

June 2009

Welcome to the June 2009 edition of IBM InfoSphere Information Server Developer's Notebook. This month we answer the question;

How can I use IBM Information Server with my database server specific and often proprietary (objects, programming, and constraint checking)?

Excellent question! As this question was received, specific mention was given to the following database resident objects; Sequences, Stored Procedures, and responses from database resident data integrity constraints.

Not every SQL database object is ANSI SQL standard, and as a result this is one of the first times this document is delving into the area of database observed differences. Still in most cases, IBM Information Server is going to abstract you from the differences of say, Oracle to SQL/Server, or Oracle version over version. You achieve full functionality without much of the observed pain.

Software versions

All of these solutions were *developed and tested* on (IBM) InfoSphere Information Server (IIS) version 8.1.1, using the Microsoft Windows XP/SP3 platform to support IIS client programs, and a RedHat Linux Advanced Server 5 (RHAS 5) 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server side components.

IBM InfoSphere Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM InfoSphere Information Server product contains the following major components;

WebSphere Business Glossary Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, InfoSphere Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM InfoSphere Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

29.1 Terms and core concepts

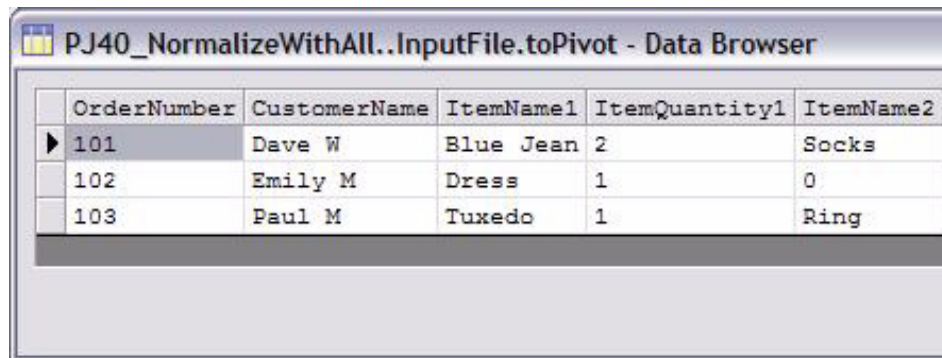
Database server Primary Keys

The first versions of relational database servers performed the 4 basic data manipulation operations we find in nearly every data environment, and little else. (Those operations include; Insert new data, Update existing data, Delete existing data, and Select or read existing data.) Conceptually, relational databases are built on many principles, including one which states data records are allowed to exist in a table in random sequence. So that you may later recall, or identify data records to Update, Delete or Select, each record must have a unique identifier, or Primary Key.

Generally the Primary Key to each data record is stated at time of record creation (Insert), and is never altered (never Updated). Creating or generating this Primary Key can be done inside or outside of the database server.

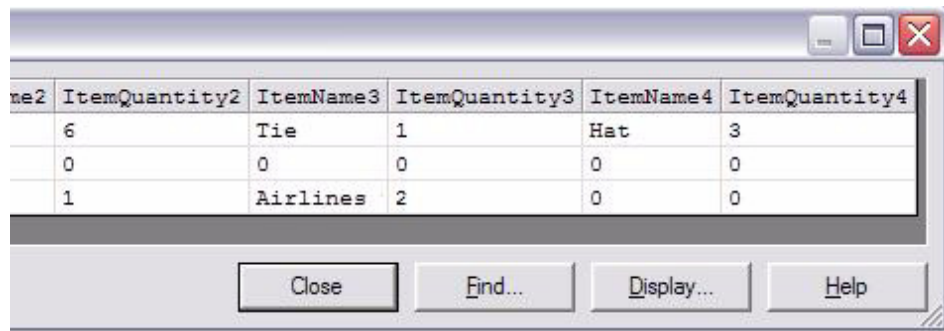
In the August 2007 edition of this document, we detailed a design pattern for an IBM Information Server (IIS) DataStage component Job, where, for example, a data file is received from an external, non-relational database source, and needs to be loaded into a relational database source. We re-introduce and continue using that example here.

Images Figure 29-1, and Figure 29-2 display a sample table, sample data, from the August 2007 edition of this document. (The number of columns was so wide, we split this display into two images.)



OrderNumber	CustomerName	ItemName1	ItemQuantity1	ItemName2
101	Dave W	Blue Jean	2	Socks
102	Emily M	Dress	1	0
103	Paul M	Tuxedo	1	Ring

Figure 29-1 Sample data, page 1 of 2.



ItemQuantity2	ItemName3	ItemQuantity3	ItemName4	ItemQuantity4
6	Tie	1	Hat	3
0		0	0	0
1	Airlines	2	0	0

Figure 29-2 Sample data, page 2 of 2.

The table displayed in Figure 29-1 and Figure 29-2 is from a non-relational database source and is *denormalized*. As in the convention in a relational database source, data from figures Figure 29-1 and Figure 29-2 will be split into two, *normalized* data tables; from this example, source data will be split into an Order Header table, and an Order Line Item table.

So that we may correctly associate the numerous Order Line Item data records with the proper (single) Order Header data record, we need to know the Primary Key of a given Order Header data record as it is created.

As stated above, primary keys may be assigned inside or outside of the database server. If our design pattern was to allow the database server to assign primary key values to each Order Header data record as it is created, consider the following;

- Each new record Insert of an Order Header data record, would have to be followed by a return message from the database server, wherein we are then informed of the newly assigned Primary Key value.

We would then have to place the assigned Primary Key value into the correct and numerous Order Line Item data records, and Insert those.

This means we would have to hold each Order Line Item data record, not Insert it, until its associated Order Header data record has been inserted.

- Each communication event with the database server has a cost in both time and CPU. Each of these events causes a small, sub-second delay in processing as a request is sent to the database server, and a response is waited for.

One design strategy that software products like IBM Information Server (IIS) make use of to optimize performance, is to send hundreds if not thousands of requests to the database server in one call. There is not, however, a database server mechanism to send (group) thousands of

responses back to any software program that initiated said call. Relational database servers can only send one response notification back at a time.

Note: Design patterns here would be to have a Reject File on the database server side, accepting data records that fail to (Insert). Or pre-validate data records before submitting them to the database server, so that we know they will (Insert) correctly.

There is also a database server ability to send, for example, 1000 new data records for Insert, and upon failure, return the row number of the offending data record and the failure cause.

Imagine we send 1000 new data records, and data record 50 will fail to Insert. Based on the capabilities of the database server, you could commit to Inserting data records 1 through 49, and discard data record 50, or fix that record and resubmit it, and also resubmit records 51 and beyond.

The math in this scenario quickly works to your disfavor-

If, for example, we needed to Insert 1,000,000 new data records, and could send them in groups of 1,000, but, every 50th row were to fail on Insert, we would actually incur the overhead to have to submit 20,000,000 new data records to the database server, as we had to resubmit the data records after data record 50 that got blocked.

In the case of needing to know the database server assigned Primary Value, there is no concept of a Reject File; the data records Inserted correctly go where they needed to go.

The most performant option, from those offered above, is to assign Primary Key values outside of the database server. We can poll the database server for the next highest Primary Key value to be assigned (Primary Keys are most often incremented sequentially), and then manage that list, manage that progression ourselves.

The IBM Information Server (IIS) object to deliver this functionality, is the DataStage component Stage entitled, Surrogate Key Generator (SKG).

Figure 29-3 displays the sample DataStage Job from the August 2007 edition of this document. The DataStage Job in Figure 29-3 does not handle Primary Keys; if the Primary Key was not delivered in the Input File at the left most edge of that Job, and that Job needs to associate Master and Detail file data records, then that Job fails to meet its design goal.

(As it stands, the Job in Figure 29-3 was only to demonstrate normalizing data, and did not mean to address the topic of Primary Key generation.)

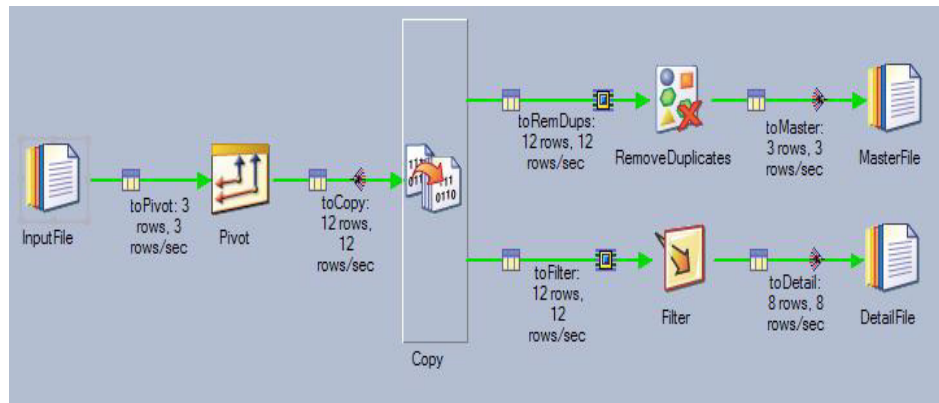


Figure 29-3 Sample Job to Normalize data, no Primary Key capability.

A more complete version of the Job displayed in Figure 29-3, one that does handle Primary Key generation, is displayed in Figure 29-4 and Figure 29-5. (This sample Job was too wide for one image.) A code review follows.

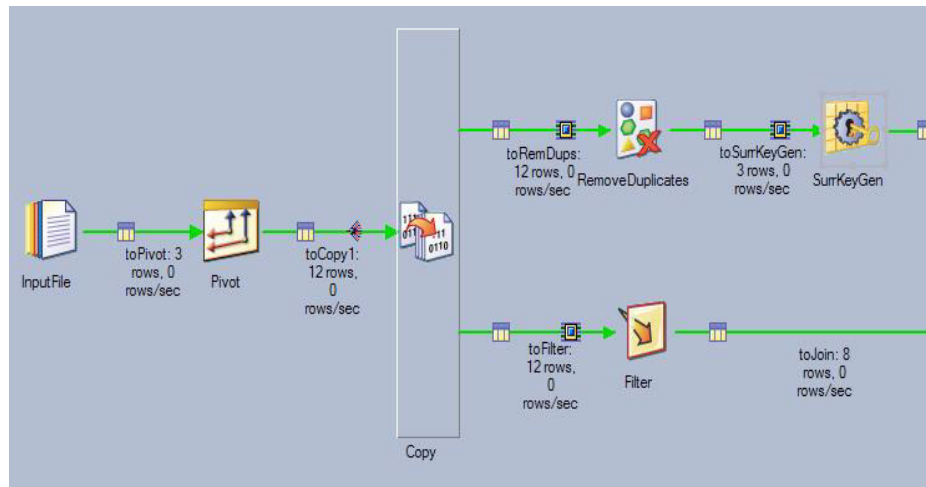


Figure 29-4 Sample Job to Normalize data, with Primary Key creation, page 1 of 2.

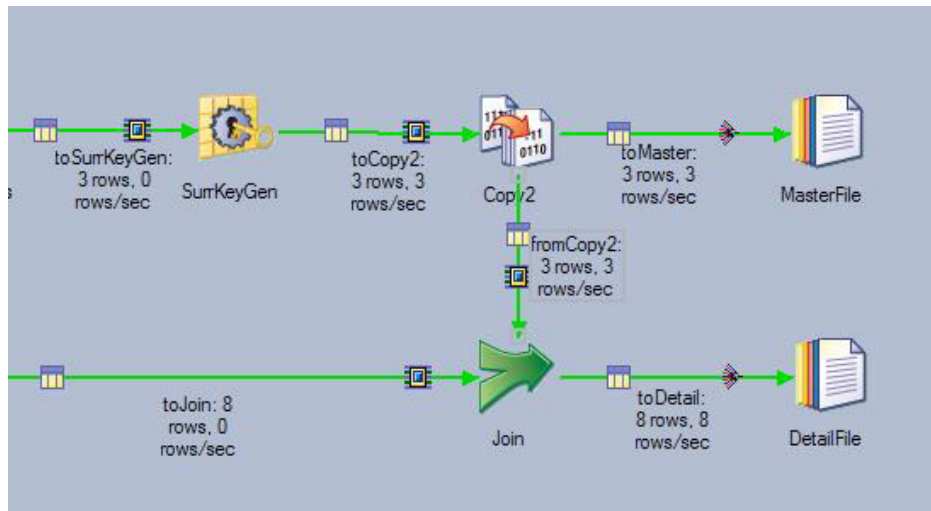


Figure 29-5 Sample Job to Normalize data, with Primary Key creation, page 2 of 2.

Related to Figure 29-4 and Figure 29-5, the following is offered;

- The IBM Information Server (IIS) DataStage component Job begins by reading data equal to that as displayed in Figure 29-1 and Figure 29-2, our sample data set discussed throughout this section.
- The Pivot Stage performs our basic data Normalization. That is how 3 input data records become 12.
- The Copy Stage allows us to send data to two or more output streams; per our visual layout and design the top stream is for Order Header data records, and the lower stream is for Order Line Item data records.
- Order Header data records-
 - As the Order Line Items repeat Order Header data, we need to remove duplicate records from the Order Header stream; hence, the Remove Duplicates Stage.
 - Then the Surrogate Key Generator Stage assigns a Primary Key value to individual Order Header data records only.

We must, however, get this newly assigned Primary Key value into each (one or more related) Order Line Item data record. That is why the output stream continues to a Copy Stage, and then a Join Stage; a Join to the Order Line Item data records.

This Join Stage will do all of its work in flight, in memory and without gating (without blocking or delay).

- The remainder of the Job consists of writing completed data records to some target Stage, in this case, two simple ASCII text files for Header data (Order Header data) and Detail data (Order Line Item data).

Again, the above Stages, and this larger design pattern are detailed in the August 2007 edition of this document. The only new technology is the Surrogate Key Generator, which is detailed in an example below.

(Database Server) Stored Procedures

As stated above, the first relational database servers could only Insert, Update, Delete and Select. (And enforce the unique Primary Key constraints.) Sybase was the first commercial relational database server to deliver Foreign Keys, column and table level Check Constraints, Default (Column Level) Constraint Clauses, Triggers, and then also, Stored Procedures.

A Stored Procedure is a database server resident piece of program code that may be used to validate data and more. The idea behind Stored Procedures and the other database objects listed above, was to place an amount of application programming inside the database server, to best ensure data integrity. Great idea, poor execution. For many, many years, there was no ANSI standard specification for a Stored Procedure Language (SPL); hence, the SPL found in Sybase is different than that found in Oracle, is different than just about everyone else. Only DB2 delivered (recently) an ANSI standard SPL implementation, and by now, few seem to care. Most modern relational database servers support Stored Procedures written in proprietary SPL, or Java, or C/C++.

Note: Whereas most modern relational database servers are fully parallel in their execution of most tasks, many still execute Stored Procedures in serial (non-parallel) mode.

Even the presence of a Stored Procedure call, either explicit or implicit, can prevent database server side parallelism, and lower observed performance.

Still, Stored Procedures exist, and you may wish to invoke them inside your IBM Information Server (IIS) DataStage component Jobs and related. Below we detail an example using the Stored Procedure Stage found within DataStage.

Note: The DataStage component Stored Procedure Stage (SPS) is not supported for every database product or platform, for various reasons.

You can still invoke SPL via an implicit call, embedding it inside any SQL SELECT. Above average database server products will also allow implicit invocation of SPL on SQL INSERT, UPDATE, and DELETE. These capabilities differ vendor by vendor.

Given a Stored Procedure named, MyUpshift(), an implicit call would resemble,

```
SELECT MyUpshift( someColumn ) FROM someTable WHERE { ... }
```

The IBM Information Server (IIS) DataStage component Stored Procedure Stage can act as a data source or target, or be placed in the middle of a data stream, and act as a Transformer. Below we detail an example of the latter.

Reject on load

Another common design pattern is how to handle or report reject records from a database table load. For example, you have thousands of new data records to Insert, but some will fail database server side integrity checking. Generally, you have one of three options;

- Discard the rejected data records.
- Leave them on the database server.

Most database servers have the concept of Reject Tables that manually or automatically receive failed data records.

- Return them to the calling function/client for further processing.

In this case, the calling function is the IBM Information Server (IIS) DataStage component Job.

Figure 29-6 below, displays a sample Job we detail in this edition of this document, that is configured to perform a SQL INSERT, and return rejected rows to a new data target; in this case, an ASCII text file.

Note: We could just as easily sent these Reject Rows to more Stages for further processing.

Send these records to a Copy Stage, and then whatever Stages you require.

While 4 data records were sent to the ODBC Connector Stage in Figure 29-6 for new record INSERT, 3 data records failed a database server side integrity constraint. (Primary Key violation, duplicate record.)

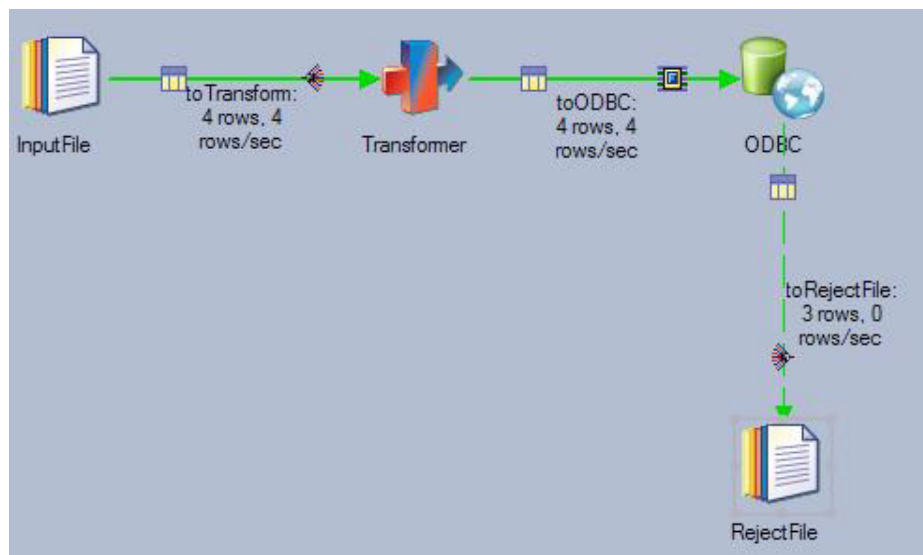


Figure 29-6 Sample Job with SQL INSERT, and a Reject Link.

29.2 Complete the following examples

In this section of this document, we create three new sets of examples. These examples are not accumulative; meaning, you can just complete the steps for example 2, if you so wish. Each example does use the same, small input ASCII text data file.

In order to host the required database server side objects (a Sequence, a Stored Procedure, and a SQL Table with an Primary Key Constraint for data integrity validation), you also need access to a SQL database server. These examples demonstrate IBM DB2/UDB and IBM Informix IDS.

You need to configure your IBM Information Server (IIS) DataStage component Project to make use of said database. How to configure and test ODBC drivers is detailed in the December 2007 edition of this document.

Step 1- Mandatory for all examples

1. Create a single, small ASCII text file equal to that as displayed in Figure 29-7.

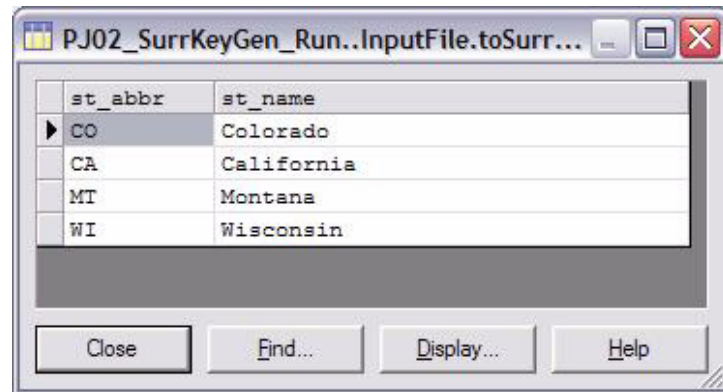


Figure 29-7 Sample input data set for all examples moving forward.

Step 2- Mandatory for the second example, Stored Procedure

2. The second example is detailed for IBM DB2/UDB.

On the server hosting DB2/UDB, a SQL command file may be invoked via the following command,

```
db2 -vf MyFile.sql
```

Create an ASCII text file (MyFile.sql) containing the following, and execute as detailed above,

```
CONNECT TO cust_db USER db2admin USING password
```

```
DROP PROCEDURE MyProcedure
```

```
CREATE PROCEDURE MyProcedure ( IN MyArg VARCHAR(32),\n                                OUT MyRet VARCHAR(32) )\n    LANGUAGE SQL
```

```
BEGIN                                \
                                     \
SET MyRet = UPPER ( MyArg ) ;        \
                                     \
END

CALL MyProcedure( 'tEst' , ? )
```

A code review related to the above includes;

- cust_db is the name of the DB2/UDB SQL database you are using.
- Our user name (USER) was db2admin.
- Password, (USING), obviously.
- The DROP PROCEDURE command is not required. We include it only to document and aid in any failed attempts.
- This Stored Procedure sends and receives 1 argument, of data type CHAR(32). This Stored Procedure folds the supplied value into an all uppercase return value
- The CALL statement is also not required, and is included only for your aid in testing.

Step 3- Mandatory for third example, capture Reject on load

3. The third example is detailed for IBM Informix IDS, but is offered as very portable SQL Syntax.

Execute the following SQL command,

```
CREATE TABLE state
(
    stablerCHAR(02) NOT NULL PRIMARY KEY,
    st_nameCHAR(32)
);
```

Example-1, Surrogate Key Generation Stage

The IBM InfoSphere Information Server (IIS) DataStage component Surrogate Key Generation (SKG) Stage can supply a dynamically calculated incremental key value itself, or via a database server resident object. When the SKG Stage supplying its own key value, a binary data file on disk is read from and written to; this file must exist before being accessed for the first time, and must always be

writable. When using a database resident object, the SKG Stage uses a database SQL Object of type Sequence. In the example that follows, we detail how to use a database server resident Sequence using DB2/UDB.

Note: Here we are using the database resident SQL Object of type Sequence directly, and thus natively.

Currently the Surrogate Key Generator (SKG) Stage supports DB2 and Oracle for this operation. Additionally as a native database call, we have to include database server client libraries in order for this example to function.

The database server resident SQL Object of type Sequence must be created by the DataStage component to IIS; you can not successfully use a Sequence that is created outside of this component. Generally then, we have 2 (count) DataStage Jobs or even 3.

- One Job to make the Sequence with the correct (next highest) key value to be assigned. We can use a Job Parameter to receive this value dynamically.
- The Job proper, that calls to use this key value.
- A Job to drop this database server resident Sequence.

We detail all 3 of these Jobs below.

4. New IBM InfoSphere Information Server (IIS) DataStage component Parallel Job to create the Sequence.

- a. Create, save and name a new Parallel Job equal to that as displayed in Figure 29-8.

This Job has 1 Stage only; From the Processing Drawer of the Palette view, Drag and Drop 1 Surrogate Key Generator Stage.

Configure this Stage as shown in the Properties dialog box, Stage -> Properties TAB.

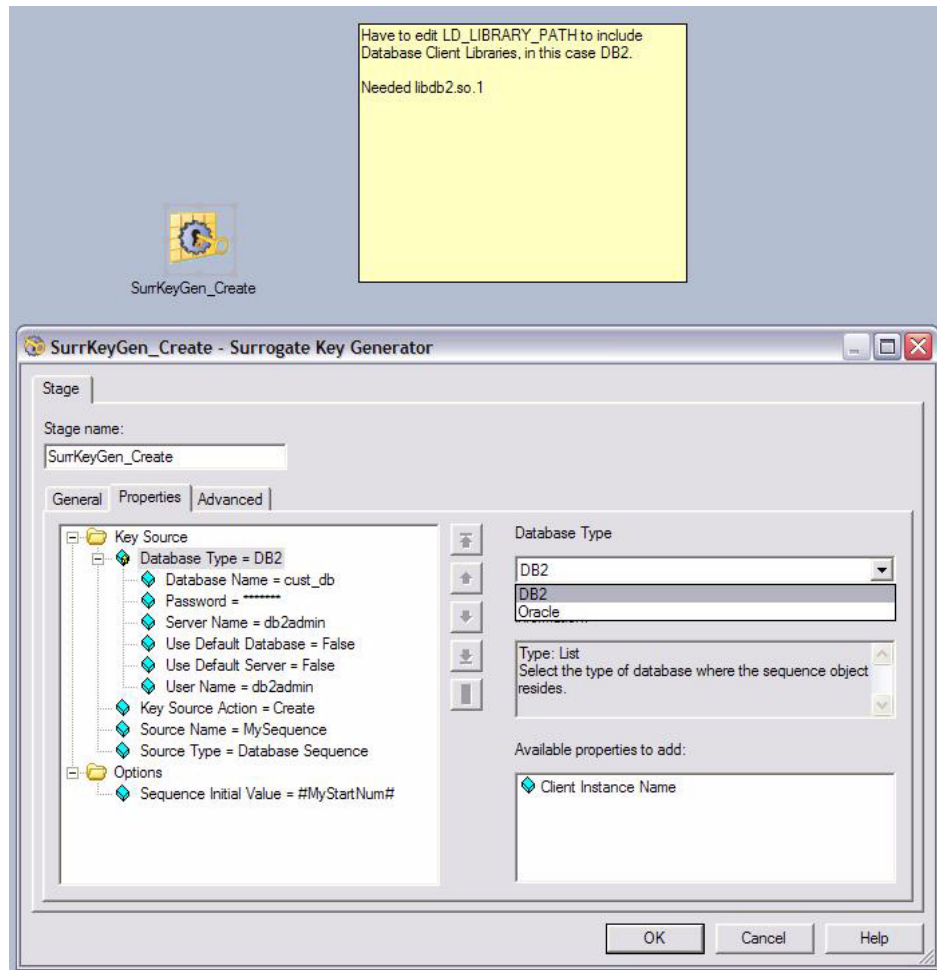


Figure 29-8 Complete Job to create a Sequence, only 1 Stage is needed.

This Stage makes use of a Job Property (Options -> Initial Sequence Value), entitled, MyStartNum. When running this job, we will be prompted dynamically for the Integer value to seed this key value generation with.

To create a new Job Property, complete the following,

- From the Menu Bar, select Edit -> Job Properties -> Parameters TAB.
- Enter a new Job Property entitled, MyStartNum, as displayed in Figure 29-9.

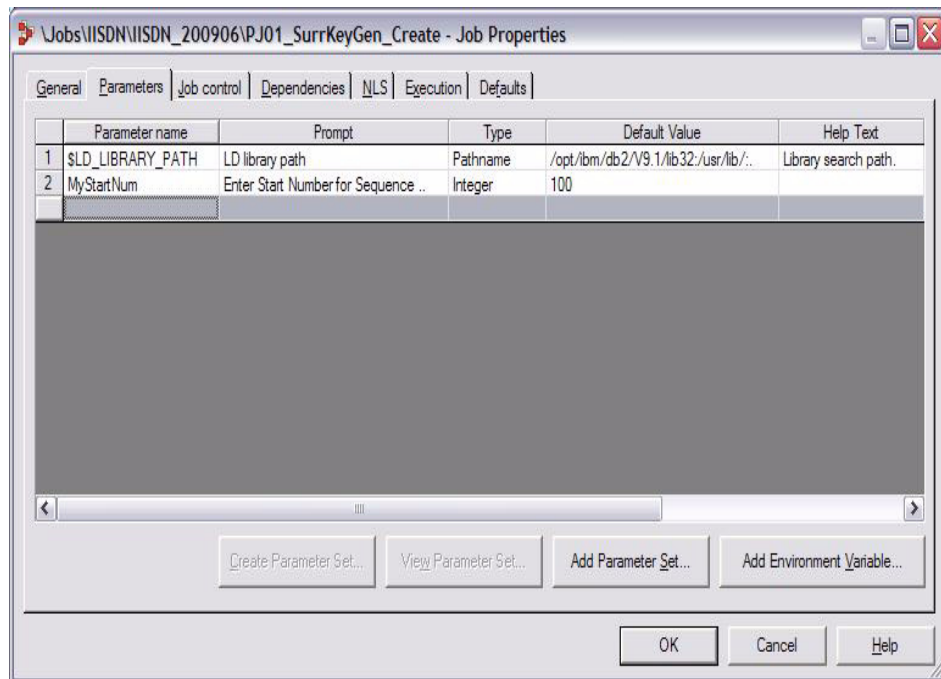


Figure 29-9 Editing Job Parameters.

- Using a DB2/UDB Sequence, required a database server specific client side library. In Figure 29-9, we also added an Environment Variable entitled, LD_LIBRARY_PATH.

(Actually LD_LIBRARY_PATH already exists for all Jobs. In essence, we are adding an override, changing its default setting.)

Where LD_LIBRARY_PATH was set to,
/usr/lib:.

We added /opt/ibm/db2/V9.1/lib32, which contained the libdb2.so.1 file. You will know which file you require after the first time you try to run this Job, and it fails. The Job log will list whatever database server specific client libraries were called for, but not found.

b. Compile and Run this Job.

With no Output Links, it is best to go to the Job Log, and determine if this Job ran successfully.

With a successful Job run, you will have created a new database server side resident SQL Object of type Sequence, with a given start value, which is ready for use by DataStage and the Surrogate Key Generator Stage.

5. New IBM InfoSphere Information Server (IIS) DataStage component Parallel Job to make use of the Sequence.
 - a. Create, save and name a new Parallel Job equal to that as displayed in Figure 29-10.

Note: The Job in Figure 29-10 has 3 Stages; an Input, an Output, and the Surrogate Key Generator (SKG) Stage in the middle.

This job represents the minimum use case for the SKG Stage. A more robust version of this job exists in Figure 29-3 and Figure 29-4.

- b. There are 2 TABs we need to adjust for this job.
The first, Stage -> Properties, is displayed below.in Figure 29-10.

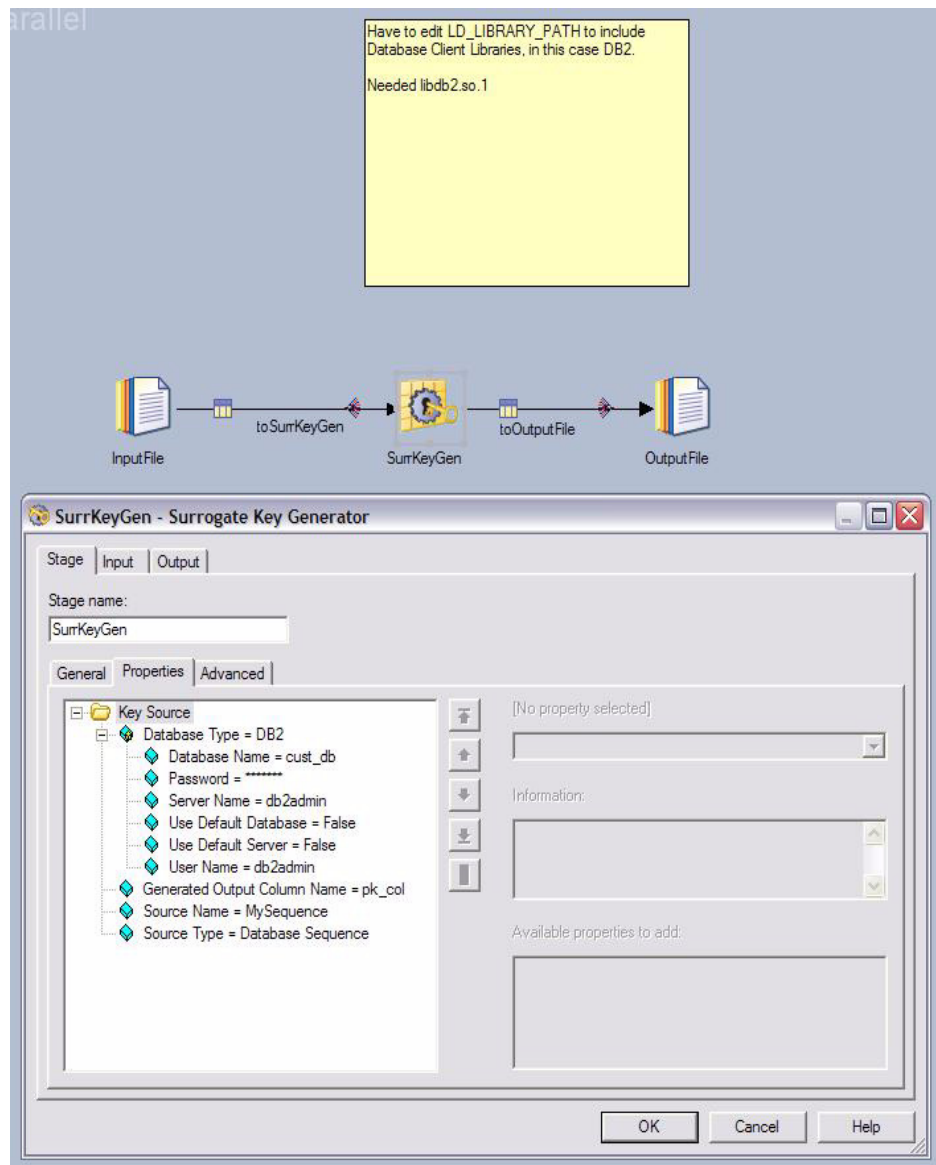


Figure 29-10 Actual job to use Surrogate Key Generator, given next key value.

The second TAB to adjust is displayed in Figure 29-11.

Here we are just specifying the output columns.

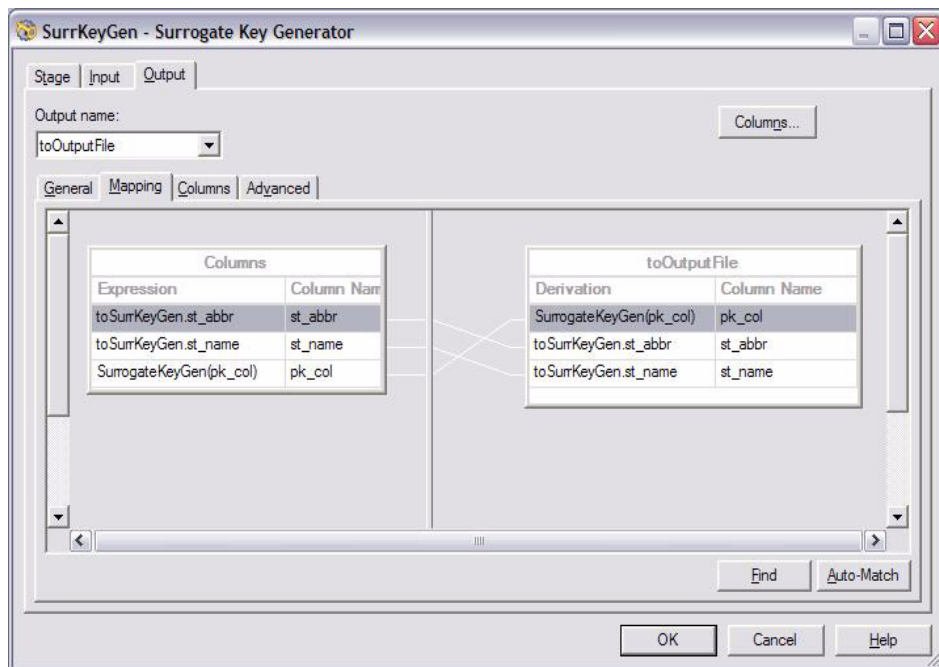
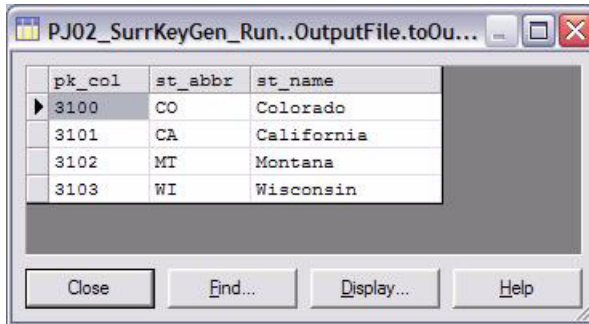


Figure 29-11 Output -> Mapping TAB to SKG Stage.

- c. As with the prior Job, you will need to adjust the LD_LIBRARY_PATH Environment Variable to include client side libraries.
- d. Compile and Run this Job.

A successful test appears in Figure 29-12.



pk_col	st_abbr	st_name
3100	CO	Colorado
3101	CA	California
3102	MT	Montana
3103	WI	Wisconsin

Figure 29-12 Successful output of Job.

6. New IBM InfoSphere Information Server (IIS) DataStage component Parallel Job to drop the Sequence.

A database server resident Sequence Object will exist and operate forever, at least until someone drops it. The reason to have a DataStage Component Job to drop a Sequence, is to be able to create a new Sequence with a new initial seed value. (In order that the Job in Step-4 above will run successfully second or subsequent times.)

Figure 29-13 displays the single Stage Job to drop a database server side resident SQL Object of type Sequence. Optionally Create, Save and Run this Job to complete your mastery of the Surrogate Key Generator (SKG) Stage.

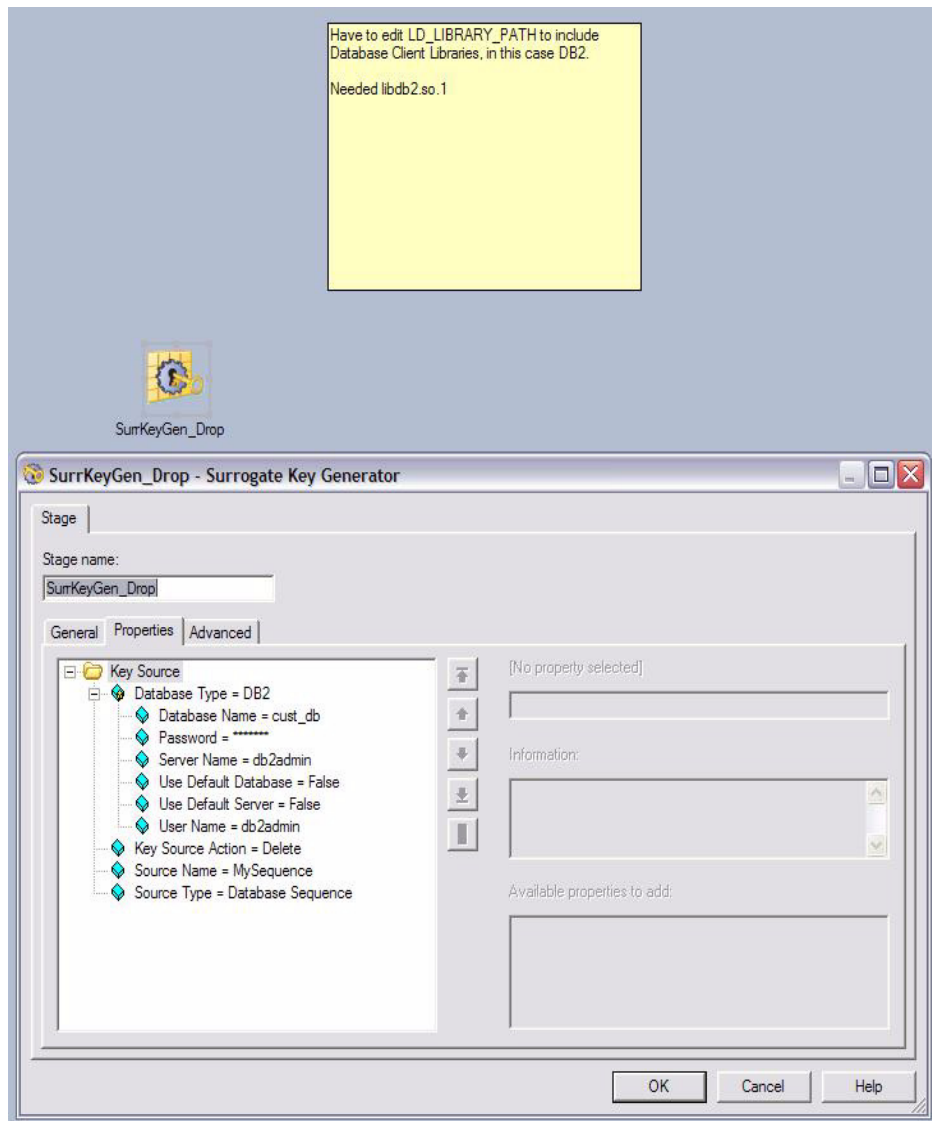


Figure 29-13 Dropping the Sequence Object, required to reset/set initial seed value.

Thus completes our work with the IBM InfoSphere Information Server (IIS) DataStage component Surrogate Key Generator Stage, and database server side resident objects of type Sequence.

Example-2, Stored Procedure Stage

The Surrogate Key Generator (SKG) Stage can interact with operating system files, or a database server resident SQL object to generate an incremental key value. That is why this Stage is located in the Processing drawer of the Palette View. The Stored Procedure Stage interacts only with SQL databases, and is located in the Database Drawer of the Palette View. Whereas the SKG Stage requires that its database server side dependent object be created inside the DataStage component to IIS, the Stored Procedure Stage has no such requirement.

As we are again making use of native database server side objects, it is required that we include database server specific client side libraries using LD_LIBRARY_PATH. Figure 29-14 displays a sample Job using the Stored Procedure (SPL) Stage to the DataStage component of IBM InfoSphere Information Server (IIS).

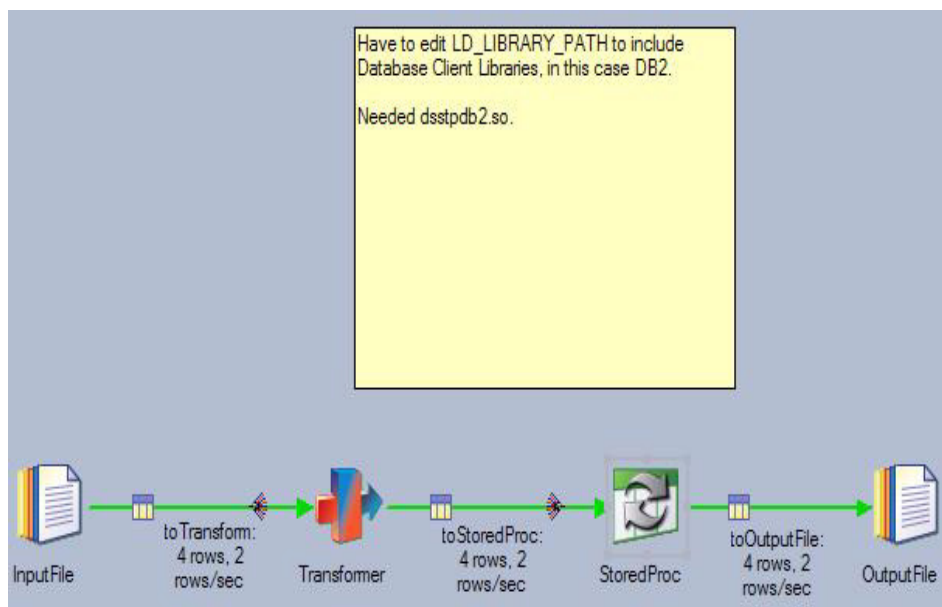


Figure 29-14 Sample Job using Stored Procedure Stage.

Comments related to Figure 29-14;

- As stated above, an SPL Stage can act as a Source or Target Stage. Here we detail using the SPL Stage as an intermediary Stage, as a Transformer.
- As a native database Stage, the SPL Stage requires client side libraries, and only supports certain database platforms. Here we detail using the

SPL Stage with DB2/UDB, although 5 or more specific database server platforms are supported.

Note: What if your specific database server platform is not supported by the Stored Procedure Stage? No worries, you can always invoke an SPL via an implicit call, embedding it inside the SQL SELECT, other.

- The Transformer Stage above is used to demonstrate our desire for explicit column data type casting whenever communicating with a database server.

The database server side Stored Procedure will have a finite column count for both input and output arguments, and have a distinct, precise data type for each. If the Stored Procedure expects a VARCHAR(32) input argument, we use the Transformer to change the data type of whatever we are sending to equal exactly a VARCHAR(32) value.

Most database servers require explicit column type casting. If we don't use a Transformer to manage this explicitly, you may not get the results you desire.

7. New IBM InfoSphere Information Server (IIS) DataStage component Parallel Job to invoke a Stored Procedure.

- a. Create a new DataStage Parallel Job equal to that as displayed in Figure 29-14.

Reposition and Rename all Stages and Links as shown.

- b. Edit the Transformer Stage.

In Example-2 above, the Stored Procedure we created and are now dependent on, receives one argument of type VARCHAR(32), and returns one argument of type VARCHAR(32).

Figure 29-15 details how we received 2 columns of type VARCHAR, but unbounded length; these columns coming from a flat file. Notice on the output side of this Stage we explicitly set the column data type of the st_name column to VARCHAR(32). The st_name column will be sent to our Stored Procedure.

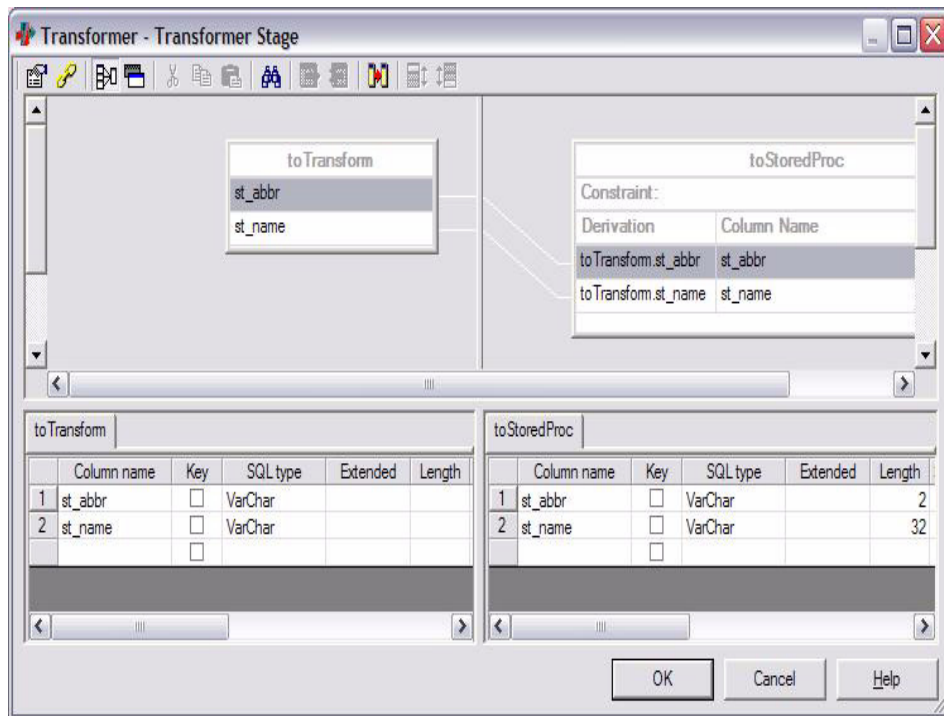


Figure 29-15 Transformer Stage, setting explicit column data type

c. Edit the Stored Procedure Stage.

There are several TABs we wish to edit in this Stage, so that it may function correctly.

i. In the Stage -> General TAB, set the following-

Example as shown in Figure 29-16.

Database vendor to DB2.

The Data source text entry field should equal the name of the database, not the DSN name.

Username and password.

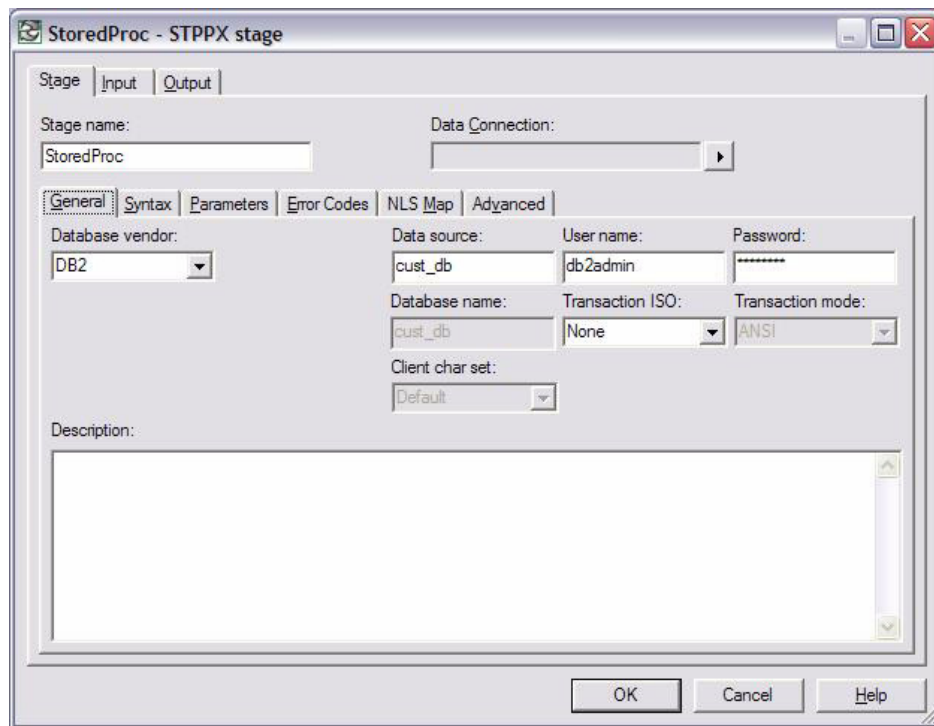


Figure 29-16 Stage -> General TAB to SPL Stage.

- ii. In the Stage -> Syntax TAB, set the following-
Example as shown in Figure 29-17.
Set the Procedure name.
Set the Procedure Type.
And set Database procedure type.

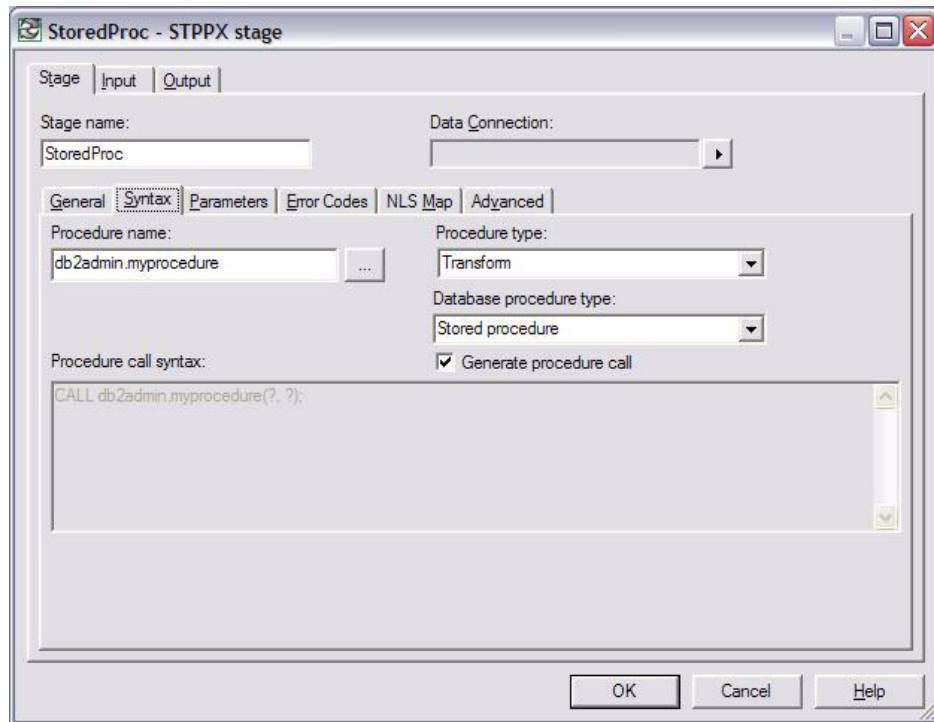


Figure 29-17 Stage -> Syntax TAB to SPL Stage.

- iii. In the Stage -> Parameters TAB, set the following-

Here is where we specify the Input and Output arguments to the Stored Procedure. In our use case, we are sending one column only, which is then overwritten with the returning value.

In Figure 29-17, you see MyArg, the named input parameter to the Stored Procedure, receives our input column, st_name. Because we overwrite the contents of st_name with the return value, st_name is set to a Parameter Type of, Input/Output.

We must also list the output parameter to the Stored Procedure, MyRet, or Parameter Type, Output.

Note: If this seems redundant, consider a larger mixture of parameters; some Input only, some Output only, some overwritten (Input/Output).

The net of this becomes;

- Each single Input argument and Output argument to the Stored Procedure gets a 1 line entry in this table.
- We wish for our input column to this Stored Procedure, st_name, to get overwritten, so it is listed as Input/Output. We could have carried an additional column just for the output, the return value of this Stored Procedure, but that would be boring.
- All other columns passing through this Stage, and referenced by the Stored Procedure, are listed as Input or Output.

Make your Stage -> Parameters TAB equal that as displayed in Figure 29-18.

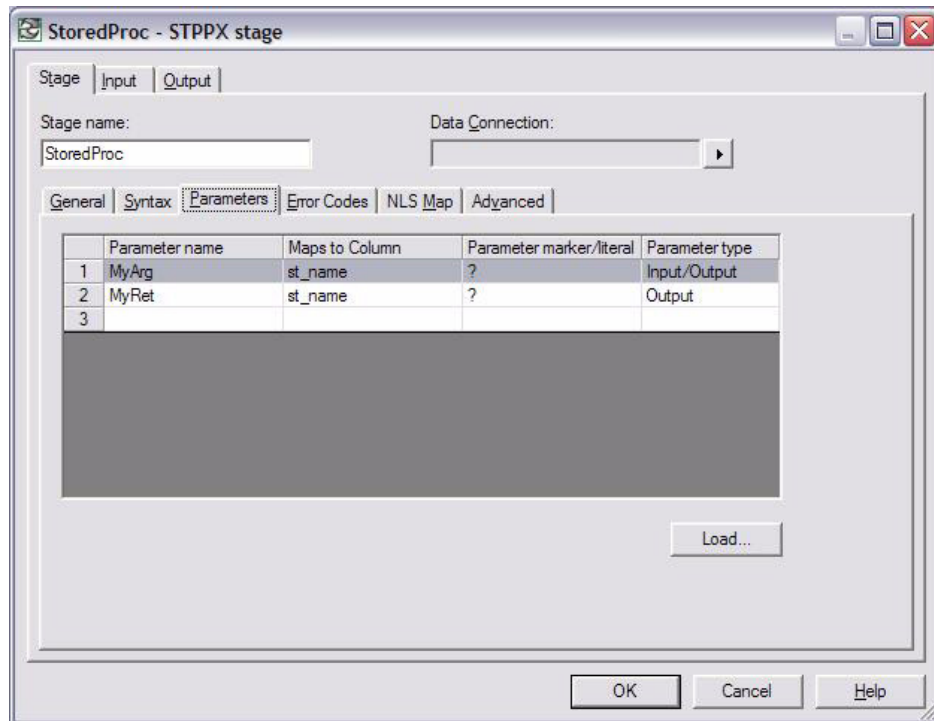


Figure 29-18 Stage -> Parameters TAB to SPL Stage.

- iv. In the Input -> General TAB, set the following-

No changes are required here, the default settings as displayed in Figure 29-19 meet our needs.

the main reason we display this page is to highlight the 'Execute procedure for each row' Check Box. Here you can call to use Stored Procedures that are essentially static; always returning the same data value, like maybe current temperature.

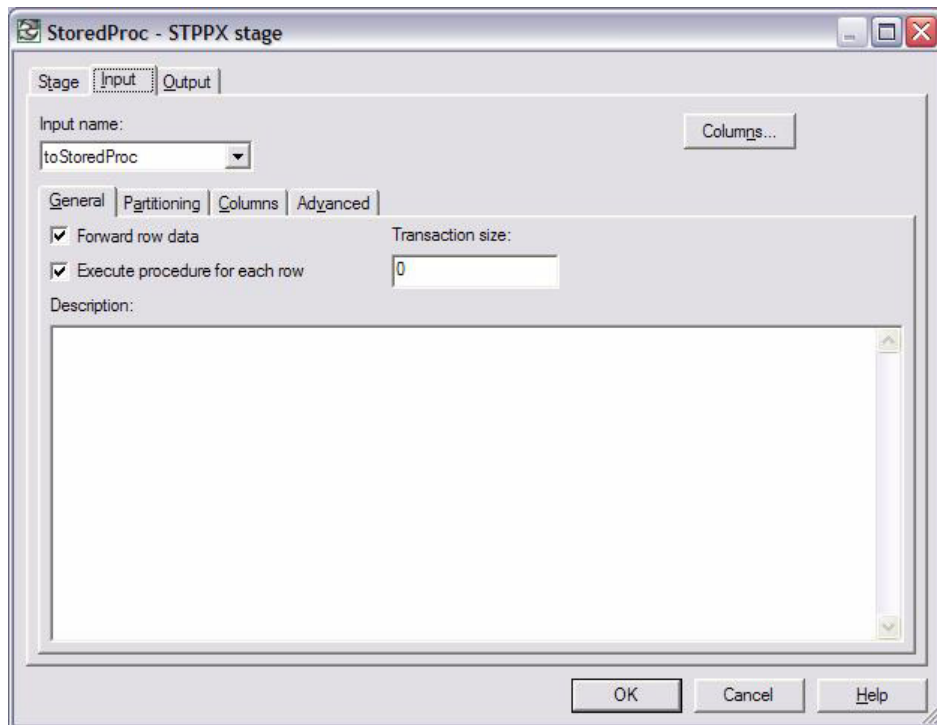


Figure 29-19 Input -> General TAB to SPL Stage.

- v. In the Output -> General TAB, set the following-

Example as shown in Figure 29-20.

Here we Un-Check the 'Procedure status to link' visual control.

If you leave this checked, you receive 2 additional output columns which you must define and receive. These columns contain diagnostic data from the database server. Example as shown,

ProcCode INTEGER(10)

ProcMess VARCHAR(128)

ProcCode is the SQLCODE from the resulting Stored Procedure invocation, whereas ProcMess is the short form error message.

Here we also see a Check Box to allow for a Stored Procedure to return 1 or more return records; we leave this control unchecked.

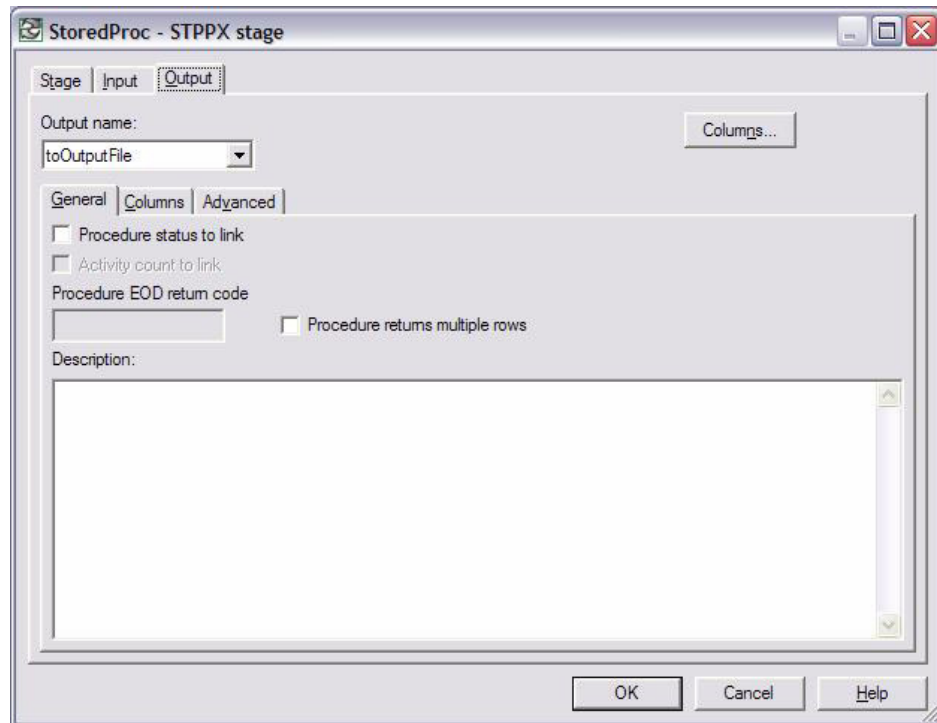


Figure 29-20 Output -> General TAB to SPL Stage.

vi. In the Output -> Columns TAB, set the following-

Example as shown in Figure 29-21.

No changes should be required here. Your result should match the image as displayed in Figure 29-21. If not, you have made an error in a previous step.

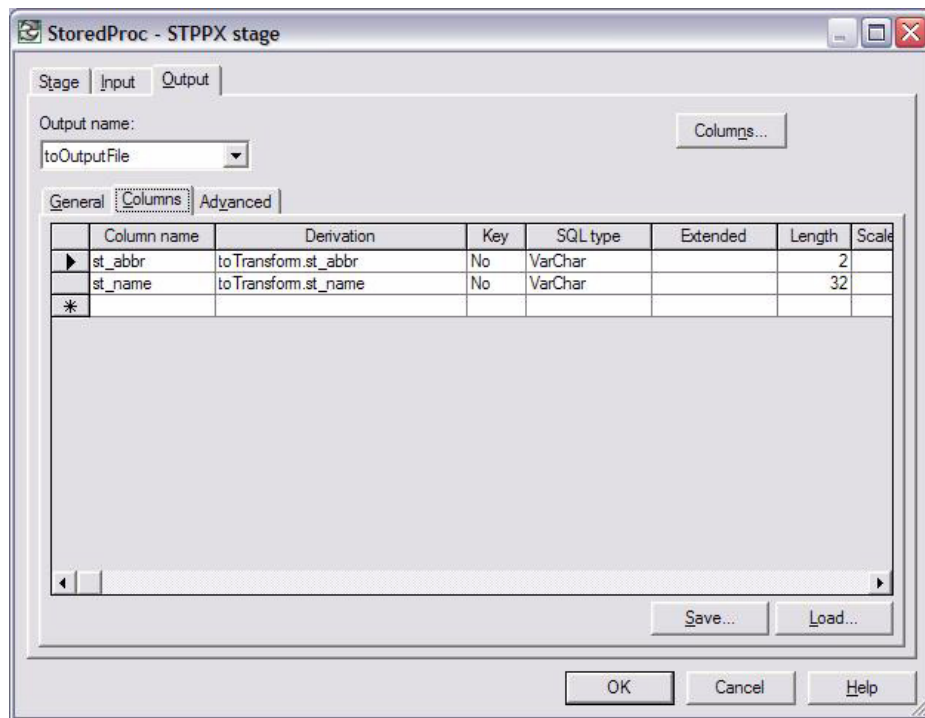


Figure 29-21 Output -> Columns TAB to SPL Stage.

- d. Don't forget to set LD_LIBRARY_PATH as before.
- e. Compile and Run this Job.

A successful sample run appears in Figure 29-22.

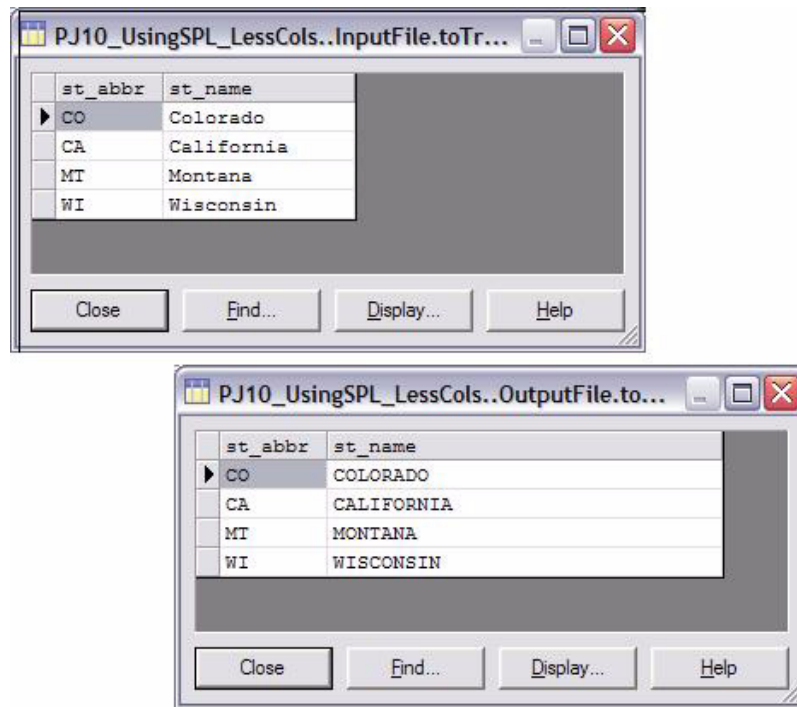


Figure 29-22 Results of successful Job run.

Example-3, Reject on load

The use case we detail in this section of this document is a Reject on Load; a database server table has integrity check constraints of any form, and during a SQL INSERT or UPDATE, the database server will reject zero or more rows.

Figure 29-23 displays the IBM InfoSphere Information Server (IIS) DataStage component Job we are about to create. A code review follows.

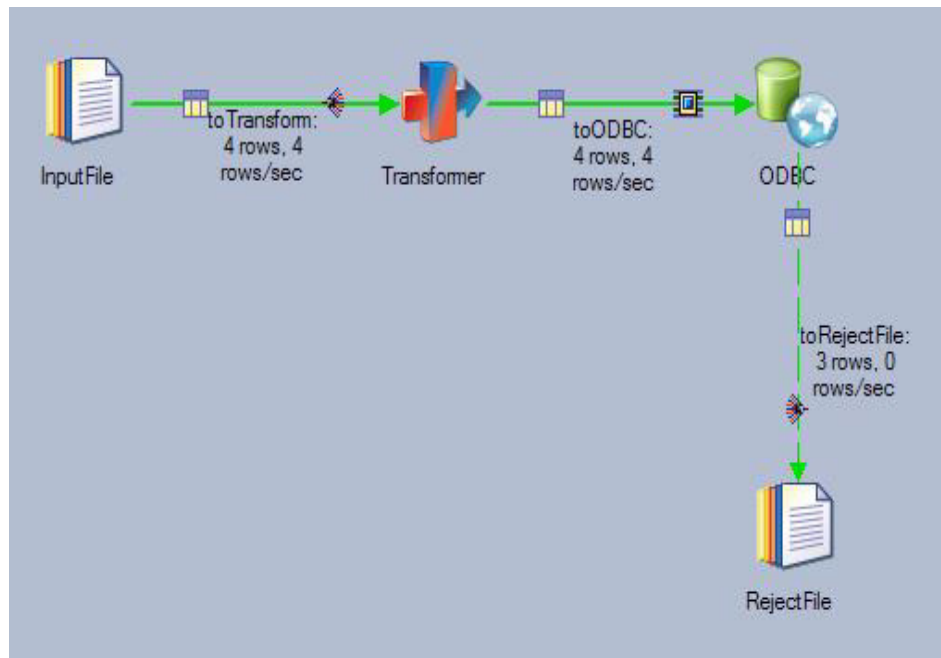


Figure 29-23 Sample Job, Reject on load.

A code review for Figure 29-22 is offered below;

- The only real working Stage is ODBC Connector, which can accept a Reject Link.
This Reject Link could lead to another processing type Stage; for simplicity, we used a terminating Stage, a flat file.
- The Transformer Stage is used to explicitly convert all submitted columns to the precise data type, as expected by the receiving database server table.

We are happy is ODBC related Stages read and write; we don't also ask them to recast data types, an operation which may fail.

- In the diagram above, 4 rows are sent to the ODBC Connector for SQL INSERT into a table, and 3 of these rows fail due a violation of a unique integrity constraint. (3 Of the submitted rows are duplicates.)

If you run this Job a second time, 4 rows will fail since they are now all duplicate. For testing, a SQL command file or similar needs to reset the table.

8. Complete the following;

- a. Create a new DataStage component Parallel Job equal to that as displayed in Figure 29-22. Reposition and Rename all Stages and Links as shown.
- b. Configure the Transformer Stage as shown in Figure 29-15, above.
- c. Configure the ODBC Connector Stage as shown in Figure 29-24.

This might be the first time you've used an ODBC Connector with an Input Link and a Reject Link.

Here, we are configuring the Input Link.

Note: We like using the ODBC Connector Stage as often as possible. If we use this Stage we a Parameter Set for the DSN Name, User name and Password, we could feasibly port these Jobs from one brand of database server to another with minimal impact.

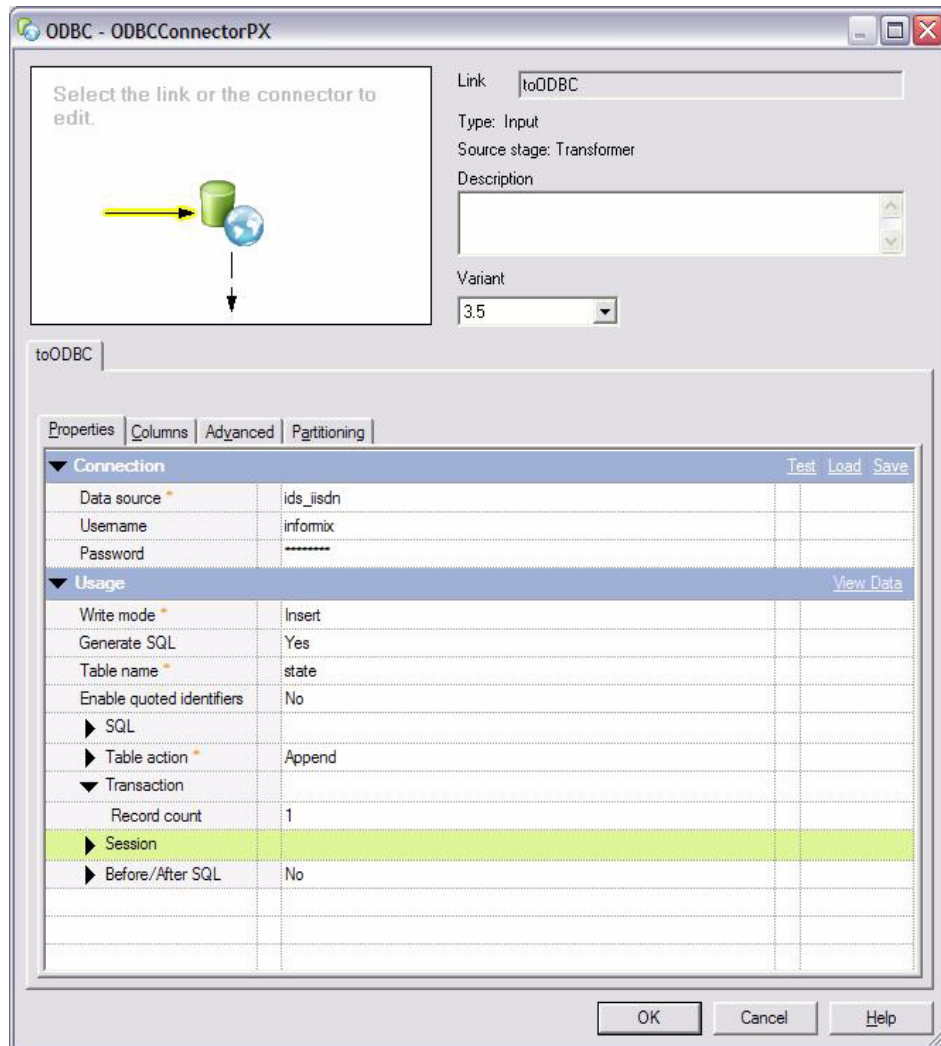


Figure 29-24 ODBC Connector, (Input Link), Properties TAB.

- d. If you Click the Reject Link in Figure 29-25, you get access to the Reject TAB.

Set the visual controls as displayed in Figure 29-25.

Do not incorrectly access the Properties TAB for the RejectLink; this only sets properties for the InputLink.

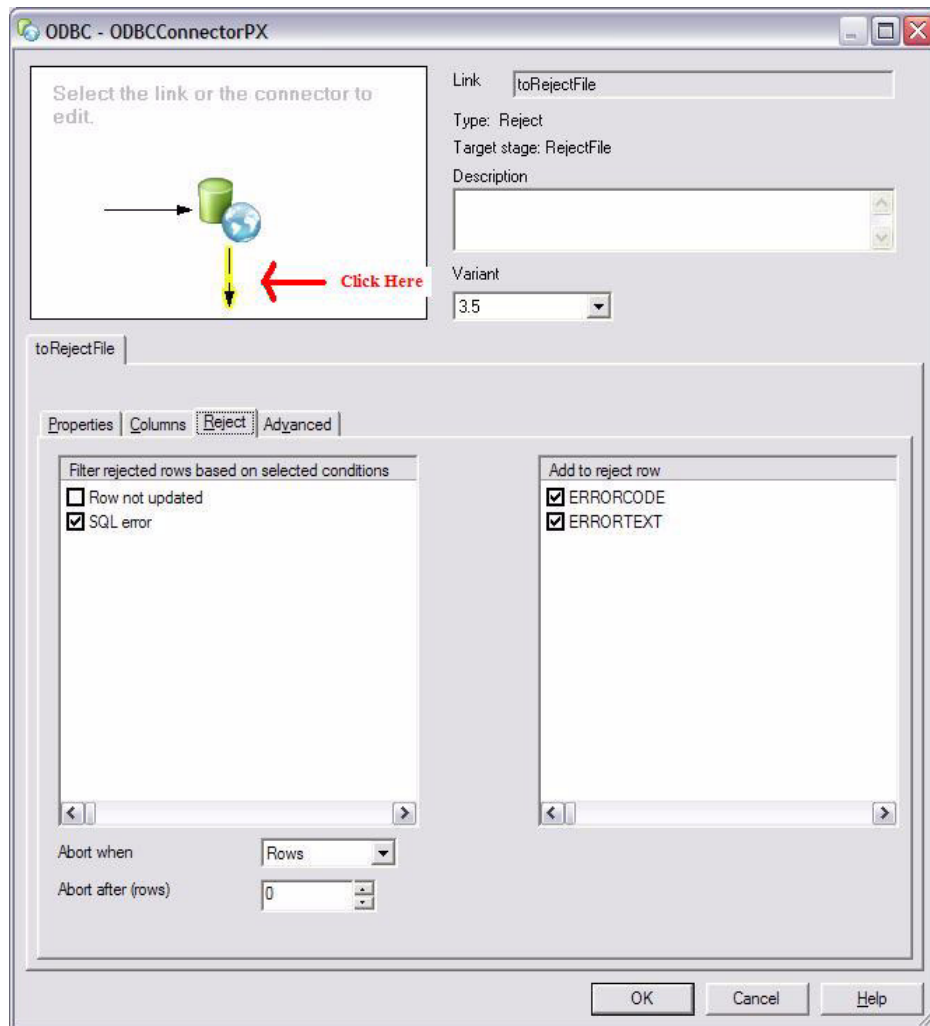


Figure 29-25 ODBC Connector, RejectLink, Reject TAB.

e. Compile and Run this Job.

A successful sample run appears in Figure 29-26.

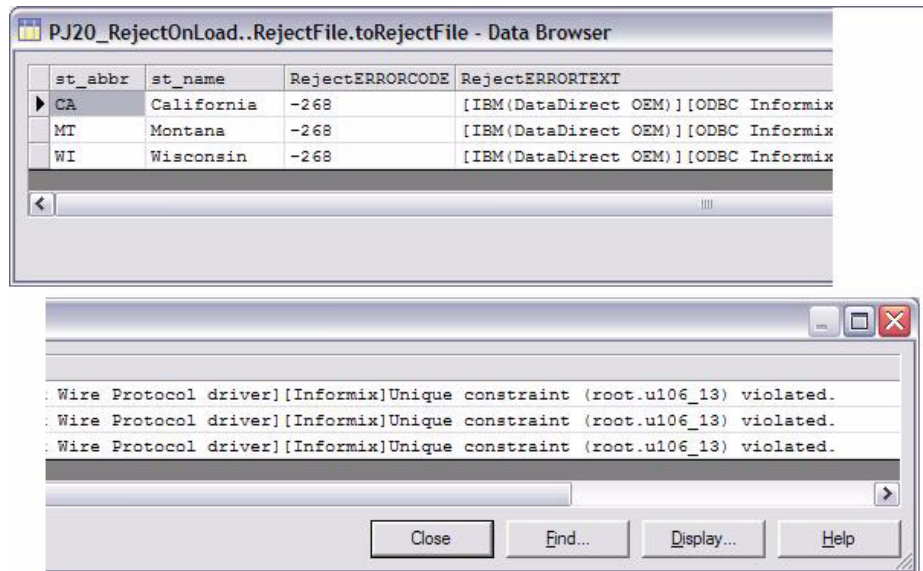


Figure 29-26 Successful sample job output.

29.3 In this document, we reviewed or created:

We detailed the use of a number of database server specific capabilities; namely, Surrogate Key Generation Stage, Stored Procedure Stage, and a design pattern of Reject on Load.

Persons who help this month.

Mark Carda, Jyoti Gupta, and Predrag (Max) Maksimovic.

Additional resources:

Legal statements:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating

trademarks that were owned by IBM at the time this information was published. A complete and current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

Other company, product or service names may be trademarks or service marks of others.

Special attributions:

The listed trademarks of the following companies require marking and attribution:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft trademark guidelines

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel trademark information

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

IBM InfoSphere Information Server Developer's Notebook -- June 2009 V1.4

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Other company, product, or service names may be trademarks or service marks of others.