Stored Procedures in Java (SPJs) Guide

Version 2.1.0

Table of Contents

License Statement

Disclaimer: *Apache Trafodion is an effort undergoing incubation at the Apache Software Foundation (ASF), sponsored by the Apache Incubator PMC. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF. <<<*

Acknowledgements

Revision History

Version	Date
2.1.0	TBD
2.0.1	July 7, 2016
2.0.0	June 6, 2016
1.3.0	January, 2016

Chapter 1. About This Document

This guide describes how to develop, deploy, and manage Stored Procedures in Java (SPJs) on Trafodion.

1.1. Intended Audience

This manual is intended for application programmers who are writing and compiling Java code for stored procedures and for database administrators who are deploying and managing Stored Procedures in Java (SPJs) on Trafodion. The reader should know:

- The Java programming language.
- JDBC and the Trafodion JDBC Type-4 Driver.
- Structured Query Language (SQL) and database terms and concepts.

Although not required, it helps to be familiar with the part of the ANSI SQL/Foundation standard called SQL/JRT (Java Routines and Types) on which this implementation of stored procedures is based.

1.2. Document Organization

This document is organized as follows:

Chapter	Description
Introduction	Defines what an SPJ is, describes the benefits of using SPJs on the database, and lists steps for developing and deploying SPJs on Trafodion.
Get Started	Describes the software requirements for using SPJs on Trafodion.
Develop SPJ Methods	Provides guidelines for writing and compiling a Java method to be used as the body of a stored procedure.
Deploy SPJ JAR Files	Explains how to deploy SPJ JAR files on Trafodion.
Create SPJs	Explains how to create, drop, and alter an SPJ on Trafodion.
Grant Privileges	Explains how to grant and revoke privileges for executing SPJs on the Trafodion database.
Execute SPJs	Explains how to execute an SPJ on Trafodion by using the CALL statement.
Performance and Troubleshooting	Describes how to improve and monitor the performance of SPJs and provides guidelines for troubleshooting common problems.
Appendix A: Sample SPJs	Provides examples of SPJ methods that demonstrate business logic on Trafodion.
Appendix B: Sample Database	Describes the sample database on which the Sample SPJs operate.

1.3. New and Changed Information

This is a new manual.

1.4. Notation Conventions

This list summarizes the notation conventions for syntax presentation in this manual.

- UPPERCASE LETTERS

  Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required.

  ```
  SELECT
  ```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

  ```
  file-name
  ```

- [ ] Brackets

  Brackets enclose optional syntax items.

  ```
  DATETIME [start-field TO] end-field
  ```

  A group of items enclosed in brackets is a list from which you can choose one item or none.

  The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines.

  For example:

  ```
  DROP SCHEMA schema [CASCADE]
  DROP SCHEMA schema [ CASCADE | RESTRICT ]
  ```

- { } Braces

  Braces enclose required syntax items.

```
FROM { grantee [, grantee ] ... }
```

  A group of items enclosed in braces is a list from which you are required to choose one item.

  The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines.

  For example:

```
INTERVAL { start-field TO end-field }
{ single-field }
INTERVAL { start-field TO end-field | single-field }
```

- | Vertical Line

  A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

```
{expression | NULL}
```

- … Ellipsis

  An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times.

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

  An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times.

  For example:

```
expression-n ...
```

- Punctuation

  Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown.

```
DAY (datetime-expression)
@script-file
```

  Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown.

  For example:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

  Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma.

```
DAY (datetime-expression) DAY(datetime-expression)
```

  If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
myfile.sh
```

- Line Spacing

  If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line.

  This spacing distinguishes items in a continuation line from items in a vertical list of selections.

```
match-value [NOT] LIKE _pattern
   [ESCAPE esc-char-expression]
```

1.5. Comments Encouraged

We encourage your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to user@trafodion.incubator.apache.org.

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document.

Chapter 2. Introduction

This chapter introduces stored procedures in Java (SPJs) in Trafodion and covers these topics:

2.1. What Is an SPJ?

A stored procedure is a type of user-defined routine (UDR) that operates within a database server and typically performs SQL operations on a database. The database server contains information about the stored procedure and controls its execution. A client application executes a stored procedure by issuing an SQL CALL statement. Unlike a user-defined function, which returns a value directly to the calling application, a stored procedure returns each output value to a dynamic parameter in its parameter list or returns a set of values to a result set array.

Trafodion supports stored procedures written in the Java programming language. The Trafodion implementation of stored procedures complies mostly, unless otherwise specified, with SQL/JRT (Java Routines and Types), which extends the ANSI SQL/Foundation standard. A stored procedure in Java (SPJ) is a Java method contained in a Java archive (JAR) file on Trafodion, registered in the database, and executed by the database engine when a client application issues a CALL statement.

The body of a stored procedure consists of a public, static Java method that returns void. These Java methods, called *SPJ methods*, are contained in classes within JAR files on the cluster hosting Trafodion.

An SPJ method must be registered as a stored procedure in the database before a client application can execute it with a CALL statement. You upload the SPJ to the cluster where Trafodion is running and then you register the SPJ as a library object using the CREATE LIBRARY statement. Next, you register the library object using the CREATE PROCEDURE statement.

2.2. Benefits of SPJs

SPJs provide an efficient and secure way to implement business logic in the database. SPJs offer these advantages:

- Java Methods Callable From SQL
- Common Packaging Technique
- Security
- Increased Productivity
- Portability

2.2.1. Java Methods Callable From SQL

With support for SPJs, Java methods are callable from any client application that connects to Trafodion. For example, you can invoke the same SPJ method from JDBC client applications and ODBC client applications. By using the database engine to invoke Java methods, you can extend the functionality of the database and share business logic among different applications.

For more information, see Execute SPJs.

2.2.2. Common Packaging Technique

Different applications can invoke the same SPJ to perform a common business function. By encapsulating business logic in an SPJ, you can maintain consistent database operations and avoid duplicating code in applications.

Applications that call SPJs are not required to know the structure of the database tables that the SPJ methods access. The application does not need to use any table or column names; it needs only the name of the stored procedure in the CALL statement. If the table structure changes, you might need to change the SPJ methods but not necessarily the CALL statements within each application.

2.2.3. Security

By using SPJs, you can conceal sensitive business logic inside SPJ methods instead of exposing it in client applications. You can also grant privileges to execute an SPJ to specific users and restrict the privileges of other users. For more information, see Grant Privileges

2.2.4. Increased Productivity

Use SPJs to reduce the time and cost of developing and maintaining client applications. By having several applications call the same SPJ, you need only change the SPJ method once when business rules or table structures change instead of changing every application that calls the SPJ.

Using the Java language to implement stored procedures increases productivity. Given the popularity of the Java language, you can leverage the existing skill set of Java programmers to develop SPJs.

The portability of the Java language enables you to write and compile Java class files for SPJs once and deploy them anywhere.

2.2.5. Portability

Because SPJ methods are written in Java, and SPJs conform to the ANSI SQL standard, SPJs are portable across different database servers. With minimal changes to SPJ methods, you can port existing SPJ JAR files from another database server to Trafodion and register the methods as stored procedures in a Trafodion database. You can also port client applications that call SPJs in other databases to Trafodion SQL with minimal changes to the CALL statements in the application.

2.3. Use SPJs

To use SPJs in Trafodion:

1. Verify that you have the required software installed on the client workstation. See Get Started.
2. Develop a Java method to be used as an SPJ:
   a. Write a static Java method:

      Create a file named `Payroll.java` with the following content:

```
import java.sql.* ;
import java.math.* ;

public class Payroll
{
   public static void adjustSalary( BigDecimal empNum
                                  , double percent, BigDecimal[] newSalary
                  ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement setSalary =
         conn.prepareStatement( "UPDATE trafodion.persnl.employee "
                           + "SET salary = salary * (1 + (? / 100)) "
                  + "WHERE empnum = ?"
                  ) ;

      PreparedStatement getSalary =
         conn.prepareStatement( "SELECT salary "
                           + "FROM trafodion.persnl.employee "
                  + "WHERE empnum = ?"
                  ) ;

      setSalary.setDouble( 1, percent ) ;
      setSalary.setBigDecimal( 2, empNum ) ;
      setSalary.executeUpdate() ;

      getSalary.setBigDecimal( 1, empNum ) ;
      ResultSet rs = getSalary.executeQuery() ;
      rs.next() ;

      newSalary[0] = rs.getBigDecimal( 1 ) ;

      rs.close() ;
      conn.close() ;
   }
}
```

b. Compile the Java source file to produce a class file:

```
$ javac Payroll.java
$
```

c. Package the SPJ class file in a JAR file:

```
jar cvf Payroll.jar Payroll.class
```

If the SPJ class refers to other classes, package the other classes in the same JAR file as the SPJ class:

```
$ jar cvf Payroll.jar Payroll.class other.class
added manifest
adding: Payroll.class(in = 1213) (out= 711)(deflated 41%)
$
```

For details, see Develop SPJ Methods.

3. Deploy the SPJ JAR file on Trafodion by creating a library object for the JAR file in one of the database schemas. For details, see Deploy SPJ JAR Files.
4. As the schema owner, create the SPJ in the database. For details, see Create SPJs.
5. Grant privileges to database users for executing the SPJ and for operating on the referenced database objects. For example, you can issue GRANT statements in an trafci session, as shown below:

```
GRANT EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary
TO "payrolldir1", "payrolldir2"
WITH GRANT OPTION ;

GRANT SELECT, UPDATE (salary)
ON TABLE trafodion.persnl.employee
TO "payrolldir1", "payrolldir2"
WITH GRANT OPTION ;
```

For details, see Grant Privileges.

6. Execute an SPJ by using a CALL statement in a client application. For example, you can issue a CALL statement in an trafci session, as shown below, or in a JDBC or ODBC client application:

```
SQL> CALL trafodion.persnl.adjustsalary( 29, 2.5, ? ) ;

NEWSALARY
------------
   139400.00

--- SQL operation complete.
```

   For details, see Execute SPJs.

7. Monitor the performance of SPJs and resolve common problems with SPJs in the database. See Performance and Troubleshooting.

Chapter 3. Get Started

Before you can start using SPJs on Trafodion, verify that you have the required software installed on the client workstation. Trafodion is delivered to you ready to use and pre-configured with the software required to support SPJs.

3.1. Required Client Software

3.1.1. Java Development Kit

To develop Java methods to be used as SPJs, you must have a Java Development Kit (JDK) installed on the client workstation. To download a JDK, go to http://www.oracle.com/technetwork/java/index.html

The version of the JDK that you download and use on the client workstation should be the same as or lower than the Java version running on Trafodion. To check the Java version that is running in Trafodion, use this approach:

- Launch `trafci` on Trafodion, and run the `LOCALHOST` or `lh java -version` command. (To use the on-platform trafci client, see the Trafodion Command Interface Guide.

  For example:

```
SQL> lh java -version

java version "1.6.0_06"
Java(TM) SE Runtime Environment (build 1.6.0_06-b02)
Java HotSpot(TM) Client VM (build 10.0-b22, mixed mode)

SQL>
```

  In this example, the returned Java version indicates that the Trafodion instance supports Java SE 6 or JDK 6 (1.6.0), or earlier versions of the JDK.

If you plan to install the Trafodion JDBC Type-4 Driver on the client workstation, you must have JDK 6 (1.6.0) or higher installed on the client workstation.

3.2. Recommended Client Software

3.2.1. Trafodion Command Interface (trafci)

`trafci` is a command-line interface in which you can run SQL statements, such as GRANT PROCEDURE and CALL statements, interactively or from script files. To install `trafci` on a client workstation, see the Trafodion Client Installation Guide.

3.2.2. HP JDBC Type 4 Driver

If you plan to use `trafci`, you must have a compatible version of the Trafodion JDBC Type-4 Driver installed on the client workstation.

To install the JDBC Type-4 driver on the client workstation, see the Trafodion Client Installation Guide.

Chapter 4. Develop SPJ Methods

Before creating, or registering, an SPJ in the database, you must write and compile the Java method to be used as the body of the SPJ. The manual refers to those Java methods as *SPJ methods*.

This chapter requires a familiarity with writing and compiling Java programs.

4.1. Guidelines for Writing SPJ Methods

Follow the guidelines for these topics when you write SPJ methods to be used as SPJs in the database:

4.1.1. Signature of the Java Method

A Java method that you use as an SPJ must have this general signature:

```
public static void myMethodName ( java-parameter-list )
```

Public Access and Static Modifiers

The Java method must be defined as `public` and `static`. If a method is `private` or `protected`, then Trafodion is unable to find the Java method when you try to register the SPJ and returns an error. The Java method must be defined as `static` so that the method can be invoked without having to instantiate its class.

Void Return Type

The return type of the Java method must be `void`. The method must not return a value directly to the caller.

Java Parameters

Except for result sets, which are described in Returning Stored Procedure Result Sets, the parameter types in the Java signature must correspond to the SQL parameters of the stored procedure that you are planning to create. For type mappings, see the table below.

Table 1. Mapping of Java Data Types to SQL Data Types

SQL Data Type	Maps to Java Data Type…
CHAR[ACTER] CHAR[ACTER] VARYING VARCHAR PIC[TURE] X[1] NCHAR NCHAR VARYING NATIONAL CHAR[ACTER] NATIONAL CHAR[ACTER] VARYING	java.lang.string
DATE	java.sql.date
TIME	java.sql.time
TIMESTAMP	java.sql.timestamp
DEC[2] PIC[TURE] S9[3] NUMERIC (including numeric with a precision greater than eighteen)[2]	java.math.bigdecimal
SMALLINT[2]	short
INT[2]	int or java.lang.integer[4]
LARGEINT[2]	long or java.lang.long[4]
FLOAT	double or java.lang.double[4]
REAL	float or java.lang.float[4]
DOUBLE PRECISION	double or java.lang.double[4]

1. Trafodion stores PIC X as a CHAR data type.
2. Numeric data types of SQL parameters must be signed, which is the default in Trafodion.
3. Trafodion stores PIC S9 as a DECIMAL or NUMERIC data type.
4. By default, the SQL data type maps to a Java primitive type. The SQL data type maps to a Java wrapper class only if you specify the wrapper class in the Java signature of the external name clause.

Output parameters in the Java signature must be arrays (for example, `int[]` or `String[]`) that accept only one value in the first element of the array at index 0. For more information, see Returning Output Values From the Java Method.

4.1.2. Returning Output Values From the Java Method

The Java method can return data to the calling application in the form of output parameters or result sets.

Output parameters in the Java signature are parameter arrays that accept only one value in the first element of the array at index 0. The array objects have a fixed size of one element.

You cannot return more than one value to an output parameter of an SPJ. Any attempt to return more than one value to an output parameter results in a Java exception, `ArrayIndexOutOfBounds`.

Result sets in the Java signature are one-element `java.sql.ResultSet[]` arrays that contain ResultSet objects that have multiple rows of data. For more information about result sets, see Returning Stored Procedure Result Sets

This subsection covers these topics related to output parameters:

- Using Arrays for Output Parameters
- Type Mapping of Output Parameters

Using Arrays for Output Parameters

You must use arrays for the output parameters of a Java method because of how Java handles the arguments of a method. Java supports arguments that are passed by value to a method and does not support arguments that are passed by reference. As a result, Java primitive types can be passed only to a method, not out of a method. Because a Java array is an object, its reference is passed by value to a method, and changes to the array are visible to the caller of the method. Therefore, arrays must be used for output parameters in a Java method.

An output parameter accepts only one value in the first element of the array at index 0. Any attempt to return more than one value to an output parameter results in a Java exception, `ArrayIndexOutOfBounds`.

For each output parameter, specify the Java type followed by empty square brackets (`[]`) to indicate that the type is an array. For example, specify an int type as `int[]` for an output parameter in the Java signature.

To return multiple values from a Java method, use an output parameter for each returned value. For example, the `supplierInfo()` method returns a supplier's name, address, city, state, and post code, each as a single string in an output parameter:

The `supplyQuantities()` method returns an average quantity, a minimum quantity, and a maximum quantity to separate output parameters of the integer type:

```
public static void supplyQuantities( int[] avgQty
                                   , int[] minQty
                                   , int[] maxQty
                                   ) throws SQLException
{

...

}
```

For more information about the SPJ examples, see Appendix A: Sample SPJs.

Type Mapping of Output Parameters

When writing an SPJ method, consider how the output of the SPJ is used in the calling application. For output parameters, the Java data type of the SPJ method must map to an SQL data type. See Table 1.

The SQL data type must then map to a compatible data type in the calling application. For the client application programming interfaces (APIs) that support SPJs and for cross-references to the appropriate manuals for type mappings between Trafodion SQL and each API, see Execute SPJs below.

4.1.3. Returning Stored Procedure Result Sets

Trafodion supports SPJs that return stored procedure result sets. A stored procedure result set is a cursor that is left open after the SPJ method executes (that is, after the CALL statement executes successfully). After the CALL statement executes successfully, the calling application can issue requests to open and then retrieve multiple rows of data from the returned result sets.

An SPJ method returns an ordered collection of result sets to the calling application by executing SELECT statements and placing each returned ResultSet object into a one-element Java array of type `java.sql.ResultSet[]`. The `java.sql.ResultSet[]` array is part of the Java method's signature and is recognized by the database engine as a container for a single stored procedure result set.

Place the `java.sql.ResultSet[]` parameters after the other Java parameters, if any, in the Java signature. If you do not place the `java.sql.ResultSet[]` parameters after the other parameters in the signature, the database engine prevents you from creating an SPJ using that Java method. This example shows the declaration of an SPJ method, `orderSummary()`, which returns a maximum of two result sets:

```
public static void orderSummary( java.lang.String onOrAfter
                               , long[] numOrders
                               , java.sql.ResultSet[] orders
                               , java.sql.ResultSet[] detail
                               )
```

This code fragment shows how the `orderSummary()` method returns one of its result sets by executing a SELECT statement and assigning the `java.sql.ResultSet` object to a `java.sql.ResultSet[]` output array:

```
// Open a result set for order num, order info rows
java.lang.String s =
     "SELECT amounts.*, orders.order_date, emps.last_name "
   + "FROM ( SELECT o.ordernum, COUNT( d.partnum ) AS num_parts, "
   + "       SUM( d.unit_price * d.qty_ordered ) AS amount "
   + "       FROM trafodion.sales.orders o, trafodion.sales.odetail d "
   + "       WHERE o.ordernum = d.ordernum "
   + "         AND o.order_date >= CAST(? AS DATE) "
   + "       GROUP BY o.ordernum ) amounts, "
   + "   trafodion.sales.orders orders, trafodion.persnl.employee emps "
   + "WHERE amounts.ordernum = orders.ordernum "
   + "  AND orders.salesrep = emps.empnum "
   + "ORDER BY orders.ordernum "
   ;

java.sql.PreparedStatement ps2 = conn.prepareStatement(s) ;
ps2.setString( 1, onOrAfter ) ;

// Assign the returned result set object to the first element of a
// java.sql.ResultSet[] output array
orders[0] = ps2.executeQuery() ;
```

For the entire example, see ORDERSUMMARY Procedure.

In an SPJ method that returns result sets, do not explicitly close the default connection or the statement object. The database engine closes the connection used to return result sets after it finishes processing the result sets. If you close the connection on which the result sets are being returned, those result sets will be lost, and the calling application will not be able to process them.

An SPJ method can return result sets that contain any data types, except large object (LOB) data. An SPJ method can return a holdable or updatable cursor as a result set. However, Trafodion SQL does not expose those attributes in the calling application. An SPJ method can return a ResultSet object that is a stored procedure result set acquired from a nested CALL statement executed by the SPJ method. However, you are discouraged from nesting CALL statements in SPJ methods. For more information, see Nested Java Method Invocations.

If an SPJ method returns multiple ResultSet objects, the database engine sorts the collection of valid result sets in chronological order according to when the underlying SQL statements were executed. If the number of result sets exceeds the declared maximum for the SPJ, only the first set of result sets up to the maximum number are returned. The database engine discards the other result sets and returns a warning to the calling application.

When an SPJ method returns a ResultSet object through a `java.sql.ResultSet[]` parameter, Trafodion SQL exposes the underlying rows of data as an SQL cursor in the calling application.

If a returned result set is a scrollable cursor, all underlying rows are included in the result set and are available to the calling application. If a returned result set is not scrollable, only those rows not processed by the SPJ method are included in the result set and are available to the calling application. If an SPJ method returns multiple occurrences of the same ResultSet object, the database engine ignores all but one occurrence and makes the underlying rows available to the calling application as a single result set.

For information about processing result sets in different calling applications, see:

- Returning Result Sets in trafci
- Returning Result Sets in an ODBC Client Application
- Returning Result Sets in a JDBC Client Application

4.1.4. Using the main() Method

You can use the `main()` method of a Java class file as an SPJ method. The `main()` method is different from other Java methods because it accepts input values in an array of `java.lang.String` objects and does not return any values in its array parameter.

For example, you can register this main() method as an SPJ:

```
public static void main (java.lang.String [] args)
{

...

}
```

When you register a `main()` method as an SPJ, you can specify zero or more SQL parameters, even though the underlying `main()` method has only one array parameter. All the SQL parameters of the SPJ must have the character string data type, CHAR or VARCHAR, and be declared with the IN mode.

If you specify the optional Java signature, the signature must be (`java.lang.String []`). For more information about

registering an SPJ, see Create SPJs.

4.1.5. Null Input and Output

You can pass a `null` value as input to or output from an SPJ method, provided that the Java data type of the parameter supports nulls. Java primitive data types do not support nulls. However, Java wrapper classes that correspond to primitive data types do support nulls. If a null is input or output for a parameter that does not support nulls, the database engine raises an error condition.

To anticipate null input or output for your SPJ, use Java wrapper classes instead of primitive data types in the method signature.

For example, this Java method uses a Java primitive data type in its signature where no null values are expected:

```
public static void employeeJob( int empNum, Integer[] jobCode )
```

This Java method also uses a Java wrapper class in its signature to anticipate a possible returned null value:

```
public static void employeeJob( int empNum, Integer[] jobCode )
```

4.1.6. Static Java Variables

To ensure that your SPJ method is portable, you should avoid using static variables in the method. The database engine does not ensure the scope and persistence of static Java variables.

4.1.7. Nested Java Method Invocations

An SPJ that invokes another SPJ by issuing a CALL statement causes additional system resources to be used. If you want an SPJ method to call another SPJ method, consider invoking the other Java method directly through Java instead of using a CALL statement. The other Java method should be packaged in the same JAR file as the SPJ method. For more information, see Compiling and Packaging Java Classes.

4.2. Accessing Trafodion

SPJ methods that access Trafodion must be from a Java class that uses JDBC method calls. Follow these guidelines when writing an SPJ method that accesses Trafodion:

4.2.1. Use of java.sql.Connection Objects

Trafodion supports a default connection in an SPJ execution environment, which has a data source URL of `"jdbc:default:connection"`. For example:

```
Connection conn =
   DriverManager.getConnection( "jdbc:default:connection" ) ;
```

`java.sql.Connection` objects that use the `"jdbc:default:connection"` URL are portable to Trafodion from other database management systems (DBMSs).

Closing Default Connections

Trafodion controls default connections in the SPJ environment and closes default connections when they are no longer needed. Therefore, you do not need to use the `close()` method in an SPJ method to explicitly close a default connection when the connection is no longer needed.

If an SPJ method returns result sets, you should not explicitly close the default connection. The database engine closes the connection used to return result sets after it finishes processing the result sets. If an SPJ method closes the connection on which the result sets are being returned, those result sets will be lost, and the calling application will not be able to process them. The JVM does not return an error or warning when the connection is closed.

A default connection that is acquired when an SPJ method executes does not necessarily remain open for future invocations of the SPJ method. Therefore, do not store default connections in static variables for future use.

Default Connection URL

The default connection URL, `"jdbc:default:connection"`, is invalid when the Java method is invoked outside the DBMS, such as when you execute the Java method in a client application. To write an SPJ method that operates in a DBMS, in a client application, or both, without having to change and recompile the code, use the `sqlj.defaultconnection` system property:

```
String s = System.property( "sqlj.defaultconnection" ) ;
if ( s == null )
{
   s = other-url ;
}

Connection c = DriverManager.getConnection( s ) ;
```

The value of `sqlj.defaultconnection` is `"jdbc:default:connection"` in a DBMS and `null` outside a DBMS.

Connection Pooling

Connection pooling, where a cache of database connections is assigned to a client session and reused, is enabled by default in the SPJ environment. The SPJ environment sets the initial connection pool size to `1`, but it does not limit the number of connections an SPJ method can make.

The SPJ environment also sets the minimum connection pool size to 1 so that there is always at least one connection available in the pool. The default settings in the SPJ environment are:

- `maxPoolSize=0`
- `minPoolSize=1`
- `initialPoolSize=1`

To change these settings, use the properties parameter of the `DriverManager.getConnection()` method as shown below:

```
java.util.Properties props = new Properties() ;

props.setProperty( "maxPoolSize", "10" ) ;
props.setProperty( "minPoolSize", "5" ) ;
props.setProperty( "initialPoolSize", "5" ) ;

Connection conn =
   DriverManager.getConnection( "jdbc:default:connection", props ) ;
```

4.2.2. Using JDBC Method Calls

Trafodion uses a JDBC Type-4 driver internally to execute the SQL statements inside an SPJ method. To enable an SPJ to perform SQL operations on a Trafodion database, use JDBC method calls in the SPJ method. The JDBC method calls must be supported by the JDBC Type-4 driver on Trafodion.

For example, if you want the SPJ method to operate on a Trafodion database, use the JDBC API that is supported by Trafodion.

You do not have to explicitly load the JDBC driver before establishing a connection to Trafodion. The database engine automatically loads the JDBC driver when the SPJ is called.

Here is an example of an SPJ method, `adjustSalary()`, that uses JDBC method calls to adjust an employee's salary in the EMPLOYEE table:

```
public class Payroll
{
   public static void adjustSalary( BigDecimal empNum
                                   , double percent
                                   , BigDecimal[] newSalary
                                   ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement setSalary =
         conn.prepareStatement( "UPDATE trafodion.persnl.employee "
                              + "SET salary = salary * (1 + (? / 100)) "
                              + "WHERE empnum = ?"
                              ) ;

      PreparedStatement getSalary =
         conn.prepareStatement( "SELECT salary "
                              + "FROM trafodion.persnl.employee "
                              + "WHERE empnum = ?"
                              ) ;

      setSalary.setDouble( 1, percent ) ;
      setSalary.setBigDecimal( 2, empNum ) ;
      setSalary.executeUpdate() ;

      getSalary.setBigDecimal( 1, empNum ) ;
      ResultSet rs = getSalary.executeQuery() ;
      rs.next() ;

      newSalary[0] = rs.getBigDecimal( 1 ) ;

      rs.close();
      conn.close();
   }
}
```

For other examples of SPJ methods, see Appendix A: Sample SPJs.

4.2.3. Referring to Database Objects in an SPJ Method

In an SPJ method, you can refer to SQL database objects by specifying three-part ANSI names that include the catalog, schema, and object name. For more information about database object names, see the Trafodion SQL Reference Manual.

The database engine propagates the names of the catalog and schema where the SPJ is registered to the SPJ environment. By default, database connections created in the SPJ method are associated with that catalog and schema, meaning that unqualified database objects with one-part or two-part names in the SPJ method are qualified with the same catalog and/or schema name as the SPJ. For example, this SPJ method, which is registered as an SPJ in the

TRAFODION.SALES schema, refers to the unqualified database object, ORDERS:

```
public static void numDailyOrders( Date date
                                 , int[] numOrders
                 ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getNumOrders =
      conn.prepareStatement( "SELECT COUNT( order_date ) "
                           + "FROM orders "
               + "WHERE order_date = ?"
               ) ;

   getNumOrders.setDate( 1, date ) ;

   ResultSet rs = getNumOrders.executeQuery() ;
   rs.next() ;

   numOrders[0] = rs.getInt( 1 ) ;

   rs.close() ;
   conn.close() ;

}
```

In the SPJ environment, the ORDERS table is qualified by default with the same catalog and schema as the SPJ, TRAFODION.SALES.

The default behavior takes effect only when `getConnection()` does not contain catalog and schema properties. Catalog and schema property values in `getConnection()` have higher precedence over the default behavior. To override the default schema name and associate a database connection with a different schema, specify the schema property during connection creation. For example, `getConnection()` in this SPJ method specifies the schema, SALES2, which overrides the default schema, SALES:

```
public static void numDailyOrders( Date date
                                 , int[] numOrders
                 ) throws SQLException
{
   Properties prop = new Properties() ;
   prop.setProperty( "schema", "SALES2" ) ;

   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection", prop) ;

   PreparedStatement getNumOrders =
      conn.prepareStatement( "SELECT COUNT( order_date ) "
                           + "FROM orders "
              + "WHERE order_date = ?"
              ) ;

   getNumOrders.setDate( 1, date ) ;

   ResultSet rs = getNumOrders.executeQuery() ;
   rs.next() ;

   numOrders[0] = rs.getInt( 1 ) ;

   rs.close() ;
   conn.close() ;

}
```

Be aware that overriding the default values by using getConnection() requires you to hard-code the catalog or schema name and might make SPJ methods less portable across systems.

4.2.4. Using the SESSION_USER or CURRENT_USER Function in an SPJ Method

SESSION_USER is an SQL function that returns the name of the authenticated database user who started the session and invoked the function, and CURRENT_USER (or USER) is an SQL function that returns the name of the database user who is authorized to invoke the function. If you plan to use the SESSION_USER or CURRENT_USER (or USER) function in an SPJ method, you should be aware of differences in their behavior depending on how external security is defined for the stored procedure.

Suppose that you write this Java method, which uses the CURRENT_USER function to return the name of the database user who is authorized to invoke the function:

```
public static void getUser( ResultSet [] rs ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   Statement stmt = conn.createStatement() ;

   rs[0] =
      stmt.executeQuery( "SELECT CURRENT_USER FROM (VALUES(1)) X(A) ; " ) ;
}
```

If this method is used in a stored procedure with external security defined as *invoker*, the CURRENT_USER function returns the name of the database user who is authorized to invoke the function, which happens to be the authenticated database user who started the session and called the stored procedure.

For example, suppose that DB USERADMINUSER creates a stored procedure named GETINVOKER using the `getUser()` method and sets the external security to invoker. If a database user named GTAPPER, who has the EXECUTE privilege on the stored procedure, calls GETINVOKER, the procedure returns his name:

```
Welcome to Apache Trafodion Command Interface
Copyright (c) 2015 Apache Software Foundation

User Name:GTAPPER Password:

Connected to Data Source: TDM_Default_DataSource

SQL> CALL trafodion.persnl.getinvoker() ;

(EXPR)
--------------------------------------------------------------------------------
GTAPPER

--- 1 row(s) selected.

--- SQL operation complete.
```

If the method is used in a stored procedure with external security defined as *definer*, the CURRENT_USER function returns the name of the database user who is authorized to invoke the function, which happens to be the user who created the stored procedure (that is, the definer of the stored procedure). When a stored procedure's external security is set to definer, any user who has the execute privilege on the stored procedure can call the procedure using the privileges of the user who created the stored procedure.

For example, suppose that DB USERADMINUSER creates a stored procedure named GETDEFINER using the `getUser()` method and sets the external security to definer. If the database user named GTAPPER, who has the EXECUTE privilege on the stored procedure, calls GETDEFINER, the procedure returns the name of the stored procedures's creator, DB USERADMINUSER, whose privileges GTAPPER is using to call the procedure:

```
SQL> SHOW USER

USER GTAPPER (NONE)

SQL> CALL trafodion.persnl.getdefiner() ;

(EXPR)
--------------------------------------------------------------------------------
DB USERADMINUSER

--- 1 row(s) selected.

--- SQL operation complete.
```

Suppose that you write this Java method, which uses the SESSION_USER function to return the name of the authenticated database user who started the session and invoked the function:

```
public static void getSessionUser( ResultSet [] rs ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   Statement stmt = conn.createStatement() ;

   rs[0] = stmt.executeQuery( "SELECT SESSION_USER FROM (VALUES(1) ) X(A) ; " ) ;
}
```

The SESSION_USER function returns the name of the authenticated database user who started the session and invoked the function, regardless of the external security setting of the stored procedure.

For example, suppose that DB USERADMINUSER creates a stored procedure named GETSESSIONUSER using the `getSessionUser()` method and sets the external security to definer. If the database user named GTAPPER, who has the EXECUTE privilege on the stored procedure, calls GETSESSIONUSER, the procedure returns his name because he is the authenticated user who started the session and invoked the function:

```
SQL> SHOW USER

USER GTAPPER (NONE)

SQL> CALL trafodion.persnl.getsessionuser() ;

(EXPR)
--------------------------------------------------------------------------------
GTAPPER

--- 1 row(s) selected.

--- SQL operation complete.
```

For more information about external security, see Understand External Security.

4.2.5. Exception Handling

For SPJ methods that access Trafodion, no special code is necessary for handling exceptions. If an SQL operation fails inside an SPJ, the error message associated with the failure is returned to the application that issues the CALL statement.

4.3. Handling Java Exceptions

If an SPJ method returns an uncaught Java exception or an uncaught chain of `java.sql.SQLException` objects, the database engine converts each Java exception object into an SQL error condition, and the CALL statement fails. Each SQL error condition contains the message text associated with one Java exception object.

If an SPJ method catches and handles exceptions itself, those exceptions do not affect SQL processing.

4.3.1. User-Defined Exceptions

The SQLSTATE values 38001 to 38999 are reserved for you to define your own error conditions that SPJ methods can return. By coding your SPJ method to throw a `java.sql.SQLException` object, you cause the CALL statement to fail with a specific user-defined SQLSTATE value and your own error message text.

If you define the SQLSTATE to be outside the range of 38001 to 38999, the database engine raises SQLSTATE 39001, external routine invocation exception.

This example uses the throw statement in the SPJ method named `numMonthlyOrders()` to raise a user-defined error condition when an invalid argument value is entered for the month:

```
public static void numMonthlyOrders( int month
                                   , int[] numOrders
                  ) throws java.sql.SQLException
{
   if ( month < 1 || month > 12 )
   {
      throw new
         java.sql.SQLException ( "Invalid value for month. "
                               + "Retry the CALL statement using a number "
                               + "from 1 to 12 to represent the month."
                  , "38001"
                  ) ;
   }

   ....
}
```

For more information about the numMonthlyOrders() method, see the MONTHLYORDERS Procedure.

For information about specific SQL errors, see the Trafodion Messages Manual, which lists the SQLCODE, SQLSTATE, message text, and cause-effect-recovery information for all SQL errors.

4.4. Compiling and Packaging Java Classes

On Trafodion, the class files of SPJ methods must be packaged in Java archive (JAR) files. After writing an SPJ method, compile the Java source file of the SPJ method into Java bytecode and package the Java bytecode in a JAR file. A Java method that you register as an SPJ might need to access, either directly or indirectly, other Java classes to operate properly. Those Java classes might include other application classes. To enable an SPJ method to refer to other application classes, put the application classes in the same JAR file as the SPJ class. All classes stored in the same JAR file as the SPJ class are accessible by default to the SPJ method.

After writing the SPJ method

1. Compile the Java source file into Java bytecode by using the Java programming language compiler, `javac`:

   ```
   javac Payroll.java
   ```

2. Put the SPJ class file and all associated class files into a Java archive (JAR) file:

   ```
   jar cvf Payroll.jar Payroll.class
   ```

   A manifest file is not needed for the JAR file.

Chapter 5. Deploy SPJ JAR Files

After developing and packaging the SPJ code into JAR files, you must move the JAR files from the client workstation to Trafodion.

You can upload a SPJ JAR file and create a library by using the CREATE LIBRARY command in the Trafodion Command Interface (trafci).

Libraries provide greater security for JAR files because libraries are database objects whose access is controlled using standard SQL security.

Deploying a JAR file to a Trafodion instance requires creating a library, and users must have the required privileges for creating libraries in order to deploy JAR files.

The examples in this chapter are based on the sample database documented in Appendix B: Sample Database.

5.1. Create a Library

Refer to the Trafodion SQL Reference Manual for full documentation of the CREATE LIBRARY statement including considerations and required privileges.

You must copy the jar file to the same directory on all the nodes in the cluster before running the CREATE LIBRARY statement. Otherwise, you will see an error message indicating that the jar file is not found.

Example

This example uses the sample Sales Class documented in Appendix A: Sample SPJs.

Your task is to add the Sales class jar file as a library object in the SALES schema. Do the following:

1. Compile the `Sales.java` source file.

   Example

   ```
   $ # Set up the environmental variables.
   $ cd $HOME/trafodion-incubator
   $ source env.sh
   $ # Prepare the class jar file
   $ cd $HOME/trafodion-spjs
   $ ls
   Inventory.java  Payroll.java  Sales.java
   $ # Compile source
   $ javac Sales.java
   $ ls Sales.*
   Sales.class  Sales.java
   $ # Package jar file
   $ jar cvf Sales.jar Sales.class
   $ ls Sales.*
   Sales.class  Sales.jar  Sales.java
   $
   ```

2. Multi-node cluster only: Create the target directory, if necessary.

   Example

   ```
   $ pdsh $MY_NODES -x $HOSTNAME mkdir $HOME/trafodion-spjs
   ```

3. Multi-node cluster only: Copy the jar file to all nodes in the cluster.

   Example

```
$ pdcp $MY_NODES Sales.jar $PWD/.
```

4. Create the library object using trafci.

```
$ pwd
/home/trafodion/trafodion-jars
$ trafci

Welcome to Apache Trafodion Command Interface
Copyright (c) 2015 Apache Software Foundation

Host Name/IP Address: localhost:23400
User Name: zz

Connected to Trafodion

SQL> CREATE LIBRARY trafodion.sales.sales FILE '/home/trafodion/trafodion-
spjs/Sales.jar' ;

--- SQL operation complete.

SQL> get libraries in schema trafodion.sales ;

SALES

--- SQL operation complete.

SQL>
```

5.2. Drop a Library

Dropping a library removes the library from the schema in the database and removes the library's underlying JAR file from Trafodion.

Refer to the Trafodion SQL Reference Manual for full documentation of the DROP LIBRARY statement including considerations and required privileges.

Example

Your task is to remove Sales class jar file from the SALES schema.

Do the following using trafci:

```
$ trafci
Welcome to Apache Trafodion Command Interface
Copyright (c) 2015 Apache Software Foundation

Host Name/IP Address: localhost:23400
User Name: zz

Connected to Trafodion

SQL> get libraries in schema trafodion.sales ;

SALES

--- SQL operation complete.

SQL> DROP LIBRARY trafodion.sales.saleslib ;

--- SQL operation complete.

SQL>
```

5.3. Display Libraries

Refer to the Trafodion SQL Reference Manual for full documentation of the GET statement.

Use the `GET libraries [in schema [catalog-name.] schema-name] statement in trafci to display the libraries in a schema.

Example

```
$ trafci
Welcome to Apache Trafodion Command Interface
Copyright (c) 2015 Apache Software Foundation

Host Name/IP Address: localhost:23400
User Name: zz

Connected to Trafodion

SQL> get libraries in schema trafodion.sales ;

SALES

--- SQL operation complete.
SQL>
```

Chapter 6. Create SPJs

You can create the procedures in Trafodion after you've deployed the libraries for the SPJ JAR files. See Deploy SPJ JAR Files.

The examples in this chapter are based on the sample database documented in Appendix B: Sample Database. Also, the `Sales.java` class are assumed to have been added as a library object using the `CREATE LIBRARY` statement. See Deploy SPJ JAR Files.

6.1. Create a Procedure

The Create Procedure tool registers an existing Java method as a stored procedure in Java (SPJ) within SQL.

Refer to the Trafodion SQL Reference Manual for full documentation of the CREATE PROCEDURE statement including considerations and required privileges.

6.1.1. Create Procedure Settings

Reference: Syntax Description of CREATE PROCEDURE.

You need to determine the following settings for the CREATE PROCEDURE statement:

Attribute	Syntax	Guidance
procedure-ref	`[[catalog-name.]schema-name.]procedure-name`	- **catalog**: must be `trafodion`. - **schema-name**: the schema the procedure is associated with. - **procedure-name**: a name that is unique and does not exist for any procedure or function in the same schema. It's a good habit to use the lowercase version of the procedure name.
sql-parameter	`[parameter-mode] [sql-identifier] sql-datatype`	- **parameter-mode**: IN (pass data to SPJ), OUT (accept data from SPJ), or INOUT (passed data to and accepts data from SPJ). - **sql-identifier**: The name of the parameter passed to the SPJ. - **sql-datatype**: SQL data type mapped to Java data type. See Java Parameters for mapping, Data Types for SQL data types. Repeat for each parameter in the SPJ.

Attribute	Syntax	Guidance
external name	`external name 'java-method-name [( java-signature )]'`	- **java-method-name**: case-sensitive name of the SPJ method of the form `[package-name.]class-name.method-name`. The Java method must exist in a Java class file, *`class-name`*`.class`, within a library registered in the database. If the class file that contains the SPJ method is part of a package, then you must also specify the package name. - **java-signature**: specifies the signature of the SPJ method and consists of `([java-datatype[, java-datatype]…])` Necessary only if you want to specify a Java wrapper class (for example, `java.lang.integer`) instead of a java primitive data type (for example, `int`). Use the `javap` utility to determine the correct values for java-signature.
library	`library [[catalog-name.]schema-name.]library-name`	- **catalog**: must be `trafodion`. - **schema-name**: the schema the library is associated with. - **library-name**: name of library containing the SPJ method. + See Create a Library for more information about creating libraries.
external security	`external security [ invoker` (default) | `definer ]`	- **invoker**: determines that users can execute, or invoke, the stored procedure using the privileges of the user who invokes the stored procedure. - **definer**: determines that users can execute, or invoke, the stored procedure using the privileges of the user who created the stored procedure. See Understand External Security for more information.
language java	`language java`	Must be set specified the procedure is an SPJ rather than a UDF. Compare: CREATE FUNCTION.
parameter style java	`parameter style java`	Must be set specified the procedure is an SPJ rather than a UDF. Compare: CREATE FUNCTION.
sql access	`no sql` | `contains sql` (default) | `modifies sql data` | `reads sql data`	- **no sql**: the SPJ cannot perform SQL operations. - **contains sql**: the SPJ can read and modify SQL data. - **modifies sql data**: the SPJ can read and modify SQL data. - **reads sql data**: the SPJ can read and modify SQL data. It's a good practice to specify `modifies sql data` and `reads sql data` over `contains sql` as it helps users of the procedure to better understand its impact.
dynamic result sets	`dynamic result sets 0` (default) to *n*	The maximum number of result sets that the SPJ can return. This option is applicable only if the method signature contains a `java.sql.resultset[]` object.

Attribute	Syntax	Guidance
transaction	`transaction required` (default) | `no transaction required`	Determines whether the SPJ must run in a transaction inherited from the calling application (`transaction required` or whether the SPJ runs without inheriting the calling application's transaction (`no transaction required`). Typically, you want the stored procedure to inherit the transaction from the calling application.
determinism	`deterministic` | `not deterministic` (default)	Whether the SPJ always returns the same values for out and inout parameters for a given set of argument values (`deterministic`) or does not return the same values (`not deterministic`). `deterministic` means that Trafodion can cache the result to improve performance.
isolation	`isolate` (default) | `no isolate`	The SPJ executes either in the environment of the database server (`no isolate`) or in an isolated environment (`isolate`, the default option). Trafodion allows both options but always executes the SPJ in the UDR server process (`isolate`).

Consider the following method in the `Sales.java` class:

```
// The TOTALPRICE procedure accepts the quantity, shipping speed, and price
// of an item, calculates the total price, including tax and shipping
// charges, and returns the total price to an input/output parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#totalprice-
procedure
// for more documentation.
public static void totalPrice( BigDecimal qtyOrdered
                , String shippingSpeed
                , BigDecimal[] price
                ) throws SQLException
{
   BigDecimal shipcharge = new BigDecimal( 0 ) ;

   if ( shippingSpeed.equals( "economy" ) )
   {
      shipcharge = new BigDecimal( 1.95 ) ;
   }
   else if ( shippingSpeed.equals( "standard" ) )
   {
      shipcharge = new BigDecimal( 4.99 ) ;
   }
   else if ( shippingSpeed.equals( "nextday" ) )
   {
      shipcharge = new BigDecimal( 14.99 ) ;
   }
   else
   {
      throw new SQLException( "Invalid value for shipping speed. "
                + "Retry the CALL statement using "
                + "'economy' for 7 to 9 days, "
                + "'standard' for 3 to 5 days, or "
                + "'nextday' for one day."
                , "38002"
                ) ;
   }

   BigDecimal subtotal   = price[0].multiply( qtyOrdered ) ;
   BigDecimal tax        = new BigDecimal( 0.0825 ) ;
   BigDecimal taxcharge  = subtotal.multiply( tax ) ;
   BigDecimal charges    = taxcharge.add( shipcharge ) ;
   BigDecimal totalprice = subtotal.add( charges ) ;

   totalprice = totalprice.setScale( 2, BigDecimal.ROUND_HALF_EVEN ) ;
   price[0] = totalprice ;

}
```

The CREATE PROCEDURE attributes for `totalPrice` are defined as follows:

Attribute	Definition	Discussion
procedure-ref	`trafodion.sales.totalprice`	Used lowercase version of totalPrice for *procedure-name*.
sql-parameter-1	`IN qty NUMERIC(18)`	The first argument to the SPJ is defined as: `BigDecimal qtyOrdered`, which maps to `NUMERIC`. The size is chosen to allow for large numbers.
sql-parameter-2	`IN speed VARCHAR(10)`	The second argument to the SPJ is defined as: `String shippingSpeed`. Possible values are: `economy`, `standard`, or `nextday`. `VARCHAR(10)` is a good choice.
sql-parameter-3	`INOUT price NUMERIC(18,2)`	The third argument to the SPJ is defines as: `BigDecimal[] price`. The SPJ uses an input price to calculate to total price including taxes and shipping cost. Therefore, `INOUT` is the appropriate choice. Given that the `qty` argument is NUMERIC(18), then this argument should be too. The precision choice fits the the type of calculation being peformed.
external name	`EXTERNAL NAME 'Sales.totalPrice'`	The class name is `Sales`. The procedure name is `totalPrice`.
library	`library trafodion.sales.sales`	The library name was defined using the CREATE LIBRARY statement. See the example in Create a Library.
external security	`invoker`[1]	Use the privileges of the SPJ invoker.
language java	`LANGUAGE JAVA`	Required for SPJs.
parameter style java	`PARAMETER STYLE JAVA`	Required for SPJs.
sql access	`NO SQL`	This SPJ performs no SQL operations, just calculations.
dynamic result sets	`DYNAMIC RESULT SETS 0`	The method does not contain a `java.sql.resultset[]` object.
transaction	`TRANSACTION REQUIRED`[1]	The method should inherit the calling application's transaction, if any.
determinism	`NOT DETERMINISTIC`[1]	The results must be recalculated each time; that is, cannot be cached.
isolation	`ISOLATE`[1]	Execute the SPJ in an isolated environment; that is, the UDR server process.

[1] Definition represents default value.

Using the information above, you create the new procedure as follows:

```
CREATE PROCEDURE trafodion.sales.totalprice( IN qty NUMERIC (18),
                                             IN speed VARCHAR (10),
                                             INOUT price NUMERIC (18,2)
                        )
   EXTERNAL NAME 'Sales.totalPrice'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   NO SQL
   DYNAMIC RESULT SETS 0
   TRANSACTION REQUIRED
   ISOLATE
   ;
```

Removing the default values to simplify the statement, you get:

```
CREATE PROCEDURE trafodion.sales.totalprice( IN qty NUMERIC (18),
                                             IN speed VARCHAR (10),
                                             INOUT price NUMERIC (18,2)
                        )
   EXTERNAL NAME 'Sales.totalPrice'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   NO SQL
   ;
```

To understand how to define `java-signature`, consider the following method in the `Inventory.java` class.

```
public static void supplierInfo( BigDecimal suppNum
                                , String[] suppName
                                , String[] streetAddr
                                , String[] cityName
                                , String[] stateName
                                , String[] postCode
                                ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getSupplier =
      conn.prepareStatement( "SELECT suppname, street, city, "
                           + "       state, postcode "
                           + "FROM trafodion.invent.supplier "
                           + "WHERE suppnum = ?"
                           ) ;

   getSupplier.setBigDecimal( 1, suppNum ) ;
   ResultSet rs = getSupplier.executeQuery() ;
   rs.next() ;

   suppName[0]   = rs.getString( 1 ) ;
   streetAddr[0] = rs.getString( 2 ) ;
   cityName[0]   = rs.getString( 3 ) ;
   stateName[0]  = rs.getString( 4 ) ;
   postCode[0]   = rs.getString( 5 ) ;

   rs.close() ;
   conn.close() ;
}
```

The CREATE PROCEDURE attributes for `supplierInfo` are defined as follows:

Attribute	Definition
procedure-ref	`trafodion.invent.supplierinfo`
sql-parameter-1	`IN suppnum NUMERIC(4)`
sql-parameter-2	`OUT suppname CHARACTER(18)`
sql-parameter-3	`OUT address CHARACTER(22)`
sql-parameter-4	`OUT city CHARACTER(14)`
sql-parameter-5	`OUT state CHARACTER(12)`
sql-parameter-6	`OUT zipcode CHARACTER(10)`
external name	`EXTERNAL NAME 'Inventory.supplierInfo ( java.math.BigDecimal, java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[] )'`
library	`LIBRARY trafodion.invent.inventory`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

You can use the `javap` utility to inspect the attributes of the method.

Example

```
$ javap Inventory.class
Compiled from "Inventory.java"
public class Inventory {
  public Inventory();
  public static void supplierInfo(java.math.BigDecimal, java.lang.String[],
java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[]) throws
java.sql.SQLException;
  public static void supplyQuantities(int[], int[], int[]) throws
java.sql.SQLException;
  public static void partLocations(int, int, java.sql.ResultSet[],
java.sql.ResultSet[]) throws java.sql.SQLException;
}
```

The entry for `supplierInfo` method shows the attributes that should be used for `java-signature`.

You don’t specify for `java.sql.ResultSet[]` attributes; these are handled via the `DYNAMIC RESULTS SET` attribute in the `CREATE PROCEDURE` statement.

6.2. Understand External Security

The external security of an SPJ determines the privileges, or rights, that users have when executing (or calling) the SPJ. An SPJ can be created with one of these types of external security: invoker or definer.

If an SPJ is created with the invoker type of external security, then the SPJ is executed with *invoker rights*.

Invoker rights allow a user who has the execute privilege on the SPJ to call the SPJ using his or her existing privileges.

In this case, the user must be granted privileges to access the underlying database objects on which the SPJ operates. If a user tries to call an SPJ that has invoker external security and that operates on database objects to which the user does not have privileges, then the CALL statement fails with an error message indicating that the user does not have the appropriate permissions.

Granting a user privileges to the underlying database objects gives the user direct access to those database objects, which could pose a risk to more sensitive or critical data to which users should not have access. For example, an SPJ might operate on a subset of the data in an underlying database object, but that database object might contain other more sensitive or critical data to which users should not have access.

If an SPJ is created with the definer type of external security, then the SPJ is executed with *definer rights*.

Definer rights allow a user who has the execute privilege on an SPJ to call the SPJ using the privileges of the user who created the SPJ.

In this case, the user does not require privileges to access the underlying database objects on which the SPJ operates. Instead, the user is allowed to access or manipulate data in the underlying database objects by invoking the SPJ. That way, users are restricted from directly accessing or manipulating more sensitive or critical data in the database.

However, be careful about the users to whom you grant execute privilege on an SPJ with definer external security because those users will be able to execute the SPJ without requiring privileges to the underlying database objects.

To set the external security of an SPJ, see Create a Procedure.

6.3. Drop a Procedure

To drop a procedure, you must own the procedure or have the DROP_PROCEDURE privilege for the schema. The schema owner can grant such a schema-level privilege to other users or roles.

For example, if the schema owner granted you privileges to drop all objects in the schema, you could drop procedures. For more information, see the GRANT SCHEMA statement in the Trafodion SQL Reference Manual

Refer to the Trafodion SQL Reference Manual for full documentation of the DROP PROCEDURE statement including considerations and required privileges.

Example

Your task is to remove Sales class jar file from the SALES schema.

Do the following using trafci:

```
$ trafci
Welcome to Apache Trafodion Command Interface
Copyright (c) 2015 Apache Software Foundation

Host Name/IP Address: localhost:23400
User Name: zz

Connected to Trafodion

SQL> get procedures for library trafodion.sales.saleslib ;

TOTALPRICE

--- SQL operation complete.

SQL> DROP PROCEDURE trafodion.sales.totalprice ;

--- SQL operation complete.

SQL>
```

6.4. Display Procedures and Their Properties

In the trafci command-line interface, use the SHOW PROCEDURES command to display the procedures in a schema. For example, this SHOW PROCEDURES command displays a list of the procedures in the SALES schema:

```
SQL> SET SCHEMA trafodion.sales ;

--- SQL operation complete.

SQL> SHOW PROCEDURES

PROCEDURE NAMES
------------------------------------------------------------------------------------
DAILYORDERS LOWERPRICE MONTHLYORDERS ORDERSUMMARY PARTDATA TOTAL PRICE

SQL>
```

You can also use a wild-card pattern to search for a particular procedure. For example, this SHOW PROCEDURES command displays all the procedures in the SALES schema that have price in their names:

```
SQL> SHOW PROCEDURES %price

PROCEDURE NAMES
------------------------------------------------------------------------------------
LOWERPRICE TOTALPRICE

SQL>
```

For more information about trafci, see the Trafodion Command Interface Guide.

Chapter 7. Grant Privileges

Security for SPJs is implemented by schema ownership rules and by granting privileges to specified database users and roles.

The schema in which an SPJ is registered is the unit of ownership. The database user who creates the schema is the owner of that schema and all objects associated with it. In Trafodion, the schema owner automatically has these privileges:

- Ability to create and drop SPJs in the schema. You can create SPJs in the schema provided that you also have the CREATE_PROCEDURE privilege for the SQL_OPERATIONS component.
- EXECUTE and WITH GRANT OPTION privileges on the SPJs in the schema

To create or drop an SPJ in a schema, you must be the schema owner or have the appropriate create or drop privileges for the schema. For more information, see Required Privileges for Creating or Dropping an SPJ.

You must have the EXECUTE privilege on the SPJ to it. The EXECUTE privilege allows a user to invoke an SPJ by issuing a CALL statement. The WITH GRANT OPTION privilege allows a user to grant the EXECUTE and WITH GRANT OPTION privileges to other users and roles. For more information, see:

- Granting Execute Privileges on an SPJ
- Granting Privileges on Referenced Database Objects
- Revoking Execute Privileges on an SPJ
- Using Script Files to Grant and Revoke Privileges

To display the current ownership and privileges, see Display Procedures and Their Properties.

7.1. Granting Execute Privileges on an SPJ

Use the GRANT PROCEDURE or GRANT statement to assign the EXECUTE and WITH GRANT OPTION privileges on an SPJ to specific database users and roles. In a GRANT statement, specify ALL PRIVILEGES to grant the EXECUTE privilege on an SPJ. For the syntax of the GRANT PROCEDURE and GRANT statements, see the Trafodion SQL Reference Manual.

If you own the SPJ, then you can grant the EXECUTE and WITH GRANT OPTION privileges on the SPJ to any database user or role. If you are not the owner of the SPJ, then you must have been granted the EXECUTE and WITH GRANT OPTION privileges on the SPJ to grant privileges to other database users and roles, or you must be associated with a role that has the EXECUTE and WITH GRANT OPTION privileges on the SPJ.

As the owner of an SPJ, you can selectively grant the EXECUTE and WITH GRANT OPTION privileges to specified database users and roles. For some SPJs, particularly ones that handle sensitive information or modify data, you should grant the EXECUTE and WITH GRANT OPTION privileges to a restricted group of users or roles.

For example, the SPJ named ADJUSTSALARY changes an employee's salary in the database. Therefore, only specific users or roles should be allowed to invoke this SPJ. In this example, the SPJ owner (or creator) grants the EXECUTE and WITH GRANT OPTION privileges on ADJUSTSALARY to the Payroll directors.

```
GRANT EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary
TO "payrolldir1", "payrolldir2" WITH GRANT OPTION ;
```

One of the Payroll directors grants the EXECUTE privilege on ADJUSTSALARY to the regional department managers:

```
GRANT EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary TO "rgn1mgr", "rgn2mgr", "rgn3mgr"
WITH GRANT OPTION ;
```

In some cases, all users of a database system might need to invoke an SPJ.

For example, the SPJ named TOTALPRICE calculates the total price of an item, including tax and shipping charges. This SPJ does not handle sensitive information or modify data and might be useful to customers or anyone within the company.

Therefore, the SPJ owner (or creator) grants the EXECUTE privilege on TOTALPRICE to PUBLIC, meaning all present and future database users and roles:

```
GRANT EXECUTE
ON PROCEDURE trafodion.sales.totalprice TO PUBLIC ;
```

After granting the EXECUTE privilege to PUBLIC, you cannot revoke the privilege from a subset of database users or roles. You must revoke the privilege from PUBLIC and then grant the privilege to specific database users and roles.

7.2. Granting Privileges on Referenced Database Objects

If the SPJ operates on a database object and the SPJ has been created with the external security specified as invoker (EXTERNAL SECURITY INVOKER), then the database users that invoke the SPJ must have the appropriate privileges on that database object.

If the SPJ has been created with the external security specified as definer (EXTERNAL SECURITY DEFINER), then users other than the procedure's creator may invoke the SPJ without needing privileges on the underlying database objects.

When the SPJ's external security is definer, users execute, or invoke, the stored procedure using the privileges of the user who created the stored procedure. The user who creates the stored procedure must have the appropriate privileges on the underlying database objects. For more information, see Understand External Security.

For example, users with the EXECUTE privilege on the SPJ named ADJUSTSALARY, which is defined with EXTERNAL SECURITY INVOKER and which selects data from and updates the EMPLOYEE table, must have the SELECT and UPDATE privileges on that SQL table. The SPJ owner (or creator) grants these access privileges to the Payroll directors:

```
GRANT SELECT, UPDATE (salary)
ON TABLE trafodion.persnl.employee
TO "payrolldir1", "payrolldir2" WITH GRANT OPTION ;
```

One of the Payroll directors then grants these access privileges to the regional department managers:

```
GRANT SELECT, UPDATE (salary)
ON TABLE trafodion.persnl.employee
TO "rgn1mgr", "rgn2mgr", "rgn3mgr" ;
```

Users with the EXECUTE privilege on the SPJ named TOTALPRICE, which does not access the database, are not required to have privileges on any database tables because that SPJ does not access any database tables.

The types of SQL statements in the underlying SPJ method, such as SELECT, UPDATE, DELETE, and INSERT, indicate which types of privileges are required for the referenced database objects.

For the syntax of the GRANT statement, see the Trafodion SQL Reference Manual.

7.3. Revoking Execute Privileges on an SPJ

Use the REVOKE PROCEDURE or REVOKE statement to remove the EXECUTE or WITH GRANT OPTION privilege on an SPJ from specific database users or roles. In a REVOKE statement, specify ALL PRIVILEGES to revoke the EXECUTE privilege on an SPJ. For the syntax of the REVOKE PROCEDURE and REVOKE statements, see the Trafodion SQL Reference Manual.

If you own the SPJ, then you can revoke the EXECUTE and WITH GRANT OPTION privileges on the SPJ from any database user or role to whom you granted those privileges or, if you did not directly grant those privileges, on behalf of the role that granted those privileges, provided that you were granted that role.

If you are not the owner of the SPJ, then you must have been granted the EXECUTE and WITH GRANT OPTION privileges on the SPJ to revoke privileges from other database users or roles, and you can revoke the privileges only from other users or roles to whom you have granted those privileges or, if you did not directly grant those privileges, on behalf of the role that granted those privileges, provided that you were granted that role.

For example, the `payrolldir1` user can revoke the EXECUTE privilege on ADJUSTSALARY from one or more regional department managers to whom the `payrolldir1` user granted those privileges.

In this example, the `payrolldir1` user revokes the EXECUTE privilege from the Region 2 department manager:

```
REVOKE EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary FROM "rgn2mgr" ;
```

The `payrolldir1` user cannot revoke the EXECUTE or WITH GRANT OPTION privilege from the `payrolldir2` user because it was the SPJ owner (or creator) who granted those privileges.

A user can revoke the WITH GRANT OPTION privilege on ADJUSTSALARY from any user or role to whom the user granted this privilege.

In this example, the SPJ owner (or creator) revokes the WITH GRANT OPTION privilege from the `payrolldir1` user:

```
REVOKE GRANT OPTION FOR EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary FROM "payrolldir1" ;
```

A user can also revoke the EXECUTE privilege from any user or role to whom the user granted this privilege and from any dependent privileges by using the CASCADE option.

In this example, the SPJ owner (or creator) revokes the EXECUTE privilege from the `payrolldir1` user and from the regional department managers to whom the `payrolldir1` user granted privileges:

```
REVOKE GRANT OPTION FOR EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary FROM "payrolldir1"
CASCADE ;
```

For SPJs on which all users (that is, PUBLIC) have privileges, you can revoke privileges from PUBLIC but not from one or more specific users or roles.

For example, this statement revokes the EXECUTE privilege on the SPJ named TOTALPRICE from all users and roles (that is, PUBLIC):

```
REVOKE EXECUTE
ON PROCEDURE trafodion.sales.totalprice FROM PUBLIC;
```

7.4. Using Script Files to Grant and Revoke Privileges

Consider keeping your GRANT or REVOKE statements in script files. That way, you can quickly and easily grant or revoke privileges to the SPJs, as needed.

7.4.1. Script File for Granting Privileges

You can use another or the same script file to grant privileges on a series of SPJs.

For example, the script file, `grantprocs.sql`, contains a series of GRANT PROCEDURE and GRANT statements:

```
?SECTION GrantSalesProcs

GRANT EXECUTE
ON trafodion.sales.monthlyorders
TO PUBLIC ;

GRANT SELECT
ON TABLE trafodion.sales.orders TO PUBLIC ;

?SECTION GrantPersnlProcs

GRANT EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary TO "payrolldir1", "payrolldir2"
WITH GRANT OPTION ;

GRANT SELECT, UPDATE(salary)
ON TABLE trafodion.persnl.employee
TO "payrolldir1", "payrolldir2" WITH GRANT OPTION ;
```

To grant privileges on the SPJs, run the script file in the trafci interface:

```
OBEY c:\grantprocs.sql (GrantSalesProcs)
```

7.4.2. Script File for Revoking Privileges

You can use another or the same script file to revoke privileges on a series of SPJs.

For example, the script file, `revokeprocs.sql`, contains a series of REVOKE PROCEDURE and REVOKE statements:

```
?SECTION RevokeSalesProcs

REVOKE EXECUTE
ON PROCEDURE trafodion.sales.monthlyorders FROM PUBLIC ;

REVOKE SELECT
ON TABLE trafodion.sales.orders FROM PUBLIC ;

?SECTION RevokePersnlProcs

REVOKE EXECUTE
ON PROCEDURE trafodion.persnl.adjustsalary FROM "payrolldir1", "payrolldir2"
CASCADE ;

REVOKE SELECT, UPDATE(salary)
ON TABLE trafodion.persnl.employee
FROM "payrolldir1", "payrolldir2" CASCADE ;
```

To revoke privileges on the SPJs, run the script file in the trafci interface:

```
OBEY c:\revokeprocs.sql (RevokeSalesProcs)
```

Chapter 8. Execute SPJs

This chapter describes how to execute SPJs by using the CALL statement and assumes that you have already registered the SPJs in Trafodion and that you have granted privileges to execute the SPJs to the appropriate database users. For information, see Deploy SPJ JAR Files and Create SPJs.

This chapter covers these topics:

- Required Privileges for Calling an SPJ
- Transaction Behavior
- Multithreading
- Using the CALL Statement
- Calling SPJs in trafci
- Calling SPJs From an ODBC Client Application
- Calling SPJs From a JDBC Client Application

The CALL statement invokes an SPJ in the database. You can issue a CALL statement from any of these applications or interfaces supported by Trafodion:

- trafci command-line interface or script file
- JDBC Type-4 client applications
- ODBC client applications

You can use a CALL statement as a stand-alone SQL statement in applications or in command-line interfaces, such as trafci. You can also use a CALL statement in a trigger but not inside a compound statement or with rowsets. The SPJ that you use in a trigger must not have any OUT or INOUT parameters or return any result sets.

8.1. Required Privileges for Calling an SPJ

To execute the CALL statement, you must have the EXECUTE privilege on the procedure. For more information, see Grant Privileges.

8.2. Transaction Behavior

The stored procedure's transaction attribute determines whether it inherits the transaction from the calling application (TRANSACTION REQUIRED) or whether it runs without inheriting the calling application's transaction (NO TRANSACTION REQUIRED). The transaction attribute is set during the creation of the stored procedure. For more information, see the Transaction Required attribute in Create a Procedure.

Typically, you want the stored procedure to inherit the transaction from the calling application. See Transaction Required. However, if the SPJ method does not access the database or if you want the stored procedure to manage its own transactions, you should set the stored procedure's transaction attribute to NO TRANSACTION REQUIRED. See No Transaction Required.

8.2.1. Transaction Required

If you want the SPJ method to inherit the transaction from the calling application, set the stored procedure's transaction attribute to TRANSACTION REQUIRED (the default setting) when creating the stored procedure. For more information, see the Transaction Required attribute in Create a Procedure. When a stored procedure's transaction attribute is TRANSACTION REQUIRED, a CALL statement automatically initiates a transaction if there is no active transaction.

Using Transaction Control Statements or Methods

If you select `Yes` for the Transaction Required attribute when creating a stored procedure, then you should not use transaction control statements (or equivalent JDBC transaction methods) in the SPJ method. Transaction control statements include COMMIT WORK and ROLLBACK WORK, and the equivalent JDBC transaction methods are `Connection.commit()` and `Connection.rollback()`.

If you try to use transaction control statements or methods in an SPJ method when the stored procedure's transaction attribute is set to TRANSACTION REQUIRED, then the transaction control statements or methods in the SPJ method are ignored and the Java virtual machine (JVM) does not report any errors or warnings.

When the stored procedure's transaction attribute is set to TRANSACTION REQUIRED, then you should rely on the transaction control statements or methods in the application that calls the stored procedure and allow the calling application to manage the transactions.

Committing or Rolling Back a Transaction

If you do not use transaction control statements in the calling application, then the transaction initiated by the CALL statement might not automatically commit or roll back changes to the database.

When AUTOCOMMIT is ON (the default setting), then the database engine automatically commits or rolls back any changes made to the database at the end of the CALL statement execution. However, when AUTOCOMMIT is OFF, the current transaction remains active until the end of the client session or until you explicitly commit or roll back the transaction.

To ensure an atomic unit of work when calling an SPJ, use the COMMIT WORK statement in the calling application to commit the transaction when the CALL statement succeeds, and use the ROLLBACK WORK statement to roll back the transaction when the CALL statement fails. For more information about transaction management, see the Trafodion SQL Reference Manual.

8.2.2. No Transaction Required

In some cases, you might not want the SPJ method to inherit the transaction from the calling application. Instead, you might want the stored procedure to manage its own transactions or to run without a transaction. Not inheriting the calling application's transaction is useful in these cases:

- The stored procedure performs several long-running operations, such as multiple DDL or table maintenance operations, on the database. In this case, you might want to commit those operations periodically from within the SPJ method to avoid locking tables for a long time.
- The stored procedure performs certain SQL operations that must run without an active transaction. For example, INSERT, UPDATE, and DELETE statements with the WITH NO ROLLBACK option are rejected when a transaction is already active, as is the case when a stored procedure inherits a transaction from the calling application. The PURGEDATA utility is also rejected when a transaction is already active.
- The stored procedure does not access the database. (For an example, see the TOTALPRICE Procedure.) In this case, the stored procedure does not need to inherit the transaction from the calling application. By setting the stored procedure's transaction attribute to NO TRANSACTION REQUIRED, you can avoid the overhead of the calling application's transaction being propagated to the stored procedure.

In these cases, you should set the stored procedure's transaction attribute to NO TRANSACTION REQUIRED when creating the stored procedure. For more information, see the Transaction Required attribute in Create a Procedure.

If you select `No` for the Transaction Required attribute when creating a stored procedure and if the SPJ method creates a JDBC default connection, then that connection has autocommit enabled by default. You can either use the autocommit transactions or disable autocommit (`conn.setAutoCommit(false);`) and use the JDBC transaction methods, `Connection.commit()` and `Connection.rollback()`, to commit or roll back work where needed.

8.3. Multithreading

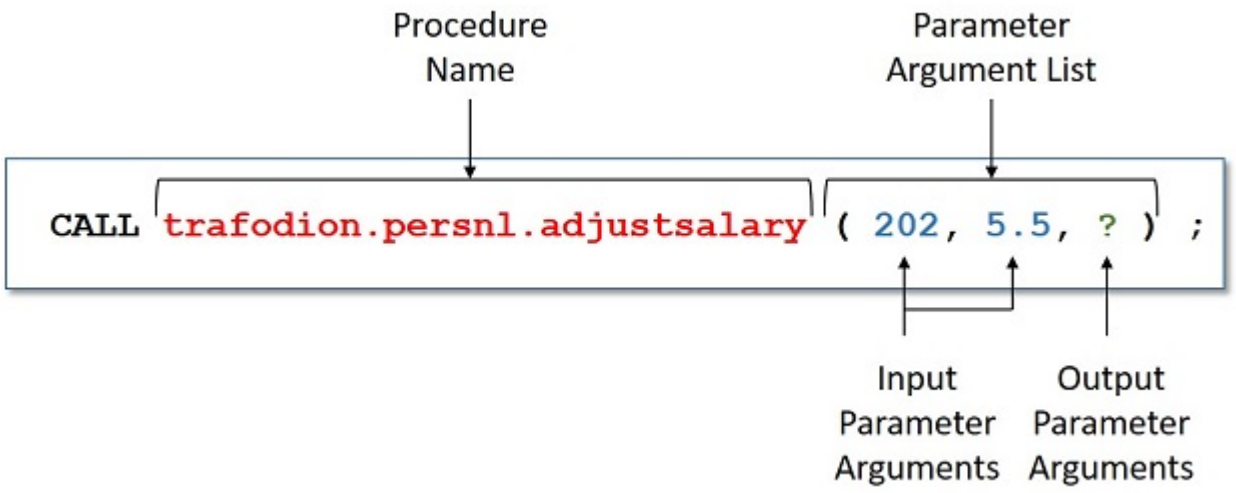
Trafodion manages a single thread of execution within an SPJ environment, even if the application that issues a CALL statement is a multi-threaded Java application.

The CALL statements in a multi-threaded application can execute in a nonblocking manner, but the SPJ methods underlying those CALL statements execute serially within a given SPJ environment.

8.4. Using the CALL Statement

To invoke a stored procedure, specify the name of the stored procedure and its arguments in a CALL statement, as shown in the figure below.

CALL Statement Elements



For the syntax of the CALL statement, see the Trafodion SQL Reference Manual.

8.4.1. Specifying the Name of the SPJ

In the CALL statement, specify the name of an SPJ that you have already created in the database. Qualify the procedure name with the same catalog and schema that you specified when you registered the SPJ. For example:

```
CALL trafodion.persnl.adjustsalary( 202, 5.5, ? ) ;
```

Or, for example:

```
SET SCHEMA trafodion.persnl ;
CALL adjustsalary( 202, 5.5, ? ) ;
```

If you do not fully qualify the procedure name, then the database engine qualifies the procedure according to the catalog and schema of the current session.

8.4.2. Listing the Parameter Arguments of the SPJ

Each argument that you list in the CALL statement must correspond to an SQL parameter of the SPJ. A result set in the Java signature of the SPJ method does not correspond to an SQL parameter. Do not specify result sets in the argument list.

For example, if you registered the stored procedure with three SQL parameters (two IN parameters and one OUT parameter), then you must list three formal parameter arguments, separated by commas, in the CALL statement:

```
CALL trafodion.persnl.adjustsalary( 202, 5, ? ) ;
```

If the SPJ does not accept arguments, you must specify empty parentheses, as shown below:

```
CALL trafodion.sales.lowerprice() ;
```

If the SPJ has one IN parameter, one OUT parameter, and two result sets, you must list the IN and OUT parameters but not the result sets in the argument list:

```
CALL trafodion.sales.ordersummary('01-01-2011', ?);
```

Data Conversion of Parameter Arguments

The database engine performs an implicit data conversion when the data type of a parameter argument is compatible with but does not match the formal data type of the stored procedure.

For stored procedure input values, the conversion is from the actual argument value to the formal parameter type.

For stored procedure output values, the conversion is from the actual output value, which has the data type of the formal parameter, to the declared type of the dynamic parameter.

Input Parameter Arguments

To pass data to an IN or INOUT parameter of an SPJ, specify an SQL value expression that evaluates to a character, date-time, or numeric value. The SQL value expression can evaluate to NULL provided that the underlying Java parameter supports null values. For more information, see Null Input and Output.

For an IN parameter argument, use one of these SQL value expressions in that table below:

Table 2. Input Parameter Argument Types

Type of Argument	Examples

Literal	`CALL adjustsalary( 202, 5.5, ? ) ;`
	`CALL dailyorders( DATE '2011-03-19', ? );`
	`CALL totalprice(23, 'nextday' , ?param ) ;`

SQL function (including CASE and CAST expressions)	`CALL dailyorders( CURRENT_DATE, ? ) ;`
Arithmetic expression	`CALL adjustsalary( 202, ?percent \ 0.25*, :OUT newsalary ) ;`
Concatenation operation	`CALL totalprice( 23, 'next' || 'day', ?param ) ;`
Scalar subquery	`CALL totalprice ( ( SELECT qty_ordered FROM odetail WHERE ordernum = 100210 AND partnum = 5100 ) , 'nextday', ?param ) ;`
Dynamic parameter	`CALL adjustsalary( ?, ?, ?) ;` `CALL adjustsalary( ?param1, ?param2, ?param3 ) ;`

For more information about SQL value expressions, see the Trafodion SQL Reference Manual.

Because an INOUT parameter passes a single value to and accepts a single value from an SPJ, you can specify only dynamic parameters for INOUT parameter arguments in a CALL statement.

Output Parameter Arguments

Except for result sets, an SPJ returns values in OUT and INOUT parameters. Each OUT or INOUT parameter accepts only one value from an SPJ. Any attempt to return more than one value to an output parameter results in a Java exception. See Returning Output Values From the Java Method.

OUT and INOUT parameter arguments must be dynamic parameters in a client application (for example, ?) or named or unnamed parameters in trafci (for example, `?param` or ?).

For information about how to call SPJs in different applications, see:

- Calling SPJs in trafci
- Calling SPJs From an ODBC Client Application
- Calling SPJs From a JDBC Client Application

Result Sets

Result sets are an ordered set of open cursors that the SPJ method returns to the calling application in `java.sql.ResultSet[]` parameter arrays. The `java.sql.ResultSet[]` parameters do not correspond to SQL parameters, so you must not include them in the parameter argument list of a CALL statement.

The calling application can retrieve multiple rows of data from the `java.sql.ResultSet[]` parameters. For information about how to process result sets in different applications, see:

- Returning Result Sets in trafci
- Returning Result Sets in an ODBC Client
- Returning Result Sets in a JDBC Client

8.5. Calling SPJs in trafci

In trafci, you can invoke an SPJ by issuing a CALL statement directly or by preparing and executing a CALL statement.

Use named or unnamed parameters anywhere in the argument list of an SPJ invoked in trafci. A named parameter is set by the SET PARAM command, and an unnamed parameter is set by the USING clause of the EXECUTE statement.

You must use a parameter for an OUT or INOUT parameter argument. trafci displays all output parameter values and result sets after you issue the CALL statement. The procedure call changes the value of a named parameter that you use as an OUT or INOUT parameter.

For more information about named and unnamed parameters, see the Trafodion Command Interface Guide.

8.5.1. Using Named Parameters

In an trafci session, invoke the SPJ named TOTALPRICE, which has two IN parameters and one INOUT parameter.

This SPJ accepts the quantity, shipping speed, and price of an item, calculates the total price, including tax and shipping charges, and returns the total price. For more information, see the TOTALPRICE Procedure.

Set the input value for the INOUT parameter by entering a SET PARAM command before calling the SPJ:

```
SQL> SET PARAM ?p 10 ;

SQL> CALL trafodion.sales.totalprice( 23, 'standard', ?p ) ;
```

The CALL statement returns the total price of the item:

```
p
--------------------
253.97

--- SQL operation complete.
```

The value of the named parameter, ?p, changes from 10 to the returned value, 253.97:

```
SQL> SHOW PARAM

p 253.97
```

8.5.2. Using Unnamed Parameters

In an trafci session, invoke the SPJ named TOTALPRICE by preparing and executing a CALL statement. The INOUT parameter accepts a value that is set by the USING clause of the EXECUTE statement and returns the total price:

```
SQL> PREPARE stmt1 FROM CALL trafodion.sales.totalprice( 50, 'nextday', ? ) ;

--- SQL command prepared.

SQL> EXECUTE stmt1 USING 2.25 ;
```

The output of the prepared CALL statement is:

```
PRICE
--------------------
136.77

--- SQL operation complete.
```

In an trafci session, invoke the SPJ named TOTALPRICE again by preparing and executing a CALL statement in which all three parameters accept values that are set by the USING clause of the EXECUTE statement. The INOUT parameter returns the total price:

```
SQL> PREPARE stmt2 FROM CALL trafodion.sales.totalprice( ?, ?, ? ) ;

--- SQL command prepared.

SQL> EXECUTE stmt2 USING 3, 'economy', 16.99 ;
```

The output of the prepared CALL statement is:

```
PRICE
--------------------

57.13

--- SQL operation complete.
```

8.5.3. Returning Result Sets in trafci

If a CALL statement returns result sets, trafci displays column headings and data for each returned result set in the same format as SELECT statements. For example, this CALL statement returns an output parameter for the number of orders and two result sets in the trafci session:

```
SQL> CALL trafodion.sales.ordersummary( '01-01-2011', ? ) ;

NUM_ORDERS
--------------------
13

ORDERNUM NUM_PARTS             AMOUNT               ORDER_DATE LAST_NAME
-------- -------------------- -------------------- ---------- --------------------
  100210                    4             19020.00 2011-04-10 HUGHES
  100250                    4             22625.00 2011-01-23 HUGHES
  101220                    4             45525.00 2011-07-21 SCHNABL
     ...                  ...                  ...        ... ...

--- 13 row(s) selected.

ORDERNUM PARTNUM UNIT_PRICE QTY_ORDERED PARTDESC
-------- ------- ---------- ----------- ------------------
  100210     244    3500.00           3 PC GOLD, 30 MB
  100210    2001    1100.00           3 GRAPHIC PRINTER,M1
  100210    2403     620.00           6 DAISY PRINTER,T2
     ...     ...        ...         ... ...

--- 70 row(s) selected.

--- SQL operation complete.
```

For other result set examples, see Appendix A: Sample SPJs.

8.6. Calling SPJs From an ODBC Client Application

You can execute a CALL statement in an ODBC client application. Microsoft ODBC requires that you put the CALL statement in an escape clause:

```
{ CALL procedure-name ( [ parameter ] [ , [ parameter ] ] ... ) }
```

For IN or INOUT parameters, use a literal or a parameter marker (?). You cannot use an empty string as an IN or INOUT parameter in the argument list. If you specify a literal for an INOUT parameter, the driver discards the output value.

For OUT parameters, you can use only a parameter marker (?). You must bind all parameter markers with the `SQLBindParameter` function before you can execute the CALL statement.

In this example, a CALL statement is executed from an ODBC client application:

```
/* Declare variables. */
SQLHSTMT hstmt ;
SQL_NUMERIC_STRUCT salary ;
SDWORD cbParam = SQL_NTS ;

/* Bind the parameter markers. */
SQLBindParameter( hstmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC, SQL_NUMERIC, 4, 0, 202, 0,
&cbParam ) ;
SQLBindParameter( hstmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_FLOAT, 0, 0, 5.5, 0,
&cbParam ) ;
SQLBindParameter( hstmt, 3, SQL_PARAM_OUTPUT, SQL_C_NUMERIC, SQL_NUMERIC, 8, 2, &
salary, 0, &cbParam ) ;

/* Execute the CALL statement. */
SQLExecDirect( hstmt, "{ CALL trafodion.persnl.adjustsalary( ?, ?, ? ) }", SQL_NTS ) ;
```

8.6.1. Returning Result Sets in an ODBC Client Application

This example shows how an ODBC client application processes the result sets returned by a CALL statement. The `SQLMoreResults()` function closes the current result set and moves processing to the next available result set.

The Trafodion ODBC API does not currently support interleaved result set processing, where more than one returned result set can be open at a time.

```
/* Allocate a statement handle */
SQLHSTMT s ;

RETCODE rc = SQLAllocHandle( SQL_HANDLE_STMT, myConnection, &s ) ;

/* Prepare a CALL */
char *stmtText = "{ CALL trafodion.sales.ordersummary( '01-01-2011', ? ) } ";
rc = SQLPrepare( s, (SQLCHAR *) stmtText, strlen( stmtText ) ) ;

/* Bind the output parameter */
_int64 num_orders = 0 ;
SQLINTEGER indicator ;

rc = SQLBindParameter( s
                     , 2
             , SQL_PARAM_OUTPUT
             , SQL_C_SBIGINT
             , SQL_BIGINT
             , 0
             , 0
             , &num_orders
             , 0
             , &indicator
             ) ;

/* Execute the CALL */
rc = SQLExecute( s ) ;

/* Process all returned result sets. The outer while loop repeats */
/* until there are no more result sets. */
while ( ( rc = SQLMoreResults( s ) ) != SQL_NO_DATA )
{
  /* The inner while loop processes each row of the current result set */
  while ( SQL_SUCCEEDED( rc = SQLFetch( hStmt ) ) )
  {
    /* Process the row */
  }
}
```

8.7. Calling SPJs From a JDBC Client Application

You can execute a CALL statement in a JDBC client application by using the JDBC CallableStatement interface. The HP JDBC Type 4 driver requires that you put the CALL statement in an escape clause:

```
{ CALL procedure-name ( [ parameter [ { , parameter } ... ] ] ) }
```

Set input values for IN and INOUT parameters by using the `set_type()` methods of the `CallableStatement` interface.

Retrieve output values from OUT and INOUT parameters by using the `get_type()` methods of the `CallableStatement` interface.

If the parameter mode is OUT or INOUT, then you must register the parameter as an output parameter by using the `registerOutParameter()` method of the `CallableStatement` interface before executing the CALL statement.

In this example, a CALL statement is executed from a JDBC client application:

```
CallableStatement stmt =
   con.prepareCall( "{ CALL trafodion.persnl.adjustsalary( ?, ?, ? ) }" ) ;

stmt.setBigDecimal( 1, 202 ) ; // x = 202
stmt.setDouble( 2, 5.5 ) ;     // y = 5.5
stmt.registerOutParameter( 3, java.sql.Types.NUMERIC ) ;
stmt.execute() ;

BigDecimal z = stmt.getBigDecimal( 3 ) ; // Retrieve the value of the OUT parameter
```

8.7.1. Returning Result Sets in a JDBC Client Application

This example shows serial result set processing in a JDBC client application where the result sets are processed in order and one at a time after the CALL statement executes. The `java.sql.Statement.getMoreResults()` method closes the current result set and moves processing to the next available result set.

```
// Prepare a CALL statement
java.sql.CallableStatement s =
   myConnection.prepareCall( "{ CALL trafodion.sales.ordersummary( '01-01-2011', ? ) }"
) ;

// Register an output parameter
s.registerOutParameter( 1, java.sql.Types.BIGINT ) ;

// Execute the CALL
boolean rsAvailable = s.execute() ;

// Process all returned result sets. The outer while loop continues
// until there are no more result sets.
while ( rsAvailable )
{
   // The inner while loop processes each row of the current result set
   java.sql.ResultSet rs = s.getResultSet() ;
   while ( rs.next() )
   {
      // Process the row
   }

   rsAvailable = s.getMoreResults() ;
}
```

This example shows how a JDBC client application can have more than one stored procedure result set open at a given time. The `java.sql.Statement.getMoreResults(int)` method uses its input argument to decide whether currently open result sets should remain open or be closed before the next result set is made available.

```
// Prepare a CALL statement
java.sql.CallableStatement s =
   myConnection.prepareCall ( "{ CALL trafodion.sales.ordersummary( '01-01-2011', ? )
}" ) ;

// Register an output parameter
s.registerOutParameter( 1, java.sql.Types.BIGINT ) ;

// Execute the CALL
s.execute() ;

// Open the FIRST result set
java.sql.ResultSet firstRS = s.getResultSet() ;

// Open the SECOND result set but do not close the FIRST
s.getMoreResults( java.sql.Statement.KEEP_CURRENT_RESULT ) ;
java.sql.ResultSet secondRS = s.getResultSet() ;

// The outer loop processes each row of the FIRST result set while
( firstRS.next())
{
   // Process a row from the FIRST result set
   // The inner loop processes some number of rows from the SECOND
   // result set. The number depends on data extracted from the
   // current row of the FIRST result set.
   for ( int i = 0 ; i < NUM_ROWS_TO_PROCESS ; i++ )
   {
      // Process a row from the SECOND result set
      secondRS.next() ;
   }
}
```

Chapter 9. Performance and Troubleshooting

This chapter describes how to improve and monitor the performance of SPJs on Trafodion and provides guidelines for troubleshooting common problems:

- Troubleshooting Common Problems
- Performance Tip
- Displaying an Execution Plan of a CALL Statement

9.1. Troubleshooting Common Problems

To resolve problems that occur when you register or execute an SPJ, follow these guidelines:

- Note the SQLCODE or SQLSTATE value of the error messages and locate the information in the [Trafodion Messages Manual], which provides cause, effect, and recovery information for all SQL errors.
- Check that the user has the appropriate permissions to create or call the SPJ. See these sections:
  - Required Privileges for Creating or Dropping an SPJ.
  - Required Privileges for Calling an SPJ.
- Check the code of the SPJ method. See Develop SPJ Methods. Fix any problems.
- If you successfully compiled, deployed, and registered the SPJ but are receiving errors when calling the SPJ, check that the output parameters in the Java method are specified as arrays. See Returning Output Values From the Java Method.
- Verify that someone did not alter the library by selecting a JAR file that contains a different class name, method name, or method signature than the original JAR file, without dropping and re-creating the SPJ.
- Check the syntax of the CALL statement in the application. See Execute SPJs. Fix any problems.
- If the SPJ is supposed to return result sets, but the result sets are not being returned to the calling application, then check that the SPJ method does not explicitly close a `java.sql.Connection` object. See Use of java.sql.Connection Objects.
- If a `java.lang.ArrayIndexOutOfBoundsException` occurs, then check that the SPJ method is not trying to insert more than one array element into a `java.sql.ResultSets[]` array. For more information, see Returning Stored Procedure Result Sets.
- To identify Java-related errors, execute the SPJ method outside Trafodion by invoking the Java method directly in a Java application that you run on a client workstation, using the Trafodion JDBC Type-4 driver to connect to Trafodion.

- If you are using JDBC tracing and logging, then follow these guidelines:
  - Execute the SPJ method outside the database by invoking the Java method directly from a Java application that you run on a client workstation, using the Trafodion JDBC Type-4 driver to connect to the Trafodion platform.
  - Verify that the file directory specified in the `T4LogFile` property exists on the client workstation and that you have write access to it.

9.2. Performance Tip

To ensure the optimal performance of SPJs on Trafodion, avoid nesting CALL statements in an SPJ method, which wastes resources and might diminish performance. For more information, see Nested Java Method Invocations.

9.3. Displaying an Execution Plan of a CALL Statement

An execution plan reveals how a CALL statement was optimized. You can display all or part of the execution plan for a CALL statement by using the EXPLAIN statement or function.

9.3.1. Using the EXPLAIN Statement

Suppose that you want to display the execution plan for this CALL statement:

```
CALL trafodion.persnl.adjustsalary( 202, 5.5, ? ) ;
```

Enter this EXPLAIN statement in an trafci session:

```
SQL> PREPARE spj1 FROM CALL trafodion.persnl.adjustsalary( 202, 5.5, ? ) ;

--- SQL command prepared.

SQL> EXPLAIN spj1 ;

------------------------------------------------------------------- PLAN SUMMARY
MODULE_NAME ............. DYNAMICALLY COMPILED
STATEMENT_NAME .......... SPJ1
PLAN_ID ................. 212206487012085509
ROWS_OUT ................ 1
EST_TOTAL_COST .......... 0
STATEMENT ............... CALL trafodion.persnl.adjustsalary( 202, 5.5, ? )

------------------------------------------------------------------- NODE LISTING
ROOT ================================== SEQ_NO 2         ONLY CHILD 1
REQUESTS_IN ............. 1
```

```
ROWS_OUT ................. 1
EST_OPER_COST ............ 0
EST_TOTAL_COST ........... 0
DESCRIPTION
  max_card_est ........... 1
  fragment_id ............ 0
  parent_frag ............ (none)
  fragment_type .......... master
  statement_index ........ 0
  affinity_value ......... 3,466,211,324
  max_max_cardinality .... 1
  total_overflow_size .... 0.00 KB xn_autoabort_interval -1
  plan_version ........... 2,500
  LDAP_USERNAME ..........   sqluser_admin
  NVCI_PROCESS ........... ON
  SHOWCONTROL_UNEXTERNALI  OFF
  SCHEMA ................. TRAFODION.INVENT
  CATALOG ................ TRAFODION
  PRIORITY ............... 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA.ASSOC2DS)
  PRIORITY ............... 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA.DATASOURCES)
  PRIORITY ............... 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA.ENVIRONMENTVALUES)
  PRIORITY ............... 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA.NAME2ID)
  PRIORITY ............... 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA.RESOURCEPOLICIES)
  select_list ............ NUMERIC(8,2) SIGNED
  input_variables ........ ?

CALL ================================== SEQ_NO 1         NO CHILDREN
TABLE_NAME ............... TRAFODION.PERSNL.ADJUSTSALARY
REQUESTS_IN .............. 1
ROWS_OUT ................. 1
EST_OPER_COST ............ 0
EST_TOTAL_COST ........... 0
DESCRIPTION
  max_card_est ........... -1
  fragment_id ............ 0
  parent_frag ............ (none)
  fragment_type .......... master
  routine_name ........... TRAFODION.PERSNL.ADJUSTSALARY
  parameter_modes ........ I I O
  sql_access_mode ........ MODIFIES SQL DATA
  external_name .......... adjustSalary
  library ................ TRAFODION.PERSNL.PAYROLL
  external_file .......... Payroll
  signature .............. (Ljava/math/BigDecimal;D[Ljava/math/BigDecimal;)V
  language ............... JAVA
  parameter_style ........ JAVA
  external_security ...... INVOKER
  max_result_sets ........ 0
  parameters ............. cast(202), cast(cast((cast(5.5) / cast(10)))), NUMERIC(8,2)
SIGNED

--- SQL operation complete.

SQL>
```

The EXPLAIN statement generates and displays all the columns of the result table of the EXPLAIN function. For the syntax of the EXPLAIN statement, see the Trafodion SQL Reference Manual.

9.3.2. Using the EXPLAIN Function

You can also prepare the CALL statement and select specific columns from the result table of the EXPLAIN function, as shown below:

```
SQL> PREPARE spj1 FROM CALL trafodion.persnl.adjustsalary( 202, 5.5, ? ) ;

--- SQL command prepared.

SQL> SELECT SUBSTRING( operator, 1, 8 ) AS "OPERATOR", operator_cost,
+> SUBSTRING( description, 1, 500 ) AS "DESCRIPTION"
+> FROM TABLE ( EXPLAIN( NULL, 'SPJ1' ) ) ;

OPERATOR OPERATOR_COST  DESCRIPTION
-------- --------------
--------------------------------------------------------------------------------
CALL                0.0 max_card_est: -1 fragment_id: 0 parent_frag: (none)
fragment_type: master routine_name:
TRAFODION.PERSNL.ADJUSTSALARY parameter_modes: I I O sql_access_mode: MODIFIES SQL DATA
external_name: adjustSalary
library: TRAFODION.PERSNL.PAYROLL external_file: Payroll signature: (Ljava/math
/BigDecimal;D[Ljava/math/BigDecimal;)V
language: JAVA parameter_style: JAVA external_security: INVOKER max_result_sets: 0
parameters: cast(202),
cast(cast((cast(5.5) / cast(10)))), NUMERIC(8,2) SIGNED
ROOT                0.0 max_card_est: 1 fragment_id: 0 parent_frag: (none)
fragment_type: master statement_index:
  0 affinity_value: 3466211324 max_max_cardinality: 1 total_overflow_size: 0.00 KB
statement: call
trafodion.persnl.adjustsalary( 202, 5.5 ,? ) xn_autoabort_interval: -1 plan_version:
2500 LDAP_USERNAME: sqluser_admin
NVCI_PROCESS: ON SHOWCONTROL_UNEXTERNALIZED_ATTRS: OFF SCHEMA: TRAFODION.INVENT
CATALOG: TRAFODION PRIORITY: 9 (for table
SYSTEM_CATALOG.MXCS_SCHEMA.ASSOC2DS) PRIORITY: 9 (for table SYSTEM_CATALOG.MXCS_SCHEMA
.D

--- 2 row(s) selected. SQL>
```

For a CALL statement, the OPERATOR column of the result table contains a row named CALL. The DESCRIPTION column contains special token pairs for the CALL operator. For descriptions of the token pairs, see this table:

Table 3. Token Pairs Description

Token	Token Description	Data Type
max_card_est	The upper limit for the operator cardinality in the query tree.	integer
fragment_id	A sequential number assigned to the fragment. 0 is always the master executor, and 1 is reserved for the Explain plan. Numbers 2 to *n* are ESP or storage-engine fragments.	integer
parent_frag	The fragment_id for the parent fragment of the current fragment. The value is (none) for the master executor.	integer
fragment_type	Type of fragment, which can be either master, ESP, or storage engine.	text
routine_name	ANSI name of the procedure.	text
parameter_modes	A sequence of characters that specifies SQL parameter modes for the procedure. I is used for an IN parameter, O for an OUT parameter, and N for an INOUT parameter. Characters are separated by a single space. The value none is returned if the procedure has no SQL parameters.	text
sql_access_mode	SQL access mode of the procedure.	text
external_name	Java method name.	text
library	ANSI name of the library object that maps to the procedure's JAR file.	text
external_file	Java class name, possibly prefixed by a package name, that contains the SPJ method.	text
signature	Java signature of the SPJ method in internal Java Virtual Machine (JVM) format.	text
language	Language in which the SPJ method is written, which is always Java.	text
parameter_style	Convention of passing parameter arguments to the stored procedure, which conforms to the Java language for SPJs.	text
external_security	External security of the stored procedure, indicating the privileges or rights that users have when executing (or calling) the procedure. The value is either INVOKER or DEFINER. For more information, see Understand External Security.	text
max_result_sets	The maximum number of result sets that this procedure can return.	integer
parameters	The parameter arguments that are passed to or from the procedure.	text

For the syntax of the EXPLAIN function, see the Trafodion SQL Reference Manual.

Chapter 10. A Sample SPJs

This appendix presents the SPJs that are shown in examples throughout this manual. The class files that contain the SPJ methods use JDBC method calls to access a sample database. For information about the sample database, see Sample Database below.

- Procedures in the SALES Schema
- Procedures in the PERSNL Schema
- Procedures in the INVENT Schema

You can download each source sample by clicking the link provided with the sample name. For example, click on Sales.java to download the sample sales class source file.

+ You can access the complete source directory at:
http://trafodion.incubator.apache.org/docs/spj_guide/resources/source/

10.1. Procedures in the SALES Schema

The Sales class contains these SPJ methods, which are useful for tracking orders and managing sales:

- LOWERPRICE Procedure
- DAILYORDERS Procedure
- MONTHLYORDERS Procedure
- TOTALPRICE Procedure
- PARTDATA Procedure
- ORDERSUMMARY Procedure

Those methods are registered as stored procedures in the SALES schema. Example 1 shows the code of the Sales.java file.

Example 1: `Sales.java` *- The Sales Class*

```
import java.sql.* ;
import java.math.* ;

public class Sales
{
   // The LOWERPRICE procedure determines which items are selling poorly (that
   // is, have less than 50 orders) and lowers the price of these items in the
   // database by 10 percent.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#lowerprice-
procedure
   // for more documentation.
   public static void lowerPrice() throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

         PreparedStatement getParts =
            conn.prepareStatement( "SELECT p.partnum, "
                                 + "SUM(qty_ordered) AS qtyOrdered "
                                 + "FROM trafodion.sales.parts p "
                                 + "LEFT JOIN trafodion.sales.odetail o "
                                 + "ON p.partnum = o.partnum "
                                 + "GROUP BY p.partnum"
                                 ) ;

         PreparedStatement updateParts =
            conn.prepareStatement( "UPDATE trafodion.sales.parts "
                                 + "SET price = price * 0.9 "
                                 + "WHERE partnum = ?"
```

```
                                    ) ;

         ResultSet rs = getParts.executeQuery() ;
         while ( rs.next() )
         {
            BigDecimal qtyOrdered = rs.getBigDecimal( 2 ) ;

            if (( qtyOrdered == null ) || ( qtyOrdered.intValue() < 50 ) )
            {
               BigDecimal partnum = rs.getBigDecimal( 1 ) ;
               updateParts.setBigDecimal( 1, partnum ) ;
               updateParts.executeUpdate() ;
            }
         }

         rs.close() ;
         conn.close() ;

   }

   // The DAILYORDERS procedure accepts a date and returns the number of
   //orders on that date to an output parameter.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#dailyorders-
procedure
   // for additional documenation.
   public static void numDailyOrders( Date date
                                    , int[] numOrders
                                    ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement getNumOrders =
         conn.prepareStatement( "SELECT COUNT(order_date) "
                              + "FROM trafodion.sales.orders "
                              + "WHERE order_date = ?"
                              ) ;

      getNumOrders.setDate( 1, date ) ;

      ResultSet rs = getNumOrders.executeQuery() ;
      rs.next() ;

      numOrders[0] = rs.getInt( 1 ) ;

      rs.close() ;
      conn.close() ;

   }

   // The MONTHLYORDERS procedure accepts an integer representing the month
   // and returns the number of orders during that month to an output parameter.
   //
```

```
    // See
http://trafodion.incubator.apache.org/docs/spj_guide/index.html#monthlyorders-procedure
    // for more documentation.
    public static void numMonthlyOrders( int month
                                       , int[] numOrders
                                       ) throws SQLException

    {
       if ( month < 1 || month > 12 )
       {
          throw new SQLException( "Invalid value for month. "
                                + "Retry the CALL statement "
                                + "using a number from 1 to 12 "
                                + "to represent the month."
                                , "38001"
                                ) ;
       }

       Connection conn =
          DriverManager.getConnection( "jdbc:default:connection" ) ;

       PreparedStatement getNumOrders =
          conn.prepareStatement( "SELECT COUNT( month( order_date ) ) "
                               + "FROM trafodion.sales.orders "
                               + "WHERE month( order_date ) = ?"
                               ) ;

       getNumOrders.setInt( 1, month ) ;

       ResultSet rs = getNumOrders.executeQuery() ;
       rs.next() ;

       numOrders[0] = rs.getInt(1) ;

       rs.close() ;
       conn.close();

    }

    // The TOTALPRICE procedure accepts the quantity, shipping speed, and price
    // of an item, calculates the total price, including tax and shipping
    // charges, and returns the total price to an input/output parameter.
    //
    // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#totalprice-
procedure
    // for more documentation.
    public static void totalPrice( BigDecimal qtyOrdered
                                 , String shippingSpeed
                                 , BigDecimal[] price
                                 ) throws SQLException
    {
       BigDecimal shipcharge = new BigDecimal( 0 ) ;

       if ( shippingSpeed.equals( "economy" ) )
```

```
      {
         shipcharge = new BigDecimal( 1.95 ) ;
      }
      else if ( shippingSpeed.equals( "standard" ) )
      {
         shipcharge = new BigDecimal( 4.99 ) ;
      }
      else if ( shippingSpeed.equals( "nextday" ) )
      {
         shipcharge = new BigDecimal( 14.99 ) ;
      }
      else
      {
         throw new SQLException( "Invalid value for shipping speed. "
                               + "Retry the CALL statement using "
                               + "'economy' for 7 to 9 days, "
                               + "'standard' for 3 to 5 days, or "
                               + "'nextday' for one day."
                               , "38002"
                               ) ;
      }

      BigDecimal subtotal   = price[0].multiply( qtyOrdered ) ;
      BigDecimal tax        = new BigDecimal( 0.0825 ) ;
      BigDecimal taxcharge  = subtotal.multiply( tax ) ;
      BigDecimal charges    = taxcharge.add( shipcharge ) ;
      BigDecimal totalprice = subtotal.add( charges ) ;

      totalprice = totalprice.setScale( 2, BigDecimal.ROUND_HALF_EVEN ) ;
      price[0] = totalprice ;

   }

   // The PARTDATA procedure accepts a part number and returns this
   // information about the part:
   //
   // * Part description, price, and quantity available as output parameters.
   // * A result set that contains rows from the ORDERS table about when this part was
ordered.
   // * A result set that contains rows from the PARTLOC table, listing locations that
have this
   //   part in stock and the quantity they have on hand.
   // * A result set that contains rows from the PARTSUPP table for suppliers who carry
this part.
   // * A result set that contains rows from the EMPLOYEE table for sales reps who have
sold this part.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#partdata-
procedure
   // for more documentation.
   public static void partData( int partNum
                              , String[] partDescription
                              , BigDecimal[] unitPrice
                              , int[] qtyAvailable
```

```
                              , ResultSet[] orders
                              , ResultSet[] locations
                              , ResultSet[] suppliers
                              , ResultSet[] reps
                              ) throws SQLException
   {

      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      // Retrieve detail about this part into the output parameters
      PreparedStatement getPartInfo =
        conn.prepareStatement( "SELECT P.partdesc, P.price, P.qty_available "
                             + "FROM trafodion.sales.parts P "
                             + "WHERE partnum = ? "
                             ) ;

      getPartInfo.setInt( 1, partNum ) ;

      ResultSet rs = getPartInfo.executeQuery() ;
      rs.next() ;

      partDescription[0] = rs.getString( 1 ) ;
      unitPrice[0]       = rs.getBigDecimal( 2 ) ;
      qtyAvailable[0]    = rs.getInt( 3 ) ;

      rs.close();

      // Return a result set of rows from the ORDERS table listing orders
      // that included this part. Each ORDERS row is augmented with the
      // quantity of this part that was ordered.
      PreparedStatement getOrders =
         conn.prepareStatement( "SELECT O.*, QTY.QTY_ORDERED "
                              + "FROM   trafodion.sales.orders O "
                              + "     , ( select ordernum, sum(qty_ordered) as
QTY_ORDERED "
                              + "         from trafodion.sales.odetail "
                              + "         where partnum = ? "
                              + "         group by ordernum ) QTY "
                              + "WHERE O.ordernum = QTY.ordernum "
                              + "ORDER BY O.ordernum "
                              ) ;


       getOrders.setInt( 1, partNum ) ;
       orders[0] = getOrders.executeQuery() ;

       // Return a result set of rows from the PARTLOC table listing
       // locations that have this part in stock and the quantity they
       // have on hand.
       PreparedStatement getLocations =
          conn.prepareStatement( "SELECT * "
                               + "FROM trafodion.invent.partloc "
                               + " WHERE partnum = ? "
```

```
                                ) ;

        getLocations.setInt( 1, partNum ) ;
        locations[0] = getLocations.executeQuery() ;

        // Return a result set of rows from the PARTSUPP table listing
        // suppliers who supply this part.
        PreparedStatement getSuppliers =
           conn.prepareStatement( "SELECT * "
                                + "FROM trafodion.invent.partsupp "
                                + "WHERE partnum = ? "
                                ) ;

        getSuppliers.setInt( 1, partNum ) ;
        suppliers[0] = getSuppliers.executeQuery() ;

        // Return a result set of rows from the EMPLOYEE table listing
        // sales reps that have sold this part.
        PreparedStatement getReps =
           conn.prepareStatement( "SELECT * "
                                + "FROM trafodion.persnl.employee "
                                + "WHERE empnum in ( SELECT O.salesrep "
                                + "                  FROM trafodion.sales.orders O, "
                                + "                       trafodion.sales.odetail D "
                                + "                  WHERE D.partnum = ? "
                                + "                    AND O.ordernum = D.ordernum ) "
                                + "ORDER BY empnum "
                                ) ;

         getReps.setInt( 1, partNum ) ;
         reps[0] = getReps.executeQuery() ;

    }

    // The ORDERSUMMARY procedure accepts a date, which is formatted as a
    // string, and returns this information about the orders on or after that
    // date:
    //
    // * The number of orders as an output parameter
    // * A result set that contains one row for each order. Each row contains
    //   fields for the order number, the number of parts ordered, total dollar
    //   amount, order date, and the name of the sales representative.
    // * A result set that contains details about each order. Each order has
    //   one or more rows that provide details about the ordered parts. Each row
    //   contains fields for the order number, part number, unit price, quantity
    //   ordered, and part description.
    //
    // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#ordersummary-
procedure
    // for more documentation.
    public static void orderSummary( java.lang.String onOrAfter
                                   , long[] numOrders
                                   , java.sql.ResultSet[] orders
                                   , java.sql.ResultSet[] detail
```

```
                                    ) throws SQLException
   {
      java.lang.String s ;

      java.sql.Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      // Get the number of orders on or after this date
      s =   "SELECT COUNT(ordernum) FROM trafodion.sales.orders "
          + "WHERE order_date >= CAST(? AS DATE) "
          ;

      java.sql.PreparedStatement ps1 = conn.prepareStatement( s ) ;
      ps1.setString( 1, onOrAfter ) ;

      java.sql.ResultSet rs = ps1.executeQuery() ;
      rs.next() ;

      numOrders[0] = rs.getLong( 1 ) ;
      rs.close() ;

      // Open a result set for order num, order info rows
      s =   "SELECT amounts.*, orders.order_date, emps.last_name "
          + "FROM ( SELECT "
          + "         o.ordernum "
          + "       , COUNT(d.partnum) AS num_parts "
          + "       , SUM(d.unit_price * d.qty_ordered) AS amount "
          + "       FROM trafodion.sales.orders o, trafodion.sales.odetail d "
          + "       WHERE o.ordernum = d.ordernum "
          + "          AND o.order_date >= CAST(? AS DATE) "
          + "       GROUP BY o.ordernum "
          + "      ) amounts "
          + "      , trafodion.sales.orders orders "
          + "      , trafodion.persnl.employee emps "
          + "WHERE amounts.ordernum = orders.ordernum "
          + "   AND orders.salesrep = emps.empnum "
          + "ORDER BY orders.ordernum "
          ;

       java.sql.PreparedStatement ps2 = conn.prepareStatement( s ) ;
       ps2.setString( 1, onOrAfter ) ;
       orders[0] = ps2.executeQuery() ;

       // Open a result set for order detail rows
       s =   "SELECT d.*, p.partdesc "
           + "FROM trafodion.sales.odetail d, trafodion.sales.parts p,
trafodion.sales.orders O "
           + "WHERE d.partnum = p.partnum AND d.ordernum = o.ordernum "
           + "   AND o.order_date >= CAST(? AS DATE) "
           + "ORDER BY d.ordernum "
           ;

       java.sql.PreparedStatement ps3 = conn.prepareStatement( s ) ;
       ps3.setString( 1, onOrAfter ) ;
```

```
        detail[0] = ps3.executeQuery() ;

    }
}
```

See the following sections for more information about each SPJ method.

10.1.1. LOWERPRICE Procedure

The LOWERPRICE procedure determines which items are selling poorly (that is, have less than 50 orders) and lowers the price of these items in the database by 10 percent.

Java Method: lowerPrice()

Example 2: `lowerPrice()` *Method*

```
// The LOWERPRICE procedure determines which items are selling poorly (that
// is, have less than 50 orders) and lowers the price of these items in the
// database by 10 percent.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#lowerprice-
procedure
// for more documentation.
public static void lowerPrice() throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement getParts =
         conn.prepareStatement( "SELECT p.partnum, "
                              + "SUM(qty_ordered) AS qtyOrdered "
                              + "FROM trafodion.sales.parts p "
                              + "LEFT JOIN trafodion.sales.odetail o "
                              + "ON p.partnum = o.partnum "
                              + "GROUP BY p.partnum"
                              ) ;

      PreparedStatement updateParts =
         conn.prepareStatement( "UPDATE trafodion.sales.parts "
                              + "SET price = price * 0.9 "
                              + "WHERE partnum = ?"
                              ) ;

      ResultSet rs = getParts.executeQuery() ;
      while ( rs.next() )
      {
         BigDecimal qtyOrdered = rs.getBigDecimal( 2 ) ;

         if (( qtyOrdered == null ) || ( qtyOrdered.intValue() < 50 ) )
         {
            BigDecimal partnum = rs.getBigDecimal( 1 ) ;
            updateParts.setBigDecimal( 1, partnum ) ;
            updateParts.executeUpdate() ;
         }
      }

      rs.close() ;
      conn.close() ;

}
```

Creating the Procedure: LOWERPRICE

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 4. Create Procedure Settings: LOWERPRICE Procedure

Attribute	Definition
procedure-ref	`trafodion.sales.lowerprice`
sql-parameter	Not applicable.
external name	`EXTERNAL NAME 'Sales.lowerPrice'`
library	`LIBRARY trafodion.sales.sales`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`MODIFIES SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.lowerprice( )
   EXTERNAL NAME 'Sales.lowerPrice'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   MODIFIES SQL DATA
   ;
```

Calling the Procedure: LOWERPRICE

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the LOWERPRICE procedure in trafci:

```
SQL> CALL trafodion.sales.lowerprice() ;

--- SQL operation complete.
```

To view the prices and quantities of items in the database with 50 or fewer orders, issue this query before and after calling the LOWERPRICE procedure:

```
SELECT *
FROM
   ( SELECT p.partnum
   , SUM(qty_ordered) AS qtyOrdered
   , p.price
     FROM trafodion.sales.parts p
     LEFT OUTER JOIN trafodion.sales.odetail o ON p.partnum = o.partnum
     GROUP BY p.partnum, p.price
   ) AS allparts
WHERE qtyOrdered < 51
ORDER BY partnum ASC
;
```

The LOWERPRICE procedure lowers the price of items with 50 or fewer orders by 10 percent in the database. For example, part number 3103, the LASER PRINTER, X1, has 40 orders and a price of 3402.00:

```
PARTNUM QTYORDERED            PRICE
------- -------------------- ----------
    212                   20    2025.00
    244                   47    2430.00
    255                   38    3240.00
   2002                   46    1215.00
   2405                   18     643.95
   3103                   40    3402.00
    ...                  ...        ...

--- 17 row(s) selected.
```

The invocation of LOWERPRICE lowers the price of this item from 3402.00 to 3061.80:

```
PARTNUM QTYORDERED            PRICE
------- -------------------- ----------
    212                   20    1822.50
    244                   47    2187.00
    255                   38    2916.00
   2002                   46    1093.50
   2405                   18     579.55
   3103                   40    3061.80
... ... ...

--- 17 row(s) selected.
```

10.1.2. DAILYORDERS Procedure

The DAILYORDERS procedure accepts a date and returns the number of orders on that date to an output parameter.

Java Method: numDailyOrders()

Example 3: `numDailyOrders()` *Method*

```
// The DAILYORDERS procedure accepts a date and returns the number of
//orders on that date to an output parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#dailyorders-
procedure
// for additional documenation.
public static void numDailyOrders( Date date
                 , int[] numOrders
                 ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getNumOrders =
      conn.prepareStatement( "SELECT COUNT(order_date) "
               + "FROM trafodion.sales.orders "
               + "WHERE order_date = ?"
               ) ;

   getNumOrders.setDate( 1, date ) ;

   ResultSet rs = getNumOrders.executeQuery() ;
   rs.next() ;

   numOrders[0] = rs.getInt( 1 ) ;

   rs.close() ;
   conn.close() ;

}
```

Creating the Procedure: DAILYORDERS

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 5. Create Procedure Settings: DAILYORDERS Procedure

Attribute	Definition
procedure-ref	trafodion.sales.dailyorders
sql-parameter-1	IN date1 DATE
sql-parameter-2	OUT number INTEGER
external name	EXTERNAL NAME 'Sales.numDailyOrders'
library	LIBRARY trafodion.sales.sales
external security	invoker[1] Choice depends on your security requirements, you can select definer instead. For more information, see Understand External Security.
language java	LANGUAGE JAVA
parameter style java	PARAMETER STYLE JAVA
sql access	READS SQL DATA
dynamic result sets	DYNAMIC RESULT SETS 0[1]
transaction	TRANSACTION REQUIRED[1]
determinism	NOT DETERMINISTIC[1]
isolation	ISOLATE[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.dailyorders( IN date1 DATE
                                            , OUT number INTEGER
                                            )
   EXTERNAL NAME 'Sales.numDailyOrders'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
   ;
```

Calling the Procedure: DAILYORDERS

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the DAILYORDERS procedure in trafci:

```
SQL> CALL trafodion.sales.dailyorders( DATE '2011-03-19', ? ) ;
```

The DAILYORDERS procedure determines the total number of orders on a specified date and returns this output in trafci:

```
NUMBER
-----------
          2

--- SQL operation complete.
```

On March 19, 2011, there were two orders.

10.1.3. MONTHLYORDERS Procedure

The MONTHLYORDERS procedure accepts an integer representing the month and returns the number of orders during that month to an output parameter.

Java Method: numMonthlyOrders()

Example 4: `numMonthlyOrders()` *Method*

```
// The MONTHLYORDERS procedure accepts an integer representing the month
// and returns the number of orders during that month to an output parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#monthlyorders-
procedure
// for more documentation.
public static void numMonthlyOrders( int month
                  , int[] numOrders
                  ) throws SQLException

{
   if ( month < 1 || month > 12 )
   {
      throw new SQLException( "Invalid value for month. "
                + "Retry the CALL statement "
                + "using a number from 1 to 12 "
                + "to represent the month."
                , "38001"
                ) ;
   }

   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getNumOrders =
      conn.prepareStatement( "SELECT COUNT( month( order_date ) ) "
               + "FROM trafodion.sales.orders "
               + "WHERE month( order_date ) = ?"
               ) ;

   getNumOrders.setInt( 1, month ) ;

   ResultSet rs = getNumOrders.executeQuery() ;
   rs.next() ;

   numOrders[0] = rs.getInt(1) ;

   rs.close() ;
   conn.close();

}
```

Creating the Procedure: MONTHLYORDERS

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 6. Create Procedure Settings: MONTHLYORDERS Procedure

Attribute	Definition
procedure-ref	trafodion.sales.monthlyorders
sql-parameter-1	IN monthnum INTEGER
sql-parameter-2	OUT ordernum INTEGER
external name	EXTERNAL NAME 'Sales.numMonthlyOrders'
library	LIBRARY trafodion.sales.sales
external security	invoker[1] Choice depends on your security requirements, you can select definer instead. For more information, see Understand External Security.
language java	LANGUAGE JAVA
parameter style java	PARAMETER STYLE JAVA
sql access	READS SQL DATA
dynamic result sets	DYNAMIC RESULT SETS 0[1]
transaction	TRANSACTION REQUIRED[1]
determinism	NOT DETERMINISTIC[1]
isolation	ISOLATE[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.monthlyorders( IN monthnum INTEGER
                                              , OUT ordernum INTEGER
                                              )
   EXTERNAL NAME 'Sales.numMonthlyOrders'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
   ;
```

Calling the Procedure: MONTHLYORDERS

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the MONTHLYORDERS procedure in trafci:

```
SQL> CALL trafodion.sales.monthlyorders( 3, ? ) ;
```

The MONTHLYORDERS procedure determines the total number of orders during a specified month and returns this output in trafci:

```
ORDERNUM
-----------
          4

--- SQL operation complete.
```

In March, there were four orders.

10.1.4. TOTALPRICE Procedure

The TOTALPRICE procedure accepts the quantity, shipping speed, and price of an item, calculates the total price, including tax and shipping charges, and returns the total price to an input/output parameter.

Java Method: totalPrice()

Example 5: `totalPrice()` *Method*

```
// The TOTALPRICE procedure accepts the quantity, shipping speed, and price
// of an item, calculates the total price, including tax and shipping
// charges, and returns the total price to an input/output parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#totalprice-
procedure
// for more documentation.
public static void totalPrice( BigDecimal qtyOrdered
                 , String shippingSpeed
                 , BigDecimal[] price
                 ) throws SQLException
{
   BigDecimal shipcharge = new BigDecimal( 0 ) ;

   if ( shippingSpeed.equals( "economy" ) )
   {
      shipcharge = new BigDecimal( 1.95 ) ;
   }
   else if ( shippingSpeed.equals( "standard" ) )
   {
      shipcharge = new BigDecimal( 4.99 ) ;
   }
   else if ( shippingSpeed.equals( "nextday" ) )
   {
      shipcharge = new BigDecimal( 14.99 ) ;
   }
   else
   {
      throw new SQLException( "Invalid value for shipping speed. "
                + "Retry the CALL statement using "
                + "'economy' for 7 to 9 days, "
                + "'standard' for 3 to 5 days, or "
                + "'nextday' for one day."
                , "38002"
                ) ;
   }

   BigDecimal subtotal   = price[0].multiply( qtyOrdered ) ;
   BigDecimal tax        = new BigDecimal( 0.0825 ) ;
   BigDecimal taxcharge  = subtotal.multiply( tax ) ;
   BigDecimal charges    = taxcharge.add( shipcharge ) ;
   BigDecimal totalprice = subtotal.add( charges ) ;

   totalprice = totalprice.setScale( 2, BigDecimal.ROUND_HALF_EVEN ) ;
   price[0] = totalprice ;

}
```

Creating the Procedure: TOTALPRICE

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 7. Create Procedure Settings: TOTALPRICE Procedure

Attribute	Definition
procedure-ref	`trafodion.sales.totalprice`
sql-parameter-1	`IN gty NUMERIC(18)`
sql-parameter-2	`IN speed VARCHAR(10)`
sql-parameter-3	`INOUT price NUMERIC(18,2)`
external name	`EXTERNAL NAME 'Sales.totalPrice ( java.math.BigDecimal, java.lang.String, java.math.BigDecimal[] )'`
library	`LIBRARY trafodion.sales.sales`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`NO SQL`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.totalprice( IN gty NUMERIC(18)
                                           , IN speed VARCHAR(10)
                                           , INOUT price NUMERIC(18,2)
                                           )
   EXTERNAL NAME 'Sales.totalPrice ( java.math.BigDecimal, java.lang.String,
java.math.BigDecimal[] )'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   NO SQL
   ;
```

Calling the Procedure: TOTALPRICE

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the TOTALPRICE procedure in trafci:

```
SQL> SET PARAM ?p 10 ;
SQL> CALL trafodion.sales.totalprice( 23, 'standard', ?p ) ;
```

The TOTALPRICE procedure calculates the total price of a purchase and returns this output in trafci:

```
p
--------------------
              253.97

--- SQL operation complete.
```

The total price of 23 items, which cost $10 each and which are shipped at the standard rate, is $253.97, including sales tax.

10.1.5. PARTDATA Procedure

The PARTDATA procedure accepts a part number and returns this information about the part:

- Part description, price, and quantity available as output parameters.
- A result set that contains rows from the ORDERS table about when this part was ordered.
- A result set that contains rows from the PARTLOC table, listing locations that have this part in stock and the quantity they have on hand.
- A result set that contains rows from the PARTSUPP table for suppliers who carry this part.
- A result set that contains rows from the EMPLOYEE table for sales reps who have sold this part.

Java Method: partData()

Example 6: `partData()` *Method*

```
// The PARTDATA procedure accepts a part number and returns this
// information about the part:
//
// * Part description, price, and quantity available as output parameters.
// * A result set that contains rows from the ORDERS table about when this part was
ordered.
// * A result set that contains rows from the PARTLOC table, listing locations that
have this
//   part in stock and the quantity they have on hand.
// * A result set that contains rows from the PARTSUPP table for suppliers who carry
this part.
// * A result set that contains rows from the EMPLOYEE table for sales reps who have
sold this part.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#partdata-
procedure
// for more documentation.
public static void partData( int partNum
               , String[] partDescription
               , BigDecimal[] unitPrice
               , int[] qtyAvailable
               , ResultSet[] orders
               , ResultSet[] locations
               , ResultSet[] suppliers
               , ResultSet[] reps
               ) throws SQLException
{

   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;
```

```
    // Retrieve detail about this part into the output parameters
    PreparedStatement getPartInfo =
      conn.prepareStatement( "SELECT P.partdesc, P.price, P.qty_available "
              + "FROM trafodion.sales.parts P "
              + "WHERE partnum = ? "
              ) ;

    getPartInfo.setInt( 1, partNum ) ;

    ResultSet rs = getPartInfo.executeQuery() ;
    rs.next() ;

    partDescription[0] = rs.getString( 1 ) ;
    unitPrice[0]       = rs.getBigDecimal( 2 ) ;
    qtyAvailable[0]    = rs.getInt( 3 ) ;

    rs.close();

    // Return a result set of rows from the ORDERS table listing orders
    // that included this part. Each ORDERS row is augmented with the
    // quantity of this part that was ordered.
    PreparedStatement getOrders =
       conn.prepareStatement( "SELECT O.*, QTY.QTY_ORDERED "
              + "FROM   trafodion.sales.orders O "
              + "     , ( select ordernum, sum(qty_ordered) as QTY_ORDERED "
              + "         from trafodion.sales.odetail "
              + "         where partnum = ? "
              + "         group by ordernum ) QTY "
              + "WHERE O.ordernum = QTY.ordernum "
              + "ORDER BY O.ordernum "
              ) ;


     getOrders.setInt( 1, partNum ) ;
     orders[0] = getOrders.executeQuery() ;

     // Return a result set of rows from the PARTLOC table listing
     // locations that have this part in stock and the quantity they
     // have on hand.
     PreparedStatement getLocations =
        conn.prepareStatement( "SELECT * "
                + "FROM trafodion.invent.partloc "
                + "WHERE partnum = ? "
                ) ;

     getLocations.setInt( 1, partNum ) ;
     locations[0] = getLocations.executeQuery() ;

     // Return a result set of rows from the PARTSUPP table listing
     // suppliers who supply this part.
     PreparedStatement getSuppliers =
        conn.prepareStatement( "SELECT * "
                + "FROM trafodion.invent.partsupp "
                + "WHERE partnum = ? "
```

```
                ) ;

    getSuppliers.setInt( 1, partNum ) ;
    suppliers[0] = getSuppliers.executeQuery() ;

    // Return a result set of rows from the EMPLOYEE table listing
    // sales reps that have sold this part.
    PreparedStatement getReps =
       conn.prepareStatement( "SELECT * "
                + "FROM trafodion.persnl.employee "
                + "WHERE empnum in ( SELECT O.salesrep "
                + "                  FROM trafodion.sales.orders O, "
                + "                       trafodion.sales.odetail D "
                + "                  WHERE D.partnum = ? "
                + "                    AND O.ordernum = D.ordernum ) "
                + "ORDER BY empnum "
                ) ;

     getReps.setInt( 1, partNum ) ;
     reps[0] = getReps.executeQuery() ;

}
```

Creating the Procedure: PARTDATA

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 8. Create Procedure Settings: PARTDATA Procedure

Attribute	Definition
procedure-ref	trafodion.sales.partdata
sql-parameter-1	IN partnum INTEGER
sql-parameter-2	OUT partdesc CHARACTER(18)
sql-parameter-3	OUT price NUMERIC(18,2)
sql-parameter-4	OUT qty_available INTEGER
external name	EXTERNAL NAME 'Sales.partData'
library	LIBRARY trafodion.sales.sales
external security	invoker[1] Choice depends on your security requirements, you can select definer instead. For more information, see Understand External Security.
language java	LANGUAGE JAVA
parameter style java	PARAMETER STYLE JAVA
sql access	READS SQL DATA
dynamic result sets	DYNAMIC RESULT SETS 4
transaction	TRANSACTION REQUIRED[1]
determinism	NOT DETERMINISTIC[1]
isolation	ISOLATE[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.partdata( IN partnum INTEGER
                                         , OUT partdesc CHARACTER(18)
                                         , OUT price NUMERIC(18,2)
                                         , OUT qty_available INTEGER
                                         )
   EXTERNAL NAME 'Sales.partData( int, java.lang.String[], java.math.BigDecimal[],
int[] )'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   DYNAMIC RESULT SETS 4
   READS SQL DATA
   ;
```

Calling the Procedure: PARTDATA

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the PARTDATA procedure in trafci:

```
SQL> CALL trafodion.sales.partdata( 212, ?, ?, ?) ;
```

The PARTDATA procedure returns this information about part 212:

```
PARTDESC           PRICE                QTY_AVAILABLE
------------------ -------------------- -------------
PC SILVER, 20 MB                1822.50          3525

ORDERNUM ORDER_DATE DELIV_DATE SALESREP CUSTNUM QTY_ORDERED
-------- ---------- ---------- -------- ------- --------------------
  400410 2011-03-27 2011-09-01      227    7654                   12
  500450 2011-04-20 2011-09-15      220     324                    8

--- 2 row(s) selected.

LOC_CODE PARTNUM QTY_ON_HAND
-------- ------- -----------
A87          212          18
G87          212          20

--- 2 row(s) selected.

PARTNUM SUPPNUM PARTCOST   QTY_RECEIVED
------- ------- ---------- ------------
    212       1    2000.00           20
    212       3    1900.00           35

--- 2 row(s) selected.

EMPNUM FIRST_NAME      LAST_NAME            DEPTNUM JOBCODE SALARY
------ --------------- -------------------- ------- ------- ----------
   220 JOHN            HUGHES                  3200     300   33000.10
   227 XAVIER          SEDLEMEYER              3300     300   30000.00

--- 2 row(s) selected.

--- SQL operation complete.
```

10.1.6. ORDERSUMMARY Procedure

The ORDERSUMMARY procedure accepts a date, which is formatted as a string, and returns this information about the orders on or after that date:

- The number of orders as an output parameter.
- A result set that contains one row for each order. Each row contains fields for the order number, the number of parts ordered, total dollar amount, order date, and the name of the sales representative.
- A result set that contains details about each order. Each order has one or more rows that provide details about the ordered parts. Each row contains fields for the order number, part number, unit price, quantity ordered, and part description.

Java Method: orderSummary()

Example 7: `orderSummary()` *Method*

```
// The ORDERSUMMARY procedure accepts a date, which is formatted as a
// string, and returns this information about the orders on or after that
// date:
//
// * The number of orders as an output parameter
// * A result set that contains one row for each order. Each row contains
//   fields for the order number, the number of parts ordered, total dollar
//   amount, order date, and the name of the sales representative.
// * A result set that contains details about each order. Each order has
//   one or more rows that provide details about the ordered parts. Each row
//   contains fields for the order number, part number, unit price, quantity
//   ordered, and part description.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#ordersummary-
procedure
// for more documentation.
public static void orderSummary( java.lang.String onOrAfter
                  , long[] numOrders
                  , java.sql.ResultSet[] orders
                  , java.sql.ResultSet[] detail
                  ) throws SQLException
{
   java.lang.String s ;

   java.sql.Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   // Get the number of orders on or after this date
   s =   "SELECT COUNT(ordernum) FROM trafodion.sales.orders "
       + "WHERE order_date >= CAST(? AS DATE) "
       ;
```

```
   java.sql.PreparedStatement ps1 = conn.prepareStatement( s ) ;
   ps1.setString( 1, onOrAfter ) ;

   java.sql.ResultSet rs = ps1.executeQuery() ;
   rs.next() ;

   numOrders[0] = rs.getLong( 1 ) ;
   rs.close() ;

   // Open a result set for order num, order info rows
   s =   "SELECT amounts.*, orders.order_date, emps.last_name "
       + "FROM ( SELECT "
       + "         o.ordernum "
       + "       , COUNT(d.partnum) AS num_parts "
       + "       , SUM(d.unit_price * d.qty_ordered) AS amount "
       + "       FROM trafodion.sales.orders o, trafodion.sales.odetail d "
       + "       WHERE o.ordernum = d.ordernum "
       + "          AND o.order_date >= CAST(? AS DATE) "
       + "       GROUP BY o.ordernum "
       + "      ) amounts "
       + "      , trafodion.sales.orders orders "
       + "      , trafodion.persnl.employee emps "
       + "WHERE amounts.ordernum = orders.ordernum "
       + "   AND orders.salesrep = emps.empnum "
       + "ORDER BY orders.ordernum "
       ;

    java.sql.PreparedStatement ps2 = conn.prepareStatement( s ) ;
    ps2.setString( 1, onOrAfter ) ;
    orders[0] = ps2.executeQuery() ;

    // Open a result set for order detail rows
    s =   "SELECT d.*, p.partdesc "
    + "FROM trafodion.sales.odetail d, trafodion.sales.parts p, trafodion.sales.orders
O "
    + "WHERE d.partnum = p.partnum AND d.ordernum = o.ordernum "
    + "   AND o.order_date >= CAST(? AS DATE) "
    + "ORDER BY d.ordernum "
    ;

    java.sql.PreparedStatement ps3 = conn.prepareStatement( s ) ;
    ps3.setString( 1, onOrAfter ) ;
    detail[0] = ps3.executeQuery() ;

}
```

Creating the Procedure: ORDERSUMMARY

Before creating the procedure, create a library named SALES in the TRAFODION.SALES schema and select the Sales.jar file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 9. Create Procedure Settings: ORDERSUMMARY Procedure

Attribute	Definition
procedure-ref	trafodion.sales.ordersummary
sql-parameter-1	IN on_or_after_date VARCHAR(20)
sql-parameter-2	OUT num_orders LARGEINT
external name	EXTERNAL NAME 'Sales.orderSummary'
library	LIBRARY trafodion.sales.sales
external security	invoker[1] Choice depends on your security requirements, you can select definer instead. For more information, see Understand External Security.
language java	LANGUAGE JAVA
parameter style java	PARAMETER STYLE JAVA
sql access	READS SQL DATA
dynamic result sets	DYNAMIC RESULT SETS 2
transaction	TRANSACTION REQUIRED[1]
determinism	NOT DETERMINISTIC[1]
isolation	ISOLATE[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.sales.ordersummary( IN on_or_after_date VARCHAR(20)
                                             , OUT num_orders LARGEINT
                                             )
   EXTERNAL NAME 'Sales.orderSummary ( java.lang.String, long[] )'
   LIBRARY trafodion.sales.sales
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   DYNAMIC RESULT SETS 2
   READS SQL DATA
   ;
```

Calling the Procedure: ORDERSUMMARY

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the ORDERSUMMARY procedure in trafci:

```
SQL> CALL trafodion.sales.ordersummary( '2011-01-01', ? ) ;
```

The ORDERSUMMARY procedure returns this information about the orders on or after the specified date, 01-01-2011:

```
NUM_ORDERS
--------------------
                  13

ORDERNUM NUM_PARTS             AMOUNT                ORDER_DATE LAST_NAME
-------- -------------------- --------------------- ---------- --------------------
  100210                    4              19020.00 2011-04-10 HUGHES
  100250                    4              22625.00 2011-01-23 HUGHES
  101220                    4              45525.00 2011-07-21 SCHNABL
     ...                  ...                   ...        ... ...
--- 13 row(s) selected.

ORDERNUM PARTNUM UNIT_PRICE QTY_ORDERED PARTDESC
-------- ------- ---------- ----------- ------------------
  100210     244    3500.00           3 PC GOLD, 30 MB
  100210    5100     150.00          10 MONITOR BW, TYPE 1
  100210    2403     620.00           6 DAISY PRINTER,T2
  100210    2001    1100.00           3 GRAPHIC PRINTER,M1
  100250    6500      95.00          10 DISK CONTROLLER
  100250    6301     245.00          15 GRAPHIC CARD, HR
  100250     244    3500.00           4 PC GOLD, 30 MB
  100250    5103     400.00          10 MONITOR COLOR, M1
     ...     ...        ...         ... ...

--- 70 row(s) selected.

--- SQL operation complete.
```

10.2. Procedures in the PERSNL Schema

The Payroll class contains these SPJ methods, which are useful for managing personnel data:

- ADJUSTSALARY Procedure
- EMPLOYEEJOB Procedure
- PROJECTTEAM Procedure
- TOPSALESREPS Procedure

Those methods are registered as stored procedures in the PERSNL schema. Example 6 shows the code of the Payroll.java file.

Example 8: `Payroll.java` *- The Payroll Class*

```
import java.sql.* ;
import java.math.* ;

public class Payroll
{
   // The ADJUSTSALARY procedure accepts an employee number and a percentage
   // value and updates the employee's salary in the database based on that
   // percentage. This method also returns the updated salary to an output
   // parameter.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#adjustsalary-
procedure
   // for more documentation.
   public static void adjustSalary( BigDecimal empNum
                                  , double percent
                                  , BigDecimal[] newSalary
                                  ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement setSalary =
         conn.prepareStatement( "UPDATE trafodion.persnl.employee "
                              + "SET salary = salary * (1 + (? / 100)) "
                              + "WHERE empnum = ?"
                              ) ;

      PreparedStatement getSalary =
         conn.prepareStatement( "SELECT salary "
                              + "FROM trafodion.persnl.employee "
                              + "WHERE empnum = ?"
                              ) ;

      setSalary.setDouble( 1, percent ) ;
```

```
      setSalary.setBigDecimal( 2, empNum ) ;
      setSalary.executeUpdate() ;

      getSalary.setBigDecimal( 1, empNum ) ;
      ResultSet rs = getSalary.executeQuery() ;
      rs.next() ;

      newSalary[0] = rs.getBigDecimal( 1 ) ;

      rs.close() ;
      conn.close() ;

   }

   // The EMPLOYEEJOB procedure accepts an employee number and returns a job
   // code or null value to an output parameter.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#employeejob-
procedure
   // for more documentation.
   public static void employeeJob( int empNum
                                 , java.lang.Integer[] jobCode
                                 ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement getJobcode =
         conn.prepareStatement( "SELECT jobcode "
                              + "FROM trafodion.persnl.employee "
                              + "WHERE empnum = ?"
                              ) ;

      getJobcode.setInt( 1, empNum ) ;
      ResultSet rs = getJobcode.executeQuery() ;
      rs.next() ;

      int num = rs.getInt(1) ;
      if ( rs.wasNull() )
         jobCode[0] = null ;
      else
         jobCode[0] = new Integer(num) ;

      rs.close() ;
      conn.close() ;

   }

   // The PROJECTTEAM procedure accepts a project code and returns the
   // employee number, first name, last name, and location of the employees
   // assigned to that project.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#projectteam-
procedure
```

```
    // for more documentation.
    public static void projectTeam( int projectCode
                                  , ResultSet[] members
                                  ) throws SQLException
    {
       Connection conn =
          DriverManager.getConnection( "jdbc:default:connection" ) ;

       PreparedStatement getMembers =
          conn.prepareStatement( "SELECT E.empnum, E.first_name, E.last_name, D.location
"
                               + "FROM trafodion.persnl.employee E,
trafodion.persnl.dept D, trafodion.persnl.project P "
                               + "WHERE P.projcode = ? "
                               + "  AND P.empnum = E.empnum "
                               + "  AND E.deptnum = D.deptnum "
                               ) ;

        getMembers.setInt( 1, projectCode ) ;
        members[0] = getMembers.executeQuery() ;

    }

    // The TOPSALESREPS procedure accepts a number representing the fiscal
    // quarter (1, 2, 3, and 4, with each number representing a range of
    // months) and returns the employee number, first name, last name, and sale
    // figures of the top five sales representatives who had the highest sales
    // (unit_price * qty_ordered) that quarter.
    //
    // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#topsalesreps-
procedure
    // for more documentation.
    public static void topSalesReps( int whichQuarter
                                   , ResultSet[] topReps
                                   ) throws SQLException
    {
       if ( whichQuarter < 1 || whichQuarter > 4 )
       {
          throw new SQLException ( "Invalid value for quarter. "
                                 + "Retry the CALL statement "
                                 + "using a number from 1 to 4 "
                                 + "to represent the quarter."
                                 , "38001"
                                 ) ;
       }

       Connection conn =
          DriverManager.getConnection( "jdbc:default:connection" ) ;

       PreparedStatement getTopReps =
         conn.prepareStatement( "SELECT [first 5] e.empnum, e.first_name, "
                              + "e.last_name, totals.total "
                              + "FROM trafodion.persnl.employee e, "
                              + "   ( SELECT o.salesrep, "
```

```
                                + "     SUM( od.unit_price * od.qty_ordered ) as total "
                                + "     FROM trafodion.sales.orders o,
trafodion.sales.odetail od "
                                + "     WHERE o.ordernum = od.ordernum "
                                + "       AND QUARTER( o.order_date ) = ? "
                                + "     GROUP BY o.salesrep "
                                + "    ) totals "
                                + "WHERE e.empnum = totals.salesrep "
                                + "ORDER BY totals.total DESCENDING "
                                ) ;

      getTopReps.setInt( 1, whichQuarter ) ;
      topReps[0] = getTopReps.executeQuery() ;

   }
}
```

See the following sections for more information about each SPJ method.

10.2.1. ADJUSTSALARY Procedure

The ADJUSTSALARY procedure accepts an employee number and a percentage value and updates the employee's salary in the database based on that percentage. This method also returns the updated salary to an output parameter.

Java Method: adjustSalary()

Example 9: `adjustSalary()` *Method*

```
// The ADJUSTSALARY procedure accepts an employee number and a percentage
// value and updates the employee's salary in the database based on that
// percentage. This method also returns the updated salary to an output
// parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#adjustsalary-
procedure
// for more documentation.
public static void adjustSalary( BigDecimal empNum
                  , double percent
                  , BigDecimal[] newSalary
                  ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement setSalary =
      conn.prepareStatement( "UPDATE trafodion.persnl.employee "
               + "SET salary = salary * (1 + (? / 100)) "
               + "WHERE empnum = ?"
               ) ;

   PreparedStatement getSalary =
      conn.prepareStatement( "SELECT salary "
               + "FROM trafodion.persnl.employee "
               + "WHERE empnum = ?"
               ) ;

   setSalary.setDouble( 1, percent ) ;
   setSalary.setBigDecimal( 2, empNum ) ;
   setSalary.executeUpdate() ;

   getSalary.setBigDecimal( 1, empNum ) ;
   ResultSet rs = getSalary.executeQuery() ;
   rs.next() ;

   newSalary[0] = rs.getBigDecimal( 1 ) ;

   rs.close() ;
   conn.close() ;

}
```

Creating the Procedure: ADJUSTSALARY

Before creating the procedure, create a library named `PAYROLL` in the `TRAFODION.PERSNL` schema and select the `Payroll.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 10. Create Procedure Settings: ADJUSTSALARY Procedure

Attribute	Definition
procedure-ref	`trafodion.persnl.adjustsalary`
sql-parameter-1	`IN empnum NUMERIC(4)`
sql-parameter-2	`IN percent FLOAT`
sql-parameter-3	`OUT newsalary NUMERIC(8,2)`
external name	`EXTERNAL NAME 'Payroll.adjustSalary'`
library	`LIBRARY trafodion.persnl.payroll`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`MODIFIES SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.persnl.adjustsalary( IN empnum NUMERIC(4)
                                              , IN percent FLOAT
                                              , OUT newsalary NUMERIC(8,2)
                                              )
   EXTERNAL NAME 'Payroll.adjustSalary'
   LIBRARY trafodion.persnl.payroll
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   MODIFIES SQL DATA
   ;
```

Calling the Procedure: ADJUSTSALARY

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the ADJUSTSALARY procedure in trafci:

```
SQL> CALL trafodion.persnl.adjustsalary(29, 2.5, ?) ;
```

The ADJUSTSALARY procedure updates the salary of employee number 29 by 2.5 percent and returns this output in trafci:

```
NEWSALARY
----------
 139400.00

--- SQL operation complete.
```

The salary of employee number 29 was originally $136,000.00 and became $139,400.00 after the invocation of ADJUSTSALARY.

10.2.2. EMPLOYEEJOB Procedure

The EMPLOYEEJOB procedure accepts an employee number and returns a job code or null value to an output parameter.

Java Method: employeeJob()

Example 10: `employeeJob()` *Method*

```
// The EMPLOYEEJOB procedure accepts an employee number and returns a job
// code or null value to an output parameter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#employeejob-
procedure
// for more documentation.
public static void employeeJob( int empNum
                  , java.lang.Integer[] jobCode
                  ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getJobcode =
      conn.prepareStatement( "SELECT jobcode "
               + "FROM trafodion.persnl.employee "
               + "WHERE empnum = ?"
               ) ;

   getJobcode.setInt( 1, empNum ) ;
   ResultSet rs = getJobcode.executeQuery() ;
   rs.next() ;

   int num = rs.getInt(1) ;
   if ( rs.wasNull() )
      jobCode[0] = null ;
   else
      jobCode[0] = new Integer(num) ;

   rs.close() ;
   conn.close() ;

}
```

Creating the Procedure: EMPLOYEEJOB

Before creating the procedure, create a library named `PAYROLL` in the `TRAFODION.PERSNL` schema and select the `Payroll.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use these values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 11. Create Procedure Settings: EMPLOYEEJOB Procedure

Attribute	Definition
procedure-ref	`trafodion.persnl.employeejob`
sql-parameter-1	`IN empnum`
sql-parameter-2	`OUT jobcode INT`
external name	`EXTERNAL NAME 'Payroll.employeeJob ( int, java.lang.Integer[] )'`
library	`LIBRARY trafodion.persnl.payroll`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.persnl.employeejob( IN empnum INT
                                             , OUT jobcode INT
                                             )
   EXTERNAL NAME 'Payroll.employeeJob ( int, java.lang.Integer[] )'
   LIBRARY trafodion.persnl.payroll
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
   ;
```

Calling the Procedure: EMPLOYEEJOB

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the EMPLOYEEJOB procedure in trafci:

```
SQL> CALL trafodion.persnl.employeejob(337, ?) ;
```

The EMPLOYEEJOB procedure accepts the employee number 337 and returns this output in trafci:

```
JOBCODE
-----------
        900

--- SQL operation complete.
```

The job code for employee number 337 is 900.

10.2.3. PROJECTTEAM Procedure

The PROJECTTEAM procedure accepts a project code and returns the employee number, first name, last name, and location of the employees assigned to that project.

Java Method: projectTeam()

Example 11: `projectTeam()` *Method*

```
// The PROJECTTEAM procedure accepts a project code and returns the
// employee number, first name, last name, and location of the employees
// assigned to that project.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#projectteam-
procedure
// for more documentation.
public static void projectTeam( int projectCode
                  , ResultSet[] members
                  ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getMembers =
      conn.prepareStatement( "SELECT E.empnum, E.first_name, E.last_name, D.location "
               + "FROM trafodion.persnl.employee E, trafodion.persnl.dept D,
trafodion.persnl.project P "
               + "WHERE P.projcode = ? "
               + "  AND P.empnum = E.empnum "
               + "  AND E.deptnum = D.deptnum "
               ) ;

    getMembers.setInt( 1, projectCode ) ;
    members[0] = getMembers.executeQuery() ;

}
```

Creating the Procedure: PROJECTTEAM

Before creating the procedure, create a library named `PAYROLL` in the `TRAFODION.PERSNL` schema and select the `Payroll.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 12. Create Procedure Settings: PROJECTTEAM Procedure

Attribute	Definition
procedure-ref	`trafodion.persnl.projectteam`
sql-parameter-1	`IN projectcode INTEGER`
external name	`EXTERNAL NAME 'Payroll.projectTeam'`
library	`LIBRARY trafodion.persnl.payroll`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 1`
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.persnl.projectteam( IN projectcode INTEGER
                                              )
   EXTERNAL NAME 'Payroll.projectTeam'
   LIBRARY trafodion.persnl.payroll
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   DYNAMIC RESULT SETS 1
   READS SQL DATA
   ;
```

Calling the Procedure: PROJECTTEAM

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the PROJECTTEAM procedure in trafci:

```
SQL> CALL trafodion.persnl.projectteam( 5000 ) ;
```

The PROJECTTEAM procedure returns this information about the employees assigned to project number 5000:

```
EMPNUM FIRST_NAME      LAST_NAME            LOCATION
------ --------------- -------------------- --------------------
    65 RACHEL          MCKAY                NEW YORK
   203 KATHRYN         HALL                 NEW YORK
... ... ...

--- 6 row(s) selected.

--- SQL operation complete.
```

10.2.4. TOPSALESREPS Procedure

The TOPSALESREPS procedure accepts a number representing the fiscal quarter (1, 2, 3, and 4, with each number representing a range of months) and returns the employee number, first name, last name, and sale figures of the top five sales representatives who had the highest sales (unit_price * qty_ordered) that quarter.

Java Method: topSalesReps()

Example 12: `topSalesReps()` *Method*

```
// The TOPSALESREPS procedure accepts a number representing the fiscal
// quarter (1, 2, 3, and 4, with each number representing a range of
// months) and returns the employee number, first name, last name, and sale
// figures of the top five sales representatives who had the highest sales
// (unit_price * qty_ordered) that quarter.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#topsalesreps-
procedure
// for more documentation.
public static void topSalesReps( int whichQuarter
                  , ResultSet[] topReps
                  ) throws SQLException
{
   if ( whichQuarter < 1 || whichQuarter > 4 )
   {
      throw new SQLException ( "Invalid value for quarter. "
                + "Retry the CALL statement "
                + "using a number from 1 to 4 "
                + "to represent the quarter."
                , "38001"
                ) ;
   }

   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getTopReps =
     conn.prepareStatement( "SELECT [first 5] e.empnum, e.first_name, "
              + "e.last_name, totals.total "
              + "FROM trafodion.persnl.employee e, "
              + "   ( SELECT o.salesrep, "
              + "     SUM( od.unit_price * od.qty_ordered ) as total "
              + "     FROM trafodion.sales.orders o, trafodion.sales.odetail od "
              + "     WHERE o.ordernum = od.ordernum "
              + "       AND QUARTER( o.order_date ) = ? "
              + "     GROUP BY o.salesrep "
              + "    ) totals "
              + "WHERE e.empnum = totals.salesrep "
              + "ORDER BY totals.total DESCENDING "
              ) ;

   getTopReps.setInt( 1, whichQuarter ) ;
   topReps[0] = getTopReps.executeQuery() ;
}
```

Creating the Procedure: TOPSALESREPS

Before creating the procedure, create a library named `PAYROLL` in the `TRAFODION.PERSNL` schema and select the `Payroll.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 13. Create Procedure Settings: TOPSALESREPS Procedure

Attribute	Definition
procedure-ref	`trafodion.persnl.topsalesreps`
sql-parameter-1	`IN whichquarter INTEGER`
external name	`EXTERNAL NAME 'Payroll.topSalesReps'`
library	`LIBRARY trafodion.persnl.payroll`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 1`
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.persnl.topsalesreps( IN whichquarter INTEGER
                                                )
   EXTERNAL NAME 'Payroll.topSalesReps'
   LIBRARY trafodion.persnl.payroll
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   DYNAMIC RESULT SETS 1
   READS SQL DATA
   ;
```

Calling the Procedure: TOPSALESREPS

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the TOPSALESREPS procedure in trafci:

```
SQL> CALL trafodion.persnl.topsalesreps( 1 ) ;
```

The TOPSALESREPS procedure returns this information about the top five sales representatives during the first fiscal quarter:

```
EMPNUM FIRST_NAME      LAST_NAME            TOTAL
------ --------------- -------------------- --------------------
   227 XAVIER          SEDLEMEYER                     172460.00
   231 HERB            ALBERT                          67025.00
   222 MARTIN          SCHAEFFER                       52000.00
   226 HEIDI           WEIGL                           28985.00
   220 JOHN            HUGHES                          22625.00

--- 5 row(s) selected.

--- SQL operation complete.
```

10.3. Procedures in the INVENT Schema

The Inventory class contains these SPJ methods, which are useful for tracking parts and suppliers:

- SUPPLIERINFO Procedure
- SUPPLYQUANTITIES Procedure
- PARTLOCS Procedure

Those methods are registered as stored procedures in the INVENT schema. Example 13 shows the code of the Inventory.java file.

Example 13: `Inventory.java` *- The Inventory Class*

```
import java.sql.* ;
import java.math.* ;

public class Inventory
{
   // The SUPPLIERINFO procedure accepts a supplier number and returns the
   // supplier's name, street, city, state, and post code to separate output
   // parameters.
   //
   // See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#supplierinfo-
procedure
   // for more documentation.
   public static void supplierInfo( BigDecimal suppNum
                                  , String[] suppName
                  , String[] streetAddr
                  , String[] cityName
                  , String[] stateName
                  , String[] postCode
                  ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement getSupplier =
         conn.prepareStatement( "SELECT suppname, street, city, "
                              + "       state, postcode "
                     + "FROM trafodion.invent.supplier "
                     + "WHERE suppnum = ?"
                     ) ;

      getSupplier.setBigDecimal( 1, suppNum ) ;
      ResultSet rs = getSupplier.executeQuery() ;
      rs.next() ;

      suppName[0]   = rs.getString( 1 ) ;
      streetAddr[0] = rs.getString( 2 ) ;
```

```
      cityName[0]   = rs.getString( 3 ) ;
      stateName[0]  = rs.getString( 4 ) ;
      postCode[0]   = rs.getString( 5 ) ;

      rs.close() ;
      conn.close() ;

   }

   // The SUPPLYQUANTITIES procedure returns the average, minimum, and maximum
   // quantities of available parts in inventory to separate output
   // parameters.
   //
   // See
http://trafodion.incubator.apache.org/docs/spj_guide/index.html#supplyquantities-
procedure
   // for more documentation.
   public static void supplyQuantities( int[] avgQty
                                      , int[] minQty
                     , int[] maxQty
                     ) throws SQLException
   {
      Connection conn =
         DriverManager.getConnection( "jdbc:default:connection" ) ;

      PreparedStatement getQty =
         conn.prepareStatement( "SELECT AVG(qty_on_hand), "
                           + "       MIN(qty_on_hand), "
                  + "       MAX(qty_on_hand) "
                  + "FROM trafodion.invent.partloc"
                  ) ;

      ResultSet rs = getQty.executeQuery() ;
      rs.next() ;

      avgQty[0] = rs.getInt( 1 ) ;
      minQty[0] = rs.getInt( 2 ) ;
      maxQty[0] = rs.getInt( 3 ) ;

      rs.close() ;
      conn.close() ;

   }

   // The PARTLOCATIONS procedure accepts a part number and quantity and returns a
   // set of location codes that have the exact quantity and a set of location
   // codes that have more than that quantity.
   //
   // See
http://trafodion.incubator.apache.org/docs/spj_guide/index.html#partlocations-procedure
   // for more documentation.
   public static void partLocations( int partNum
                                   , int quantity
                  , ResultSet exactly[]
```

```
                    , ResultSet moreThan[]
                    ) throws SQLException

    {
       Connection conn =
          DriverManager.getConnection( "jdbc:default:connection" ) ;

       PreparedStatement getLocationsExact =
          conn.prepareStatement( "SELECT L.loc_code, L.partnum, L.qty_on_hand "
                              + "FROM trafodion.invent.partloc L "
                    + "WHERE L.partnum = ? "
                    + "  AND L.qty_on_hand = ? "
                    + " ORDER BY L.partnum "
                    ) ;

       getLocationsExact.setInt( 1, partNum ) ;
       getLocationsExact.setInt( 2, quantity) ;

       PreparedStatement getLocationsMoreThan =
          conn.prepareStatement( "SELECT L.loc_code, L.partnum, L.qty_on_hand "
                              + "FROM trafodion.invent.partloc L "
                    + "WHERE L.partnum = ? "
                    + "  AND L.qty_on_hand > ? "
                    + "ORDER BY L.partnum "
                    ) ;

       getLocationsMoreThan.setInt( 1, partNum ) ;
       getLocationsMoreThan.setInt( 2, quantity) ;

       exactly[0]  = getLocationsExact.executeQuery() ;
       moreThan[0] = getLocationsMoreThan.executeQuery() ;

    }
}
```

See the following sections for more information about each SPJ method.

10.3.1. SUPPLIERINFO Procedure

The SUPPLIERINFO procedure accepts a supplier number and returns the supplier's name, street, city, state, and post code to separate output parameters.

Java Method: supplierInfo()

Example 14: `supplierInfo()` *Method*

```
// The SUPPLIERINFO procedure accepts a supplier number and returns the
// supplier's name, street, city, state, and post code to separate output
// parameters.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#supplierinfo-
procedure
// for more documentation.
public static void supplierInfo( BigDecimal suppNum
                  , String[] suppName
                  , String[] streetAddr
                  , String[] cityName
                  , String[] stateName
                  , String[] postCode
                  ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getSupplier =
      conn.prepareStatement( "SELECT suppname, street, city, "
               + "       state, postcode "
               + "FROM trafodion.invent.supplier "
               + "WHERE suppnum = ?"
               ) ;

   getSupplier.setBigDecimal( 1, suppNum ) ;
   ResultSet rs = getSupplier.executeQuery() ;
   rs.next() ;

   suppName[0]   = rs.getString( 1 ) ;
   streetAddr[0] = rs.getString( 2 ) ;
   cityName[0]   = rs.getString( 3 ) ;
   stateName[0]  = rs.getString( 4 ) ;
   postCode[0]   = rs.getString( 5 ) ;

   rs.close() ;
   conn.close() ;

}
```

Creating the Procedure: SUPPLIERINFO

Before creating the procedure, create a library named `INVENTORY` in the `TRAFODION.INVENT` schema and select the `Inventory.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 14. Create Procedure Settings: SUPPLIERINFO Procedure

Attribute	Definition
procedure-ref	`trafodion.invent.supplierinfo`
sql-parameter-1	`IN suppnum NUMERIC(4)`
sql-parameter-2	`OUT suppname CHARACTER(18)`
sql-parameter-3	`OUT address CHARACTER(22)`
sql-parameter-4	`OUT city CHARACTER(14)`
sql-parameter-5	`OUT state CHARACTER(12)`
sql-parameter-6	`OUT zipcode CHARACTER(10)`
external name	`EXTERNAL NAME 'Inventory.supplierInfo (java.math.BigDecimal, java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[])''
library	`LIBRARY trafodion.invent.inventory`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.invent.supplierinfo( IN  suppnum NUMERIC(4)
                                              , OUT suppname CHARACTER(18)
                                              , OUT address CHARACTER(22)
                                              , OUT city CHARACTER(14)
                                              , OUT state CHARACTER(12)
                                              , OUT zipcode CHARACTER(10)
                                              )
   EXTERNAL NAME 'Inventory.supplierInfo ( java.math.BigDecimal
                                         , java.lang.String[]
                                         , java.lang.String[]
                                         , java.lang.String[]
                                         , java.lang.String[]
                                         , java.lang.String[]
                                         )'
   LIBRARY trafodion.invent.inventory
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
   ;
```

Calling the Procedure: SUPPLIERINFO

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the SUPPLIERINFO procedure in trafci:

```
SQL> CALL trafodion.invent.supplierinfo( 25,?,?,?,?,? ) ;
```

The SUPPLIERINFO procedure accepts the supplier number 25 and returns this output in trafci:

```
SUPPNAME           ADDRESS                CITY           STATE        ZIPCODE
------------------ ---------------------- -------------- ------------ ---------
Schroeder's Ltd    212 Strasse Blvd West  Hamburg        Rhode Island 22222

--- SQL operation complete.
```

Supplier number 25 is Schroeder’s Ltd. and is located in Hamburg, Rhode Island.

10.3.2. SUPPLYQUANTITIES Procedure

The SUPPLYQUANTITIES procedure returns the average, minimum, and maximum quantities of available parts in inventory to separate output parameters.

Java Method: supplyQuantities()

Example 15: `supplyQuantities()` *Method*

```
// The SUPPLYQUANTITIES procedure returns the average, minimum, and maximum
// quantities of available parts in inventory to separate output
// parameters.
//
// See
http://trafodion.incubator.apache.org/docs/spj_guide/index.html#supplyquantities-
procedure
// for more documentation.
public static void supplyQuantities( int[] avgQty
                   , int[] minQty
                   , int[] maxQty
                   ) throws SQLException
{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getQty =
      conn.prepareStatement( "SELECT AVG(qty_on_hand), "
               + "       MIN(qty_on_hand), "
               + "       MAX(qty_on_hand) "
               + "FROM trafodion.invent.partloc"
               ) ;

   ResultSet rs = getQty.executeQuery() ;
   rs.next() ;

   avgQty[0] = rs.getInt( 1 ) ;
   minQty[0] = rs.getInt( 2 ) ;
   maxQty[0] = rs.getInt( 3 ) ;

   rs.close() ;
   conn.close() ;

}
```

Creating the Procedure: SUPPLYQUANTITIES

Before creating the procedure, create a library named `INVENTORY` in the `TRAFODION.INVENT` schema and select the `Inventory.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 15. Create Procedure Settings: SUPPLYQUANTITIES Procedure

Attribute	Definition
procedure-ref	`trafodion.invent.supplyquantities`
sql-parameter-1	`OUT avrg INT`
sql-parameter-2	`OUT minm INT`
sql-parameter-3	`OUT maxm INT`
external name	`EXTERNAL NAME 'Inventory.supplyQuantities'`
library	`LIBRARY trafodion.invent.inventory`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 0`[1]
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.invent.supplyquantities( OUT avrg INT
                                                  , OUT minm INT
                                                  , OUT maxm INT
                                                  )
   EXTERNAL NAME 'Inventory.supplyQuantities'
   LIBRARY trafodion.invent.inventory
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
   ;
```

Calling the Procedure: SUPPLYQUANTITIES

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the SUPPLYQUANTITIES procedure in trafci:

```
SQL> CALL trafodion.invent.supplyquantities( ?,?,? ) ;
```

The SUPPLYQUANTITIES procedure returns this output in trafci:

```
AVRG         MINM         MAXM
----------- ----------- -----------
        167           0        1132

--- SQL operation complete.
```

The average number of items in inventory is 167, the minimum number is 0, and the maximum number is 1132.

10.3.3. PARTLOCATIONS Procedure

The PARTLOCATIONS procedure accepts a part number and quantity and returns a set of location codes that have the exact quantity and a set of location codes that have more than that quantity.

Java Method: partLocations()

Example 16: `partLocations()` *Method*

```
// The PARTLOCATIONS procedure accepts a part number and quantity and returns a
// set of location codes that have the exact quantity and a set of location
// codes that have more than that quantity.
//
// See http://trafodion.incubator.apache.org/docs/spj_guide/index.html#partlocations-
procedure
// for more documentation.
public static void partLocations( int partNum
                , int quantity
                , ResultSet exactly[]
                , ResultSet moreThan[]
                ) throws SQLException

{
   Connection conn =
      DriverManager.getConnection( "jdbc:default:connection" ) ;

   PreparedStatement getLocationsExact =
      conn.prepareStatement( "SELECT L.loc_code, L.partnum, L.qty_on_hand "
               + "FROM trafodion.invent.partloc L "
               + "WHERE L.partnum = ? "
               + "  AND L.qty_on_hand = ? "
               + " ORDER BY L.partnum "
               ) ;

   getLocationsExact.setInt( 1, partNum ) ;
   getLocationsExact.setInt( 2, quantity) ;

   PreparedStatement getLocationsMoreThan =
      conn.prepareStatement( "SELECT L.loc_code, L.partnum, L.qty_on_hand "
               + "FROM trafodion.invent.partloc L "
               + "WHERE L.partnum = ? "
               + "  AND L.qty_on_hand > ? "
               + "ORDER BY L.partnum "
               ) ;

   getLocationsMoreThan.setInt( 1, partNum ) ;
   getLocationsMoreThan.setInt( 2, quantity) ;

   exactly[0]  = getLocationsExact.executeQuery() ;
   moreThan[0] = getLocationsMoreThan.executeQuery() ;

}
```

Creating the Procedure: PARTLOCS

Before creating the procedure, create a library named `INVENTORY` in the `TRAFODION.INVENT` schema and select the `Inventory.jar` file to upload to Trafodion for that library. For more information, see Create a Library.

After creating the library, use the values listed in the table below to define and execute the CREATE PROCEDURE command. For more information, see Create a Procedure.

Table 16. Create Procedure Settings: PARTLOCS Procedure

Attribute	Definition
procedure-ref	`trafodion.invent.partlocs`
sql-parameter-1	`IN partnum INT`
sql-parameter-2	`IN qty INT`
external name	`EXTERNAL NAME 'Inventory.partLocations'`
library	`LIBRARY trafodion.invent.inventory`
external security	`invoker`[1] Choice depends on your security requirements, you can select `definer` instead. For more information, see Understand External Security.
language java	`LANGUAGE JAVA`
parameter style java	`PARAMETER STYLE JAVA`
sql access	`READS SQL DATA`
dynamic result sets	`DYNAMIC RESULT SETS 2`
transaction	`TRANSACTION REQUIRED`[1]
determinism	`NOT DETERMINISTIC`[1]
isolation	`ISOLATE`[1]

[1] Definition represents default value. Not included in the CREATE PROCEDURE example.

```
CREATE PROCEDURE trafodion.invent.partlocs( IN partnum INT
                                          , IN qty INT
                                          )
   EXTERNAL NAME 'Inventory.partLocations'
   LIBRARY trafodion.invent.inventory
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   DYNAMIC RESULT SETS 2
   READS SQL DATA
   ;
```

Calling the Procedure: PARTLOCS

Make sure that users who will be calling the stored procedure have the appropriate execute privileges. For more information, see Grant Privileges.

To invoke the PARTLOCS procedure in trafci:

```
SQL> CALL trafodion.invent.partlocs( 212, 18 ) ;
```

The PARTLOCS procedure accepts the part number 212 and returns a set of locations that have 18 of those parts and a set of locations that have more than 18 of those parts:

```
LOC_CODE PARTNUM QTY_ON_HAND
-------- ------- -----------
G87          212          20

--- 1 row(s) selected.

LOC_CODE PARTNUM QTY_ON_HAND
-------- ------- -----------
G87          212          20

--- 1 row(s) selected.

--- SQL operation complete.
```

Chapter 11. B Sample Database

This appendix presents the Sample Database schemas and tables on which the SPJs in this manual operate:

- PERSNL Schema
  - JOB Table
  - EMPLOYEE Table
  - DEPT Table
  - PROJECT Table
- SALES Schema
  - CUSTOMER Table
  - ORDERS Table
  - ODETAIL Table
  - PARTS Table
- INVENT Schema
  - SUPPLIER Table
  - PARTSUPP Table
  - PARTLOC Table

Click on the link for the schema or table name to download the definition file.

11.1. PERSNL Schema

The PERSNL schema stores employee data.

```
CREATE SCHEMA trafodion.persnl ;
SET SCHEMA trafodion.persnl ;
```

11.1.1. JOB Table

The JOB table maps job codes to job descriptions.

```
CREATE TABLE trafodion.persnl.job
( jobcode NUMERIC (4)  UNSIGNED NO DEFAULT  NOT NULL
, jobdesc VARCHAR (18)          DEFAULT ' ' NOT NULL
, PRIMARY KEY ( jobcode )
) ;

INSERT INTO trafodion.persnl.job VALUES
  ( 100, 'MANAGER' )
, ( 200, 'PRODUCTION SUPV' )
, ( 250, 'ASSEMBLER' )
, ( 300, 'SALESREP' )
, ( 400, 'SYSTEM ANALYST' )
, ( 420, 'ENGINEER' )
, ( 450, 'PROGRAMMER' )
, ( 500, 'ACCOUNTANT' )
, ( 600, 'ADMINISTRATOR' )
, ( 900, 'SECRETARY' )
;

UPDATE STATISTICS FOR TABLE trafodion.persnl.job ON EVERY COLUMN ;
```

11.1.2. EMPLOYEE Table

The `EMPLOYEE` table maps records employee information.

```
CREATE TABLE trafodion.persnl.employee
( empnum     NUMERIC (4)    UNSIGNED NO DEFAULT  NOT NULL
, first_name CHARACTER (15)          DEFAULT ' ' NOT NULL
, last_name  CHARACTER (20)          DEFAULT ' ' NOT NULL
, deptnum    NUMERIC (4)    UNSIGNED NO DEFAULT  NOT NULL
, jobcode    NUMERIC (4)    UNSIGNED DEFAULT     NULL
, salary     NUMERIC (8, 2) UNSIGNED DEFAULT     NULL
, PRIMARY KEY ( empnum )
) ;

ALTER TABLE trafodion.persnl.employee
   ADD CONSTRAINT empnum_constrnt CHECK ( empnum BETWEEN 0001 AND 9999 )
   ;

CREATE INDEX xempname ON employee
( last_name
, first_name
) ;

CREATE INDEX xempdept ON employee
( deptnum
) ;

CREATE VIEW trafodion.persnl.emplist AS
SELECT
```

```
  empnum
, first_name
, last_name
, deptnum
, jobcode
FROM employee
;

INSERT INTO trafodion.persnl.employee VALUES
  (   1, 'ROGER',     'GREEN',       9000, 100,  175500.00 )
, (  23, 'JERRY',     'HOWARD',      1000, 100,  137000.10 )
, (  29, 'JANE',      'RAYMOND',     3000, 100,  136000.00 )
, (  32, 'THOMAS',    'RUDLOFF',     2000, 100,  138000.40 )
, (  39, 'KLAUS ',    'SAFFERT',     3200, 100,   75000.00 )
, (  43, 'PAUL',      'WINTER',      3100, 100,   90000.00 )
, (  65, 'RACHEL',    'MCKAY',       4000, 100,  118000.00 )
, (  72, 'GLENN',     'THOMAS',      3300, 100,   80000.00 )
, (  75, 'TIM',       'WALKER',      3000, 300,   32000.00 )
, (  87, 'ERIC',      'BROWN',       4000, 400,   89000.00 )
, (  89, 'PETER',     'SMITH',       3300, 300,   37000.40 )
, (  93, 'DONALD',    'TAYLOR',      3100, 300,   33000.00 )
, ( 104, 'DAVID',     'STRAND',      4000, 400,   69000.00 )
, ( 109, 'STEVE',     'COOK',        4000, 400,   68000.00 )
, ( 111, 'SHERRIE',   'WONG',        3500, 100,   70000.00 )
, ( 178, 'JOHN',      'CHOU',        3500, 900,   28000.00 )
, ( 180, 'MANFRED',   'CONRAD',      4000, 450,   32000.00 )
, ( 201, 'JIM',       'HERMAN',      3000, 300,   19000.00 )
, ( 202, 'LARRY',     'CLARK',       1000, 500,   25000.75 )
, ( 203, 'KATHRYN',   'HALL',        4000, 400,   96000.00 )
, ( 205, 'GINNY',     'FOSTER',      3300, 900,   30000.00 )
, ( 206, 'DAVE',      'FISHER',      3200, 900,   25000.00 )
, ( 207, 'MARK',      'FOLEY',       4000, 420,   33000.00 )
, ( 208, 'SUE',       'CRAMER',      1000, 900,   19000.00 )
, ( 209, 'SUSAN',     'CHAPMAN',     1500, 900,   17000.00 )
, ( 210, 'RICHARD',   'BARTON',      1000, 500,   29000.00 )
, ( 211, 'JIMMY',     'SCHNEIDER',   1500, 600,   26000.00 )
, ( 212, 'JONATHAN',  'MITCHELL',    1500, 600,   32000.00 )
, ( 213, 'ROBERT',    'WHITE',       1500, 100,   90000.00 )
, ( 214, 'JULIA',     'KELLY',       1000, 500,   50000.00 )
, ( 215, 'WALTER',    'LANCASTER',   4000, 450,   33000.50 )
, ( 216, 'JOHN',      'JONES',       4000, 450,   40000.00 )
, ( 217, 'MARLENE',   'BONNY',       4000, 900,   24000.90 )
, ( 218, 'GEORGE',    'FRENCHMAN',   4000, 420,   36000.00 )
, ( 219, 'DAVID',     'TERRY',       2000, 250,   27000.12 )
, ( 220, 'JOHN',      'HUGHES',      3200, 300,   33000.10 )
, ( 221, 'OTTO',      'SCHNABL',     3200, 300,   33000.00 )
, ( 222, 'MARTIN',    'SCHAEFFER',   3200, 300,   31000.00 )
, ( 223, 'HERBERT',   'KARAJAN',     3200, 300,   29000.00 )
, ( 224, 'MARIA',     'JOSEF',       4000, 420,   18000.10 )
, ( 225, 'KARL',      'HELMSTED',    4000, 450,   32000.00 )
, ( 226, 'HEIDI',     'WEIGL',       3200, 300,   22000.00 )
, ( 227, 'XAVIER',    'SEDLEMEYER',  3300, 300,   30000.00 )
, ( 228, 'PETE',      'WELLINGTON',  3100, 300,   32000.20 )
, ( 229, 'GEORGE',    'STRICKER',    3100, 300,   32222.00 )
```

```
, ( 230, 'ROCKY',     'LEWIS',       2000, 200,    24000.00 )
, ( 231, 'HERB',      'ALBERT',      3300, 300,    33000.00 )
, ( 232, 'THOMAS',    'SPINNER',     4000, 450,    45000.00 )
, ( 233, 'TED',       'MCDONALD',    2000, 250,    29000.00 )
, ( 234, 'MARY',      'MILLER',      2500, 100,    56000.00 )
, ( 235, 'MIRIAM',    'KING',        2500, 900,    18000.00 )
, ( 321, 'BILL',      'WINN',        2000, 900,    32000.00 )
, ( 337, 'DINAH',     'CLARK',       9000, 900,    37000.00 )
, ( 343, 'ALAN',      'TERRY',       3000, 900,    39500.00 )
, ( 557, 'BEN',       'HENDERSON',   4000, 400,    65000.00 )
, ( 568, 'JESSICA',   'CRINER',      3500, 300,    39500.00 )
, ( 990, 'THOMAS',    'STIBBS',      3500, NULL,       NULL )
, ( 991, 'WAYNE',     'O''NEIL',     3500, NULL,       NULL )
, ( 992, 'BARRY',     'KINNEY',      3500, NULL,       NULL )
, ( 993, 'PAUL',      'BUSKETT',     3100, NULL,       NULL )
, ( 994, 'EMMY',      'BUSKETT',     3100, NULL,       NULL )
, ( 995, 'WALT',      'FARLEY',      3100, NULL,       NULL )
;

UPDATE STATISTICS FOR TABLE trafodion.persnl.employee ON EVERY COLUMN ;
```

11.1.3. DEPT Table

The DEPT table maps records department information.

```
CREATE TABLE trafodion.persnl.dept
( deptnum  NUMERIC (4) UNSIGNED NO DEFAULT  NOT NULL
, deptname CHARACTER (12)       NO DEFAULT  NOT NULL
, manager  NUMERIC (4) UNSIGNED NO DEFAULT  NOT NULL
, rptdept  NUMERIC (4) UNSIGNED DEFAULT 0   NOT NULL
, location VARCHAR (18)         DEFAULT ' ' NOT NULL
, PRIMARY KEY ( deptnum )
) ;

CREATE INDEX xdeptmgr ON dept
( manager
) ;

CREATE INDEX xdeptrpt ON dept
( rptdept
) ;

ALTER TABLE trafodion.persnl.dept
   ADD CONSTRAINT mgrnum_constrnt
   CHECK (manager BETWEEN 0000 AND 9999)
   ;

ALTER TABLE trafodion.persnl.dept
   ADD CONSTRAINT deptnum_constrnt
      CHECK ( deptnum IN
              ( 1000
```

```
               , 1500
               , 2000
               , 2500
               , 3000
               , 3100
               , 3200
               , 3300
               , 3500
               , 4000
               , 4100
               , 9000
               )
            )
            ;
3
CREATE VIEW trafodion.persnl.mgrlist
( first_name
, last_name
, department
)
AS SELECT
  first_name
, last_name
, deptname
FROM dept, employee
WHERE dept.manager = employee.empnum
;

INSERT INTO trafodion.persnl.dept VALUES
  ( 1000, 'FINANCE',        23, 9000, 'CHICAGO'      )
, ( 1500, 'PERSONNEL',     213, 1000, 'CHICAGO'      )
, ( 2000, 'INVENTORY',      32, 9000, 'LOS ANGELES'  )
, ( 2500, 'SHIPPING',      234, 2000, 'PHOENIX'      )
, ( 3000, 'MARKETING',      29, 9000, 'NEW YORK'     )
, ( 3100, 'CANADA SALES',   43, 3000, 'TORONTO'      )
, ( 3200, 'GERMNY SALES',   39, 3000, 'FRANKFURT'    )
, ( 3300, 'ENGLND SALES',   72, 3000, 'LONDON'       )
, ( 3500, 'ASIA SALES',    111, 3000, 'HONG KONG'    )
, ( 4000, 'RESEARCH',       65, 9000, 'NEW YORK'     )
, ( 4100, 'PLANNING',       87, 4000, 'NEW YORK'     )
, ( 9000, 'xxCORPORATE',     1, 9000, 'CHICAGO'      )
;

UPDATE STATISTICS FOR TABLE trafodion.persnl.dept ON EVERY COLUMN ;
```

11.1.4. PROJECT Table

The PROJECT table maps records information about projects.

```
CREATE TABLE trafodion.persnl.project
( projcode       NUMERIC (4)  UNSIGNED NO DEFAULT
```

```
NOT NULL
, empnum         NUMERIC (4)  UNSIGNED NO DEFAULT
NOT NULL
, projdesc       VARCHAR (18)          DEFAULT ' '
NOT NULL
, start_date     DATE                  DEFAULT DATE '2011-07-01'
NOT NULL
, ship_timestamp TIMESTAMP             DEFAULT TIMESTAMP '2011-08-01:12:00:00.000000'
NOT NULL
, est_complete   INTERVAL DAY          DEFAULT INTERVAL '30' DAY
NOT NULL
, PRIMARY KEY ( projcode, empnum )
) ;

INSERT INTO trafodion.persnl.project
VALUES
  ( 1000, 213, 'SALT LAKE CITY',    DATE '2011-04-10', TIMESTAMP '2011-04-
21:08:15:00.00',   INTERVAL '15' DAY )
, ( 1000, 211, 'SALT LAKE CITY',    DATE '2011-04-10', TIMESTAMP '2011-04-
21:08:15:00.00',   INTERVAL '15' DAY )
, ( 1000,  23, 'SALT LAKE CITY',    DATE '2011-04-10', TIMESTAMP '2011-04-
21:08:15:00.00',   INTERVAL '15' DAY )
, ( 1000,   1, 'SALT LAKE CITY',    DATE '2011-04-10', TIMESTAMP '2011-04-
21:08:15:00.00',   INTERVAL '15' DAY )
, ( 2000, 227, 'ROSS PRODUCTS',     DATE '2011-06-10', TIMESTAMP '2011-07-
21:08:30:00.0000', INTERVAL '30' DAY )
, ( 2000, 109, 'ROSS PRODUCTS',     DATE '2011-06-10', TIMESTAMP '2011-07-
21:08:30:00.0000', INTERVAL '30' DAY )
, ( 2000, 215, 'ROSS PRODUCTS',     DATE '2011-06-10', TIMESTAMP '2011-07-
21:08:30:00.0000', INTERVAL '30' DAY )
, ( 2000,  65, 'ROSS PRODUCTS',     DATE '2011-06-10', TIMESTAMP '2011-07-
21:08:30:00.0000', INTERVAL '30' DAY )
, ( 2500,  65, 'MONTANA TOOLS',     DATE '2011-10-10', TIMESTAMP '2011-12-
21:09:00:00.0000', INTERVAL '60' DAY )
, ( 2500, 207, 'MONTANA TOOLS',     DATE '2011-10-10', TIMESTAMP '2011-12-
21:09:00:00.0000', INTERVAL '60' DAY )
, ( 2500, 232, 'MONTANA TOOLS',     DATE '2011-10-10', TIMESTAMP '2011-12-
21:09:00:00.0000', INTERVAL '60' DAY )
, ( 2500, 180, 'MONTANA TOOLS',     DATE '2011-10-10', TIMESTAMP '2011-12-
21:09:00:00.0000', INTERVAL '60' DAY )
, ( 2500,  93, 'MONTANA TOOLS',     DATE '2011-10-10', TIMESTAMP '2011-12-
21:09:00:00.0000', INTERVAL '60' DAY )
, ( 3000,  65, 'AHAUS TOOL/SUPPLY', DATE '2011-08-21', TIMESTAMP '2011-10-
21:08:10:00.0000', INTERVAL '60' DAY )
, ( 3000, 221, 'AHAUS TOOL/SUPPLY', DATE '2011-08-21', TIMESTAMP '2011-10-
21:08:10:00.0000', INTERVAL '60' DAY )
, ( 3000, 226, 'AHAUS TOOL/SUPPLY', DATE '2011-08-21', TIMESTAMP '2011-10-
21:08:10:00.0000', INTERVAL '60' DAY )
, ( 3000, 224, 'AHAUS TOOL/SUPPLY', DATE '2011-08-21', TIMESTAMP '2011-10-
21:08:10:00.0000', INTERVAL '60' DAY )
, ( 3000, 225, 'AHAUS TOOL/SUPPLY', DATE '2011-08-21', TIMESTAMP '2011-10-
21:08:10:00.0000', INTERVAL '60' DAY )
, ( 4000,  75, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
```

```
, ( 4000,  29, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
, ( 4000, 231, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
, ( 4000, 228, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
, ( 4000, 223, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
, ( 4000, 568, 'THE WORKS',         DATE '2011-09-21', TIMESTAMP '2011-10-
21:10:15:00.0000', INTERVAL '30' DAY )
, ( 5000,  65, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
, ( 5000, 568, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
, ( 5000, 557, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
, ( 5000, 216, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
, ( 5000, 203, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
, ( 5000, 218, 'ASIA PROJECT',      DATE '2011-09-28', TIMESTAMP '2011-10-
28:09:25:01.1111', INTERVAL '30' DAY )
;

UPDATE STATISTICS FOR TABLE trafodion.persnl.project ON EVERY COLUMN ;
```

11.2. SALES Schema

The SALES schema stores customer and sales data.

```
CREATE SCHEMA trafodion.sales ;
SET SCHEMA trafodion.sales ;
```

11.2.1. CUSTOMER Table

The CUSTOMER table maps records information about customers.

```
CREATE TABLE trafodion.sales.customer
( custnum  NUMERIC (4) UNSIGNED NO DEFAULT   NOT NULL
, custname CHARACTER (18)       NO DEFAULT   NOT NULL
, street   CHARACTER (22)       NO DEFAULT   NOT NULL
, city     CHARACTER (14)       NO DEFAULT   NOT NULL
, state    CHARACTER (12)       DEFAULT ' '  NOT NULL
, postcode CHARACTER (10)       NO DEFAULT   NOT NULL
, credit   CHARACTER (2)        DEFAULT 'C1' NOT NULL
, PRIMARY KEY ( custnum )
) ;

INSERT INTO trafodion.sales.customer VALUES
  (   21, 'CENTRAL UNIVERSITY', 'UNIVERSITY WAY',        'PHILADELPHIA',  '
PENNSYLVANIA', '19104',    'A1' )
, (  123, 'BROWN MEDICAL CO',   '100 CALIFORNIA STREET', 'SAN FRANCISCO', 'CALIFORNIA',
'94944',    'C2' )
, (  143, 'STEVENS SUPPLY',     '2020 HARRIS STREET',    'DENVER',        'COLORADO',
'80734',    'A2' )
, (  324, 'PREMIER INSURANCE',  '3300 WARBASH',          'LUBBOCK',       'TEXAS',
'76308',    'A1' )
, (  543, 'FRESNO STATE BANK',  '2300 BROWN BLVD',       'FRESNO',        'CALIFORNIA',
'93921',    'B3' )
, (  926, 'METALL-AG.',         '12 WAGNERRING',         'FRANKFURT',     'WEST
GERMANY', '34',       'D4' )
, ( 1234, 'DATASPEED',          '300 SAN GABRIEL WAY',   'NEW YORK',      'NEW YORK',
'10014',    'C1' )
, ( 3210, 'BESTFOOD MARKETS',   '3333 PHELPS STREET',    'LINCOLN',       'NEBRASKA',
'68134',    'A4' )
, ( 3333, 'NATIONAL UTILITIES', '6500 TRANS-CANADIENNE', 'QUEBEC',        'CANADA',
'H4T 1X4',  'A1' )
, ( 5635, 'ROYAL CHEMICALS',    '45 NEW BROAD STREET',   'LONDON',        'ENGLAND',
'EC2M 1NH', 'B2' )
, ( 7654, 'MOTOR DISTRIBUTING', '2345 FIRST STREET',     'CHICAGO',       'ILLINOIS',
'60610',    'E4' )
, ( 7777, 'SLEEPWELL HOTELS',   '9000 PETERS AVENUE',    'DALLAS',        'TEXAS',
'75244',    'B1' )
, ( 9000, 'BUNKNOUGHT INN',     '4738 RALPH STREET',     'BAYONNE',       'NEW JERSEY',
'09520',    'C1' )
, ( 9010, 'HOTEL OREGON',       '333 PORTLAND AVE.',     'MEDFORD',       'OREGON',
'97444',    'C2' )
, ( 9033, 'ART SUPPLIES, INC.', '22 SWEET ST.',          'PITTSBURGH',    'PENNA.',
'08333',    'C3' )
;

UPDATE STATISTICS FOR TABLE trafodion.sales.customer ON EVERY COLUMN;
```

11.2.2. ORDERS Table

The ORDERS table maps records information about sales orders.

```
CREATE TABLE trafodion.sales.orders
( ordernum   NUMERIC (6) UNSIGNED NO DEFAULT                NOT NULL
, order_date DATE                 DEFAULT DATE '2011-07-01' NOT NULL
, deliv_date DATE                 DEFAULT DATE '2011-08-01' NOT NULL
, salesrep   NUMERIC (4) UNSIGNED DEFAULT 0                 NOT NULL
, custnum    NUMERIC (4) UNSIGNED NO DEFAULT                NOT NULL
, PRIMARY KEY ( ordernum )
) ;

ALTER TABLE trafodion.sales.orders
   ADD CONSTRAINT trafodion.sales.date_constrnt CHECK ( deliv_date >= order_date )
   ;

CREATE INDEX xordrep ON orders
( salesrep
) ;

CREATE INDEX xordcus ON orders
( custnum
) ;

CREATE VIEW trafodion.sales.ordrep AS SELECT
  empnum
, last_name
, ordernum
, o.custnum
FROM
  trafodion.persnl.employee e
, trafodion.sales.orders o
, trafodion.sales.customer c
WHERE e.empnum = o.salesrep
  AND o.custnum = C.custnum
;

CREATE INDEX xcustnam ON customer
(
custname
) ;

CREATE VIEW trafodion.sales.custlist AS SELECT
  custnum
, custname
, street
, city
, state
, postcode
FROM trafodion.sales.customer
;

INSERT INTO trafodion.sales.orders VALUES
  ( 100210, DATE '2011-04-10', DATE '2011-04-10', 220, 1234 )
, ( 100250, DATE '2011-01-23', DATE '2011-06-15', 220, 7777 )
, ( 101220, DATE '2011-07-21', DATE '2011-12-15', 221, 5635 )
, ( 200300, DATE '2011-02-06', DATE '2011-07-01', 222,  926 )
```

```
, ( 200320, DATE '2011-02-17', DATE '2011-07-20', 223,   21 )
, ( 200490, DATE '2011-03-19', DATE '2011-11-01', 226,  123 )
, ( 300350, DATE '2011-03-03', DATE '2011-08-10', 231,  543 )
, ( 300380, DATE '2011-03-19', DATE '2011-08-20', 226,  123 )
, ( 400410, DATE '2011-03-27', DATE '2011-09-01', 227, 7654 )
, ( 500450, DATE '2011-04-20', DATE '2011-09-15', 220,  324 )
, ( 600480, DATE '2011-05-12', DATE '2011-10-10', 226, 3333 )
, ( 700510, DATE '2011-06-01', DATE '2011-10-20', 229,  143 )
, ( 800660, DATE '2011-10-09', DATE '2011-11-01', 568, 3210 )
;

UPDATE STATISTICS FOR TABLE trafodion.sales.orders ON EVERY COLUMN ;
```

11.2.3. ODETAIL Table

The `ODETAIL` table maps records detailed information about sales orders.

```
CREATE TABLE trafodion.sales.odetail
( ordernum    NUMERIC (6) UNSIGNED NO DEFAULT NOT NULL
, partnum     NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, unit_price  NUMERIC (8, 2)       NO DEFAULT NOT NULL
, qty_ordered NUMERIC (5) UNSIGNED NO DEFAULT NOT NULL
, PRIMARY KEY ( ordernum, partnum )
) ;

INSERT INTO trafodion.sales.odetail VALUES
  ( 100210,  244,  3500.00,   3 )
, ( 100210, 2001,  1100.00,   3 )
, ( 100210, 2403,   620.00,   6 )
, ( 100210, 5100,   150.00,  10 )
, ( 100250,  244,  3500.00,   4 )
, ( 100250, 5103,   400.00,  10 )
, ( 100250, 6301,   245.00,  15 )
, ( 100250, 6500,    95.00,  10 )
, ( 101220,  255,  3900.00,  10 )
, ( 101220, 5103,   400.00,   3 )
, ( 101220, 7102,   275.00,   7 )
, ( 101220, 7301,   425.00,   8 )
, ( 200300,  244,  3500.00,   8 )
, ( 200300, 2001,  1000.00,  10 )
, ( 200300, 2002,  1400.00,  10 )
, ( 200320, 5504,   165.00,   5 )
, ( 200320, 6201,   195.00,  16 )
, ( 200320, 6301,   245.00,   6 )
, ( 200320, 6400,   540.00,   7 )
, ( 200490, 3210,   715.00,   1 )
, ( 200490, 5505,   350.00,   1 )
, ( 300350,  244,  2800.00,  20 )
, ( 300350, 5100,   150.00,   5 )
, ( 300350, 5110,   525.00,  12 )
, ( 300350, 6301,   245.00,   5 )
```

```
, ( 300350, 6400,   550.00,   5 )
, ( 300380,  244,  3000.00,   6 )
, ( 300380, 2402,   320.00,  12 )
, ( 300380, 2405,   760.00,   8 )
, ( 400410,  212,  2450.00,  12 )
, ( 400410,  255,  3800.00,  12 )
, ( 400410, 2001,  1000.00,  36 )
, ( 400410, 6301,   240.00,  48 )
, ( 400410, 6400,   500.00,  70 )
, ( 400410, 7301,   415.00,  36 )
, ( 500450,  212,  2500.00,   8 )
, ( 500450,  255,  3900.00,  12 )
, ( 500450, 2001,  1100.00,  16 )
, ( 500450, 2002,  1500.00,  16 )
, ( 500450, 2402,   330.00,  48 )
, ( 600480, 2001,  1000.00,  60 )
, ( 600480, 2002,  1450.00,  20 )
, ( 600480, 2003,  1900.00,  40 )
, ( 600480, 3103,  4000.00,  40 )
, ( 600480, 3205,   625.00,  20 )
, ( 600480, 5100,   135.00,  60 )
, ( 600480, 5103,   390.00,  20 )
, ( 600480, 7301,   425.00,  40 )
, ( 700410, 2003,  1900.00,  65 )
, ( 700410, 2403,   650.00,  10 )
, ( 700510,  255,  4000.00,   4 )
, ( 700510, 6500,    95.00,   8 )
, ( 700510, 7102,   275.00,   5 )
, ( 800660,  244,  3000.00,   6 )
, ( 800660, 2001,  1000.00,  30 )
, ( 800660, 2403,   600.00,  48 )
, ( 800660, 2405,   795.00,  10 )
, ( 800660, 3201,   525.00,   6 )
, ( 800660, 3205,   600.00,  18 )
, ( 800660, 3210,   715.00,   6 )
, ( 800660, 4102,    26.00, 130 )
, ( 800660, 5100,   150.00,  12 )
, ( 800660, 5101,   200.00,   6 )
, ( 800660, 5110,   490.00,  48 )
, ( 800660, 5504,   165.00,  18 )
, ( 800660, 6201,   195.00,   6 )
, ( 800660, 6301,   235.00,  24 )
, ( 800660, 6400,   525.00,  30 )
, ( 800660, 6401,   700.00,  36 )
, ( 800660, 6500,    95.00,  22 )
, ( 800660, 7102,   275.00,   6 )
, ( 800660, 7301,   425.00,  12 )
;

UPDATE STATISTICS FOR TABLE trafodion.sales.odetail ON EVERY COLUMN ;
```

11.2.4. PARTS Table

The PARTS table maps records information about parts.

```
CREATE TABLE trafodion.sales.parts
( partnum       NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, partdesc      CHARACTER (18)       NO DEFAULT NOT NULL
, price         NUMERIC (8, 2)       NO DEFAULT NOT NULL
, qty_available NUMERIC (5)          DEFAULT 0  NOT NULL
, PRIMARY KEY ( partnum )
) ;

CREATE INDEX xpartdes ON parts
( partdesc
) ;

INSERT INTO trafodion.sales.parts VALUES
  (  186, '186 MegaByte Disk',  186186.86, 186  )
, (  212, 'PC SILVER, 20 MB',     2500.00, 3525 )
, (  244, 'PC GOLD, 30 MB',       3000.00, 4426 )
, (  255, 'PC DIAMOND, 60 MB',    4000.00, 3321 )
, ( 2001, 'GRAPHIC PRINTER,M1',   1100.00, 2100 )
, ( 2002, 'GRAPHIC PRINTER,M2',   1500.00, 3220 )
, ( 2003, 'GRAPHIC PRINTER,M3',   2000.00, 2200 )
, ( 2402, 'DAISY PRINTER,T1',      350.00, 4425 )
, ( 2403, 'DAISY PRINTER,T2',      650.00, 3312 )
, ( 2405, 'DAISY PRINTER, T3',     795.00, 2712 )
, ( 3103, 'LASER PRINTER, X1',    4200.00, 3300 )
, ( 3201, 'HARD DISK 20 MB',       525.00, 4436 )
, ( 3205, 'HARD DISK 30 MB',       625.00, 2209 )
, ( 3210, 'HARD DISK 40 MB',       715.00, 3314 )
, ( 4102, 'DISKETTE HD, BOX',       28.00, 6540 )
, ( 5100, 'MONITOR BW, TYPE 1',    150.00, 3237 )
, ( 5101, 'MONITOR BW, TYPE 2',    200.00, 2400 )
, ( 5103, 'MONITOR COLOR, M1',     400.00, 3328 )
, ( 5110, 'MONITOR COLOR, M2',     525.00, 3236 )
, ( 5504, 'MEMORY CARD, 512KB',    165.00, 2630 )
, ( 5505, 'MEMORY CARD, 1 MB',     315.00, 3830 )
, ( 6201, 'GRAPHIC CARD, LR',      195.00, 2306 )
, ( 6301, 'GRAPHIC CARD, HR',      245.00, 2331 )
, ( 6400, 'STREAMING TAPE,M20',    550.00, 1268 )
, ( 6401, 'STREAMING TAPE,M60',    725.00, 1308 )
, ( 6500, 'DISK CONTROLLER',        95.00, 2532 )
, ( 6603, 'PRINTER CONTROLLER',     45.00,  430 )
, ( 7102, 'SMART MODEM, 1200',     275.00, 2200 )
, ( 7301, 'SMART MODEM, 2400',     425.00, 2332 )
;

UPDATE STATISTICS FOR TABLE trafodion.sales.parts ON EVERY COLUMN ;
```

11.2.5. INVENT Schema

The `INVENT` schema stores inventory data.

```
CREATE SCHEMA trafodion.invent ;
SET SCHEMA trafodion.invent ;
```

11.2.6. SUPPLIER Table

The `SUPPLIER` table maps records information about suppliers.

```
CREATE TABLE trafodion.invent.supplier
( suppnum  NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, suppname CHARACTER (18)       NO DEFAULT NOT NULL
, street   CHARACTER (22)       NO DEFAULT NOT NULL
, city     CHARACTER (14)       NO DEFAULT NOT NULL
, state    CHARACTER (12)       NO DEFAULT NOT NULL
, postcode CHARACTER (10)       NO DEFAULT NOT NULL
, PRIMARY KEY ( suppnum )
) ;

CREATE INDEX xsuppnam ON supplier
( suppname
) ;

INSERT INTO trafodion.invent.supplier VALUES
  (   1, 'NEW COMPUTERS INC',    '1800 KING ST.',         'SAN FRANCISCO', 'CALIFORNIA
',   '94112' )
, (   2, 'DATA TERMINAL INC',    '2000 BAKER STREET',     'LAS VEGAS',     'NEVADA',
'66134' )
, (   3, 'HIGH DENSITY INC',     '7600 EMERSON',          'NEW YORK',      'NEW YORK',
'10230' )
, (   6, 'MAGNETICS INC',        '1000 INDUSTRY DRIVE',   'LEXINGTON',     'MASS',
'02159' )
, (   8, 'ATTRACTIVE CORP',      '7777 FOUNTAIN WAY',     'CHICAGO',       'ILLINOIS',
'60610' )
, (  10, 'LEVERAGE INC',         '6000 LINCOLN LANE',     'DENVER',        'COLORADO',
'80712' )
, (  15, 'DATADRIVE CORP',       '100 MAC ARTHUR',        'DALLAS',        'TEXAS',
'75244' )
, (  20, 'Macadam''S PC''s',     '106 River Road',        'New Orleans',   'Louisiana',
'67890' )
, (  25, 'Schroeder''s Ltd',     '212 Strasse Blvd West', 'Hamburg',       'Rhode
Island', '22222' )
, (  30, 'O''Donnell''s Drives', '729 West Palm Beach ',  'San Antonio',   'Texas',
'78344' )
, (  35, 'Mac''Murphys PC''s',   '93323 Alemeda',         'Menlo Park',    'California
',   '94025' )
, (  36, 'MAC''MURPHYS PCB''s',  '93323 Alemeda Suite B', 'Menlo Park',    'California
',   '94025' )
, (  90, 'laser jets inc',       '284 blue ridge way',    'levittown',     'penna.',
'09520' )
, (  92, 'watercolors',          '84 north grand avenue', 'menlo park',    'california
',   '94025' )
, (  95, 'application do''ers',  '2846 yellowwood drive', 'wayland',       'mass',
'02158' )
, (  99, 'terminals, inc.',      '2 longfellow way',      'heightstown',   'nj',
'08520' )
, ( 186, '186 Disk Makers',      '186 Dis Way',           'Dat Way',       'Wisconsin',
'00186' )
;

UPDATE STATISTICS FOR TABLE trafodion.invent.supplier ON EVERY COLUMN ;
```

11.2.7. PARTSUPP Table

The PARTSUPP table maps parts to suppliers.

```
CREATE TABLE trafodion.invent.partsupp
( partnum      NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, suppnum      NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, partcost     NUMERIC (8, 2)       NO DEFAULT NOT NULL
, qty_received NUMERIC (5) UNSIGNED DEFAULT 0  NOT NULL
, PRIMARY KEY ( partnum, suppnum )
) ;

CREATE INDEX XSUPORD ON partsupp
( suppnum
) ;

CREATE VIEW trafodion.invent.view207
( partnumber
, partdescrpt
, suppnumber
, supplrname
, partprice
, qtyreceived
)
AS SELECT
  x.partnum
, partdesc
, x.suppnum
, suppname
, partcost
, qty_received
FROM
  trafodion.invent.partsupp x
, trafodion.sales.parts p
, trafodion.invent.supplier s
WHERE x.partnum = p.partnum
  AND x.suppnum = s.suppnum
;

CREATE VIEW trafodion.invent.view207n
( partnumber
, partdescrpt
, suppnumber
, supplrname
, partprice
, qtyreceived
)
AS SELECT
  x.partnum
, p.partdesc
, s.suppnum
, s.suppname
```

```
, x.partcost
, x.qty_received
FROM trafodion.invent.supplier s
LEFT JOIN trafodion.invent.partsupp x ON s.suppnum = x.suppnum
LEFT JOIN trafodion.sales.parts p     ON x.partnum = p.partnum
;

CREATE VIEW trafodion.invent.viewcust
( custnumber
, cusname
, ordernum
)
AS SELECT
  c.custnum
, c.custname
, o.ordernum
FROM trafodion.sales.customer c
LEFT JOIN trafodion.sales.orders o ON c.custnum = o.custnum
;

CREATE VIEW trafodion.invent.viewcs AS SELECT
  custname
FROM trafodion.sales.customer
UNION SELECT
  suppname
FROM trafodion.invent.supplier ;

INSERT INTO trafodion.invent.partsupp VALUES
  (  212,  1, 2000.00,  20 )
, (  212,  3, 1900.00,  35 )
, (  244,  1, 2400.00,  50 )
, (  244,  2, 2200.00,  66 )
, (  255,  1, 3300.00,  35 )
, (  255,  3, 3000.00,  46 )
, ( 2001,  1,  700.00, 100 )
, ( 2001,  2,  750.00,  55 )
, ( 2002,  1, 1000.00, 120 )
, ( 2002,  6, 1100.00,  20 )
, ( 2003,  1, 1300.00, 100 )
, ( 2003,  2, 1400.00,  50 )
, ( 2003, 10, 1450.00,  50 )
, ( 2402,  1,  200.00,  35 )
, ( 2403,  1,  300.00, 200 )
, ( 2405,  1,  500.00,  40 )
, ( 2405,  6,  450.00,  50 )
, ( 3103,  1, 3200.00, 200 )
, ( 3103, 15, 3300.00, 100 )
, ( 3201,  1,  380.00,  36 )
, ( 3205,  1,  425.00, 150 )
, ( 3210,  6,  470.00,  10 )
, ( 3210, 15,  450.00,  25 )
, ( 4102,  6,   20.00, 115 )
, ( 4102,  8,   19.00, 140 )
, ( 4102, 15,   21.00,  30 )
```

```
, ( 5100,  6,  100.00,  50 )
, ( 5100,  8,  105.00,  40 )
, ( 5100, 15,   95.00,  60 )
, ( 5101,  8,  135.00,  33 )
, ( 5101, 15,  125.00,  43 )
, ( 5103,  8,  265.00,  20 )
, ( 5103, 15,  250.00,  58 )
, ( 5110,  1,  335.00, 100 )
, ( 5110,  2,  350.00,  36 )
, ( 5504,  2,   85.00,  10 )
, ( 5504,  6,   75.00,  10 )
, ( 5504, 15,   78.00,  10 )
, ( 5505, 15,  200.00, 100 )
, ( 6201,  1,  100.00, 110 )
, ( 6301,  1,  150.00, 230 )
, ( 6400,  1,  390.00,  50 )
, ( 6401,  2,  500.00,  20 )
, ( 6401,  3,  480.00,  38 )
, ( 6500,  2,   60.00, 140 )
, ( 6500,  3,   65.00,  32 )
, ( 6603,  2,   25.00, 150 )
, ( 7102, 10,  165.00, 100 )
, ( 7301,  1,  300.00,  32 )
;

UPDATE STATISTICS FOR TABLE trafodion.invent.partsupp ON EVERY COLUMN ;
```

11.2.8. PARTLOC Table

The PARTLOC table records number of parts to on hand.

```
CREATE TABLE trafodion.invent.partloc
( loc_code    CHARACTER (3)        NO DEFAULT NOT NULL
, partnum     NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL
, qty_on_hand NUMERIC (7)          DEFAULT 0  NOT NULL
, PRIMARY KEY ( loc_code, partnum )
) ;

INSERT INTO trafodion.invent.partloc VALUES
  ( 'A10', 2001,  800 )
, ( 'A21',  255,   21 )
, ( 'A34', 6201,    0 )
, ( 'A35', 6301,    0 )
, ( 'A36', 6400,   34 )
, ( 'A66', 6603,  300 )
, ( 'A67', 6401,  454 )
, ( 'A78',  244,   43 )
, ( 'A78', 5505,  100 )
, ( 'A87',  212,   18 )
, ( 'A88', 2403,  735 )
, ( 'A88', 5504,   30 )
```

```
, ( 'A94', 3205,  200 )
, ( 'A98', 5110,  510 )
, ( 'G11', 2002,   20 )
, ( 'G34', 6201,  106 )
, ( 'G35', 6301,  331 )
, ( 'G36', 6400, 1034 )
, ( 'G43', 5100,   77 )
, ( 'G45', 4102,   69 )
, ( 'G65', 3201,   36 )
, ( 'G68', 6500, 1132 )
, ( 'G76', 2405,   42 )
, ( 'G76', 7301,   32 )
, ( 'G78', 5505,    0 )
, ( 'G87',  212,   20 )
, ( 'G87', 3103,    0 )
, ( 'G87', 3210,   44 )
, ( 'G87', 2402,    0 )
, ( 'G88', 2403,   32 )
, ( 'G88', 5504,    0 )
, ( 'G89', 5101,   86 )
, ( 'G94', 3205,   59 )
, ( 'G98', 5103,   28 )
, ( 'G98', 5110,  136 )
, ( 'P10', 2001,    0 )
, ( 'P12', 2002,  200 )
, ( 'P12', 2003,    0 )
, ( 'P15', 2003,  200 )
, ( 'P66', 6603,   40 )
, ( 'P67', 6401,   54 )
, ( 'P68', 6500,    0 )
, ( 'P76', 7102,  200 )
, ( 'P78',  244,   23 )
, ( 'P87', 3103,  300 )
;

UPDATE STATISTICS FOR TABLE trafodion.invent.partloc ON EVERY COLUMN ;
```