** If you don't know how the stored procedure was defined, you can use the JDBC metadata routines to get information about the stored procedure, such as the name and type of the parameters it takes.

The following example uses the getProcedureColumns() method to get the name and the type of the gettotalbymanu procedure.

##### Listing 19. Java code

public static void executeStoredGetOutParams(Connection con,String procname) {

try {

DatabaseMetaData dbmd = con.getMetaData();

ResultSet rs = dbmd.getProcedureColumns("stores7","",procname.toUpperCase(),null);

while (rs.next())

if (rs.getShort("COLUMN\_TYPE")==DatabaseMetaData.procedureColumnOut) {

System.out.println("OUT Parame: "+ rs.getString("COLUMN\_NAME"));

System.out.println("Type: "+rs.getString("DATA\_TYPE") );

}

}

catch (Exception e) {

e.printStackTrace();

}

}

Alternately, you can check if the stored procedure has OUT parameters using the CallableStatement.hasOutParameter() method. If it was defined with OUT parameters, it will return TRUE, as shown in Listing 20.

##### Listing 20. Sample Java code

CallableStatement cstmt = con.prepareCall("{ call gettotalbymanu(?,?)}");

if (((IfxCallableStatement) cstmt).hasOutParameter())

System.out.println("Stored procedure has OUT parameters ");

// perform the logic

## Calling stored procedures with multiple signatures

The Informix database server supports the overloaded stored procedures. You can have stored procedures with the same name, but different parameters (or signatures) to perform different operations based on the parameter it takes. A basic example could be these two procedures as shown in [Listing 21](http://www.ibm.com/developerworks/data/library/techarticle/dm-1209storedprocedures/index.html?ca=drs#listing21) and [Listing 22](http://www.ibm.com/developerworks/data/library/techarticle/dm-1209storedprocedures/index.html?ca=drs#listing22).

##### Listing 21. Stored procedure definition No. 1

CREATE PROCEDURE GETORDERS(CUS\_ID INT) RETURNING INT

AS ORDER\_NUM, DATE AS ORDER\_DATE, LVARCHAR AS SHINSTRUC;

DEFINE W\_ORDERN INT;

DEFINE W\_ORDERD DATE;

DEFINE W\_SHIP LVARCHAR;

FOREACH

SELECT ORDER\_NUM,ORDER\_DATE,SHIP\_INSTRUCT

INTO W\_ORDERN,W\_ORDERD,W\_SHIP

FROM ORDERS WHERE ORDERS.CUSTOMER\_NUM=CUS\_ID

RETURN W\_ORDERN,W\_ORDERD,W\_SHIP WITH RESUME;

END FOREACH;

END PROCEDURE;

##### Listing 22. Stored procedure definition No. 2

CREATE PROCEDURE GETORDERS(ORD\_DATE DATE) RETURNING INT

AS ORDER\_NUM, DATE AS ORDER\_DATE, LVARCHAR AS SHINSTRUC;

DEFINE W\_ORDERN INT;

DEFINE W\_ORDERD DATE;

DEFINE W\_SHIP LVARCHAR;

FOREACH

SELECT ORDER\_NUM,ORDER\_DATE,SHIP\_INSTRUCT

INTO W\_ORDERN,W\_ORDERD,W\_SHIP

FROM ORDERS WHERE ORDERS.ORDER\_DATE=ORD\_DATE

RETURN W\_ORDERN,W\_ORDERD,W\_SHIP WITH RESUME;

END FOREACH;

END PROCEDURE;

They both have the same name (GETORDERS), but the first one uses an INT parameter to get the orders for a specific customer, and the second has a DATE parameter to return the orders for a specific date, as shown in Listing 23.

##### Listing 23. Example for multiple signatures

CREATE PROCEDURE GETORDERS(CUS\_ID INT) RETURNING INT

AS ORDER\_NUM, DATE AS ORDER\_DATE, LVARCHAR AS SHINSTRUC;

CREATE PROCEDURE GETORDERS(ORD\_DATE DATE) RETURNING INT

AS ORDER\_NUM, DATE AS ORDER\_DATE, LVARCHAR AS SHINSTRUC;

To execute these stored procedures from a JDBC application, you must provide the parameter type in the SQL syntax so the Informix engine will know which stored procedure you want to run. Use the ::datatype prefix in the placeholder, as shown in Listing 24.

##### Listing 24. Examples for placeholder

{call getorders(?::INT)}

{call getorders(?::DATE)}

Listing 25 shows how to execute the GETORDERS(DATE) procedure.

##### Listing 25. Java code to demonstrate how to use stored procedure with multiple signatures

public static void executeSprocInParams\_date(Connection con, String o\_date) {

try {

PreparedStatement pstmt = con.prepareStatement("{call getorders(?::DATE)}");

pstmt.setString(1, o\_date);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

System.out.println("Order Number\t: "+rs.getString("order\_num"));

System.out.println("Order Date\t: "+rs.getString("order\_date"));

System.out.println("Instructions\t: "+rs.getString("shinstruc"));

System.out.println();

}

rs.close();

pstmt.close();

}

catch (Exception e) {

e.printStackTrace();

}

}

## Conclusion

This article has shown various ways to access simple and complex stored procedures from a JDBC application. The links to the server and JDBC documentation in the [Resources](http://www.ibm.com/developerworks/data/library/techarticle/dm-1209storedprocedures/index.html?ca=drs#resources) section will provide assistance in exploring on the code examples to help you build efficient stored procedures and the equivalent JDBC application. The understanding gained from this article should help you transfer complex business logic into stored procedures and export them from your JDBC application.