IRI Software Integration With Oracle Job Scheduler

Table of Contents

Overview

This document provides information about the invocation of IRI's data extraction and transformation software (FACT and CoSort, respectively) from Oracle® Job Scheduler. It contains specific technical descriptions, and explains how IRI tools can be used directly, indirectly, or instead of more expensive or less efficient extraction and transformation functions in Oracle, Teradata®, and DataStage®.

Technical requirement

CoSort (SortCL) and Oracle Jobs Scheduler Integration. CoSort (SortCL) can be executed from the Oracle procedure language by adding a run command package. The package is a java class and is loaded into Oracle using a run command.

The Oracle database and CoSort are included in one server installation. Oracle users have permission to read, write, and execute a file.

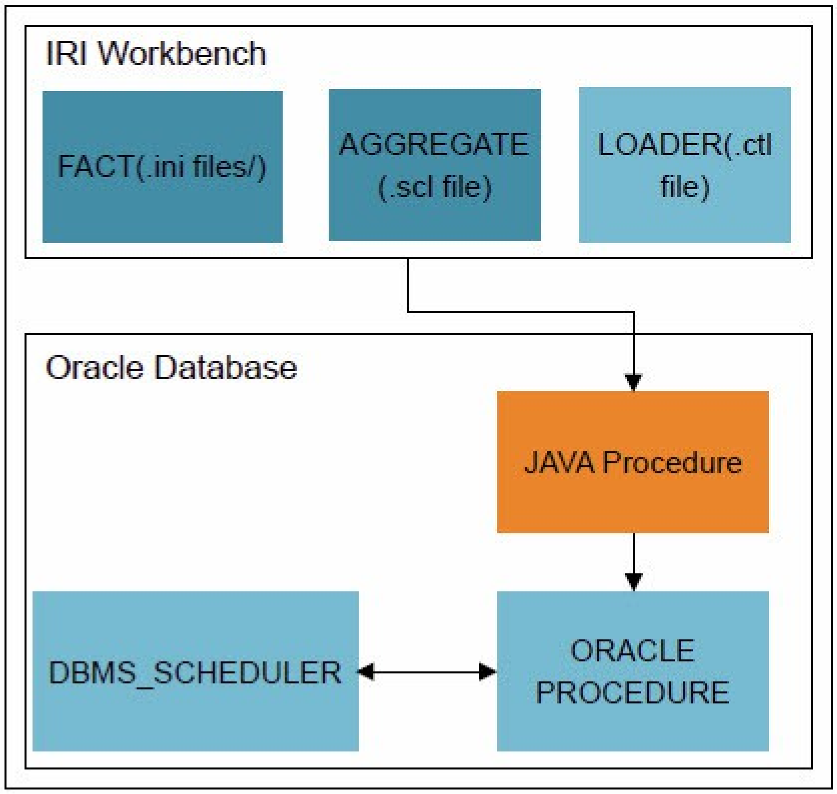
Software requirement

- CoSort v8 or above
- IRI Workbench
- Oracle server version 10 or above
- Bash script or shell script available for Nix type server and bat script available for Windows type server
- EasyShell

Summary

The following diagram depicts the global concept of how to integrate your ETL process, define the jobs schedule, and control the inter-job dependency process.

1. The job script of the ETL process (FACT, aggregate and loading) is created in the optional IRI Workbench GUI. IRI Workbench is used to create job scripts in the SortCL language.
2. The Java procedure is a Java class that performs the execution of any windows command execution outputs from FACT(.ini) files, Aggregate (.scl) files and Loader (.ctl) files.
3. The Oracle procedure is the Oracle subprogram that performs a specific action. You must declare and define the procedure before it can be invoked. The Oracle procedure is used to execute the process flow from ETL that you have created in IRI Workbench.
4. DBMS Scheduler is the Oracle package that provides a collection of scheduling functions and procedures that are callable from any PL/SQL program.



Preparation

Prepare the IRI Workbench

1. Open IRI Workbench and create a new IRI project by clicking **File**>**New**>**IRI Project**.
2. In the **Project name** field, enter a name for your project, and then click **Finish**. We are going to be using "DemoOracle" throughout this tutorial.
3. Create a new folder within the project by clicking **File**>**New**>**Other**. In the list that displays, click the arrow before **General** and then select **Folder**. Click **Next**.
4. Select your project name from the list as the parent folder. In the **Folder name** field, enter `ETL`. Click **Finish**.

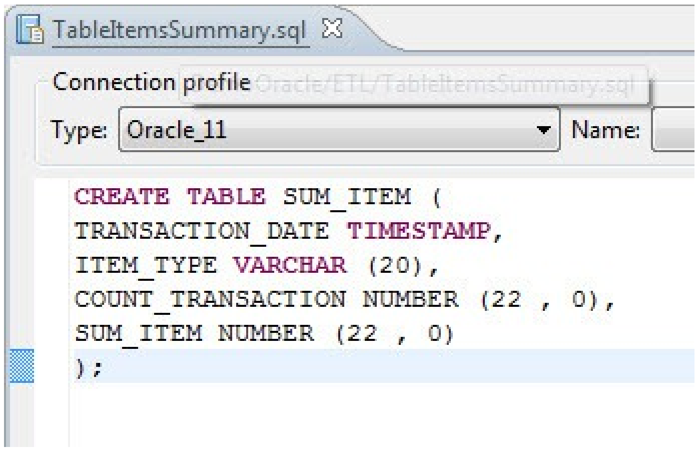
Create the target table

Make sure you are connected to your desired database with the appropriate connection profile settings.

1. Create an SQL File by clicking **File**>**New**>**Other**, and then the arrow before SQL Development. Select **SQL File** and click **Next**.
2. Select the project that you just created in the **Enter or select the parent folder** field, and in the **Field name** field, enter `TableItemsSummary.sql`.
3. In the editing window, type the following and save the file.

```
CREATE TABLE SUM_ITEM (
TRANSACTION_DATE TIMESTAMP,
ITEM_TYPE VARCHAR (20),
COUNT_TRANSACTION NUMBER (22 , 0),
SUM_ITEM NUMBER (22 , 0)
);
```

4. To execute the SQL file, right click on TableItemsSummary.sql in the Project Explorer and click **Execute SQL Files**.



```
CREATE TABLE SUM_ITEM (
TRANSACTION_DATE TIMESTAMP,
ITEM_TYPE VARCHAR (20),
COUNT_TRANSACTION NUMBER (22 , 0),
SUM_ITEM NUMBER (22 , 0)
);
```

Grant Java execution permissions for the Workbench

Make sure that you have the DBMS Java package installed on your machine.

1. Open the command line and connect to your database as the DBA.
2. Type the following command:
   ```
   select role from dba_roles where role like ’%JAVA%’;
   ```

   And then type:

   ```
   GRANT JAVASYSPRIV TO <INSERT USERNAME>;
   GRANT JAVAUSERPRIV TO <INSERT USERNAME>;
   ```

3. As the DBA, execute the following three commands:
   ```
   EXEC DBMS_JAVA.grant_permission(‘<INSERT USERNAME>’,
   'java.io.FilePermission', '<<ALL FILES>>', 'read ,write, execute, delete');

   EXEC DBMS_JAVA.grant_permission(‘<INSERT USERNAME>’,
   'SYS:java.lang.RuntimePermission', 'writeFileDescriptor', '*');

   EXEC DBMS_JAVA.grant_permission(‘<INSERT USERNAME>’,
   'SYS:java.lang.RuntimePermission', 'readFileDescriptor', '*');
   ```

Create the runScript Java procedure in ORACLE

The purpose of this procedure is to run any windows command. This procedure is created as a Java procedure in Oracle so that the output from the Java sources can be captured by any Oracle operations. In this case, you will also be creating a function to capture any output from this Java source called `runCommand`. If the output value is OK, then the windows command has been successfully executed. If an error message is returned, then the process cannot be done by this command.

1. Create an SQL File in your ETL folder called runScript.sql. In the **Database server type** field, select your database and click **Finish**.
2. In the editing window, type the following code and save the file. Reconnect to the database using the schema that was given permission for Java execution. Copy the code from the runScript.sql file and paste it into the SQL*Plus command line. Keep the connection open for later use.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "RUNSCRIPT" AS
import java.io.*;

  public class runScript {
  public static String run(String args) throws IOException{
  String retVal = null;
  int errorNum;
       try
       {
             String cmd = args;
             System.out.println("Starting the command..");
             Process p = Runtime.getRuntime().exec("cmd /C " + cmd);
             errorNum = p.waitFor();  // Wait for proc to complete.
```

```
            //if errorNum does not equal 0 then there was an error while handling the
            //command
            if(errorNum != 0)
            {
                  retVal = "ERROR";
            }
            else
            {
                  retVal = "OK";
            }
      }
      catch(Exception e)
      {
            e.printStackTrace();
            retVal = e.toString();
      }

      return retVal;

  }
};
/
```

Create the runCommand function in Oracle

1. Create an SQL file in your ETL folder named runCommand.sql. In the **Database server type** field, select your database, and then click **Finish**.
2. In the editing window, type the following code and save the file. Copy and paste the code into the SQL*Plus command line again.

```
CREATE OR REPLACE FUNCTION runCommand (pCommand IN VARCHAR2)
RETURN VARCHAR2 IS
LANGUAGE JAVA NAME 'runScript.run (java.lang.String) return java.lang.String ';
/
```

ETL process

FACT

1. Create an INI file by clicking **File**>**New**>**IRI**>**FACT Config File**, and then click **Next**.
2. In the **Folder** field, select the project that you just created, and in the **Field name** field, enter `items.ini`.
3. In the next few steps, enter your desired database connection settings and make sure that the following options are corresponding with the following values.

```
Query = “SELECT * FROM ITEMS”
Outfile = items.dat
Outformat = Variable
Delim = ‘ | ’
Addlastdelim = No
Removelf = No
Framefield = items
Framchar = ’ " ‘
```

Aggregate

1. Using the CoSort menu, create a new SCL script with the code shown below and run it as an IRI job.

```
/INFILE = C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.dat
   /FIELD=(transaction_date ,POSITION=1 ,SEPARATOR='|', ISO_DATE)
   /FIELD=(item_type        ,POSITION=2 ,SEPARATOR='|')
   /FIELD=(item_size        ,POSITION=3 ,SEPARATOR='|')
   /FIELD=(transaction_id   ,POSITION=4 ,SEPARATOR='|')
   /FIELD=(total            ,POSITION=5 ,SEPARATOR='|')
/SORT
   /KEY = transaction_date
   /KEY = item_type

/OUTFILE = C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.agg
   /FIELD=(transaction_date ,POSITION=1 ,SEPARATOR='|', ISO_DATE)
   /FIELD=(item_type        ,POSITION=2 ,SEPARATOR='|')
   /FIELD=(count_item       ,POSITION=3 ,SEPARATOR='|')
   /FIELD=(sum_item         ,POSITION=4 ,SEPARATOR='|')
   /COUNT count_item BREAK transaction_date OR item_type
   /SUM sum_item from total BREAK transaction_date OR item_type

/STAT = C:\IRI\CoSort95\workbench\workspace\Test2\ETL\books_agg.stat
```

Load

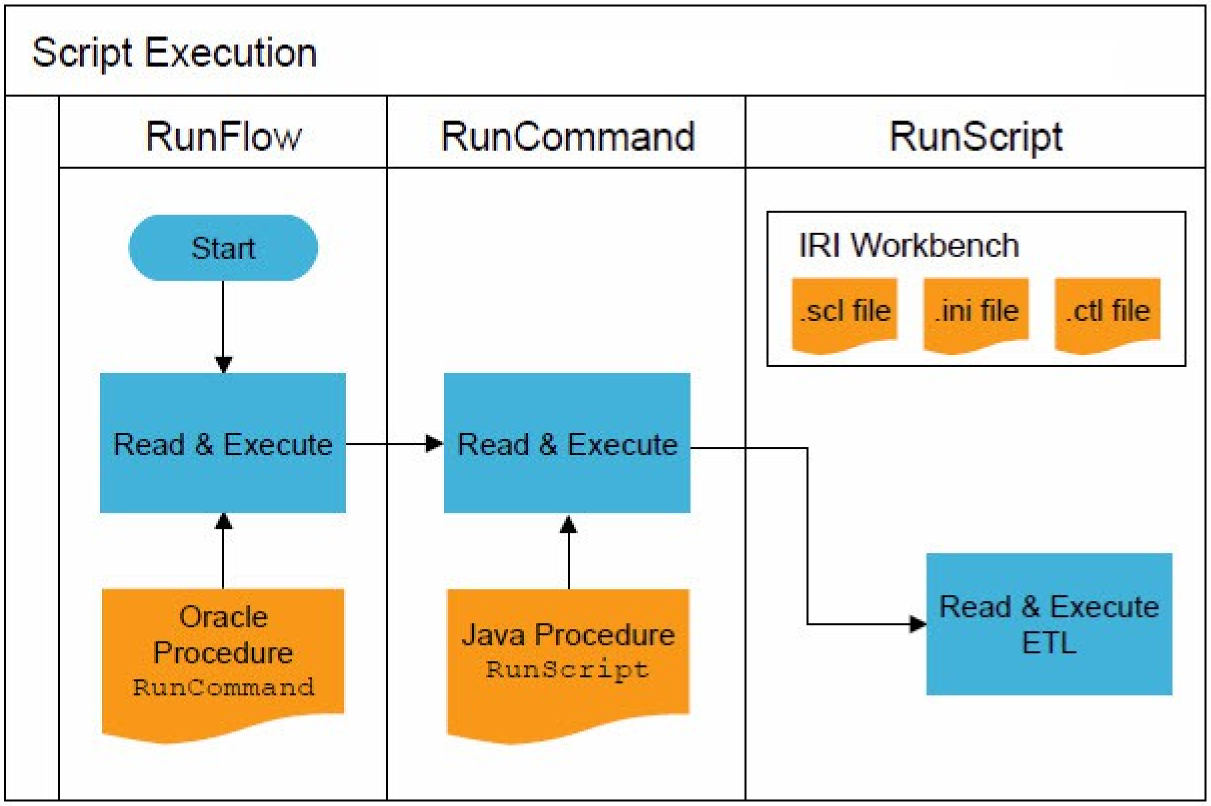
1. Load the data into the Oracle database by creating the following control (CTL) file called items_agg.ctl. Right click on the ETL folder and click **New**>**File**.

```
LOAD DATA
INFILE 'C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.agg'
APPEND
INTO TABLE SUM_ITEM
FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
     TRANSACTION_DATE TIMESTAMP "SYYYYMMDDHH24MISS.FF6",
     ITEM_TYPE,
     COUNT_ITME,
     SUM_ITEM
)
```

2. Right click on the items_agg.ctl file and select **Easy Shell>Open** and run the following command: sqlldr userid=<username>/<password> control=items_agg.ctl DIRECT=true

Assemble the ETL process

The steps of the ETL process are assembled into one Oracle procedure. This procedure can run ETL processing and perform multiple jobs. The following diagram depicts how the process is assembled.



The Java program runScript serves to run any windows commands, including FACT, sortCL, or sqlldr. The Oracle function runCommand serves to capture any messages from the Java program that are generated, either a success message or an error message.

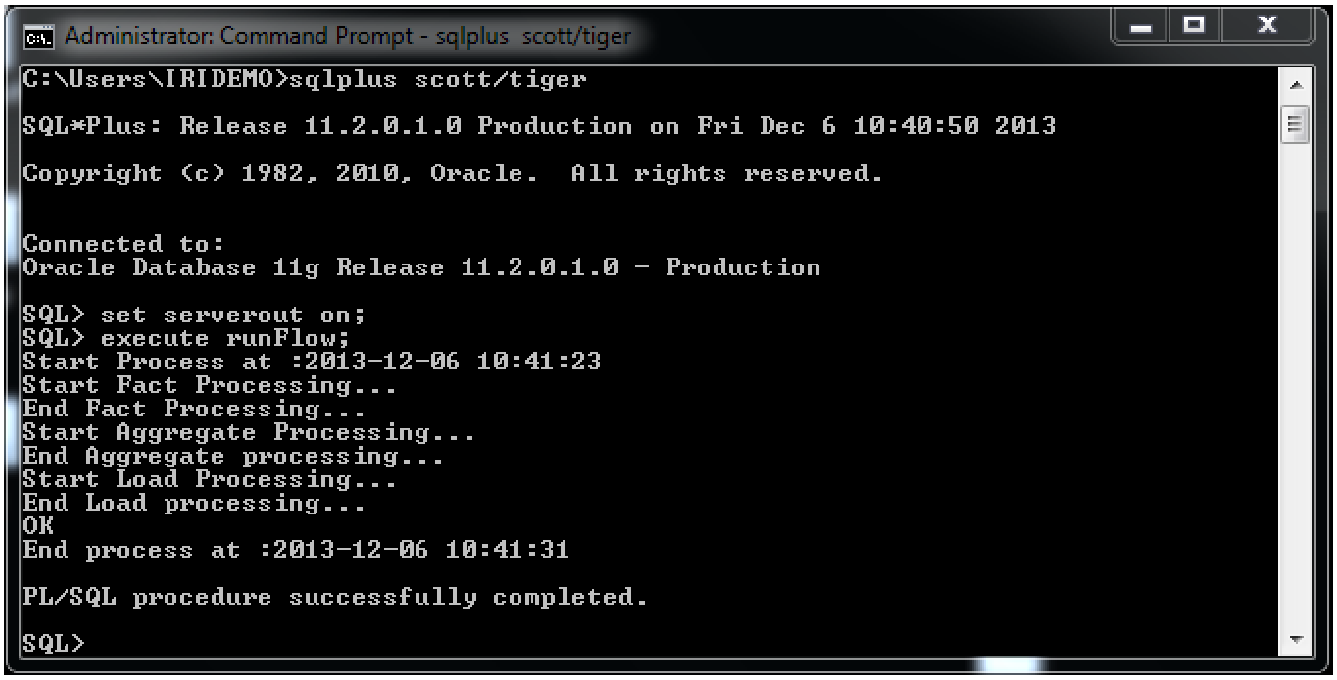
The Oracle procedure runFlow serves to assemble the ETL process as an interrelated process. If the previous processes failed, then the next process will not run.

Create the runFlow procedure in Oracle

1. Create an SQL file in your ETL folder called runFlow.sql. In the **Database server type** field, select your database, and then click **Finish**.
2. In the editing window, type the following code and save the file. Also, open the database connection under the user with permissions, and copy and paste the code into SQL*Plus.

```
CREATE OR REPLACE procedure runFlow as
   retVal varchar2(10);
begin
   dbms_output.put_line ('Start Process at :' || to_char (sysdate, 'yyyy-mm-dd
hh24:mi:ss'));
   --start fact
   retVal := null;
   dbms_output.put_line ('Start Fact Processing...');
   retVal := runCommand ('fact
C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.ini');
   if retVal is null or retVal <> 'OK' then
         dbms_output.put_line (retVal);
         GOTO endProgram;
   end if;
   dbms_output.put_line ('End Fact Processing...');
   -- Call sortcl and the scl file
   dbms_output.put_line ('Start Aggregate Processing...');
   retVal := null;
   retVal := runCommand
('sortcl/spec=C:/IRI/CoSort95/workbench/workspace/DemoOracle/ETL/item_agg.scl');
   if retVal is null or retVal <> 'OK' then
         dbms_output.put_line (retVal);
         GOTO endProgram;
   end if;
   dbms_output.put_line ('End Aggregate processing...');
   -- Start SQLLoader
   dbms_output.put_line ('Start Load Processing...');
   retval := null;
   retval := runCommand ('sqlldr control=C:\IRI\CoSort95\workbench\workspace\DemoOr-
acle\ETL\items_agg.ctl userid=scott/tiger DIRECT=TRUE.');
   if retVal is null or retVal <> 'OK' then
         dbms_output.put_line (retVal);
         GOTO endProgram;
   end if;
   dbms_output.put_line ('End Load processing...');
   dbms_output.put_line (retVal);
   --Terminate program if error occurred or terminate regularly
   <<endProgram>>
   dbms_output.put_line ('End process at :'||to_char (sysdate, 'yyyy-mm-dd
hh24:mi:ss'));
   null;
exception
   WHEN OTHERS THEN
   dbms_output.put_line (sqlerrm);
end;
/
```

3. Open Easy Shell and connect to your database. Enter `set serveroutput on;` followed by `execute runFlow;`



```
C:\Users\IRIDEMO>sqlplus scott/tiger

SQL*Plus: Release 11.2.0.1.0 Production on Fri Dec 6 10:40:50 2013

Copyright (c) 1982, 2010, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Release 11.2.0.1.0 - Production

SQL> set serverout on;
SQL> execute runFlow;
Start Process at :2013-12-06 10:41:23
Start Fact Processing...
End Fact Processing...
Start Aggregate Processing...
End Aggregate processing...
Start Load Processing...
End Load processing...
OK
End process at :2013-12-06 10:41:31

PL/SQL procedure successfully completed.

SQL>
```

Note that it takes 9 seconds for the E, T, and L processing of 1 million records.

The dependencies of each process are determined by the `retVal` parameter. If `retVal` contains NULL values or does not contain the OK value, the process will not continue. It will continue if the previous process has finished.

The result, output and statistics from the CoSort file will be created, and the result will be loaded into the database.

The resulting output files created by FACT and CoSort are:

- items.dat - the result of the FACT table items
- items.agg - the result of aggregate data that was sorted
- SortCL job statistics file

Create an Oracle scheduling job

Use DBMS_JOB to create an oracle scheduling job and assign it the oracle procedure you previously created.

The dependencies of DBMS_JOB are:

- dba_jobs
- dba_jobs_running
- all_jobs
- all_jobs_running
- user_jobs
- user_jobs_running

and the following is the common interval table for assigning a job and date setting:

Interval	**Description**
SYSDATE + 1	Execute daily
SYSDATE + 7	Execute once per week
SYSDATE + 1/24	Execute hourly
SYSDATE + 10/1440	Execute every 10 minutes
SYSDATE + 30/86400	Execute every 30 seconds
NULL	Do no re-execute

The line "*v_date date := to_date('20130102 080000','yyyymmdd hh24miss');*" is used to indicate the specific date and time of execution. The first part inside the brackets indicates the date and time while the second indicates the format. Therefore in this example it would execute on the second of January 2013 at 8am.

Create job

1. Create an SQL file in your ETL folder called jobSchedule.sql. In the **Database server type** field, select your database, and then click **Finish**.
2. In the editing window, type the following code and save the file. Be sure to use your preferred date and time.

```
DECLARE
JobNo user_jobs.job%TYPE;
v_date date := to_date('20130308 050000','yyyymmdd hh24miss');
BEGIN
    dbms_job.submit( JobNo,  --Job ID
    'begin runFlow; end;', -- Procedure to execute
    v_date, -- start running at
    'sysdate + 1' -- interval of jobs
    );
    COMMIT;
END;
/
```

Example job

1. Assign the job to run every Tuesday at 8 am and every Friday at 3 pm.

```
DECLARE
    JobNo user_jobs.job%TYPE;
    v_date1 date := to_date('20130312 080000','yyyymmdd hh24miss');
    v_date2 date := to_date('20130315 150000','yyyymmdd hh24miss');
BEGIN
    dbms_job.submit( JobNo,  --Job ID
    'begin runFlow; end;', -- Procedure to execute
    v_date1, -- start running at
    'SYSDATE + 7' -- interval of jobs
    );
COMMIT;
  dbms_job.submit(  JobNo,  --Job ID
  'begin runFlow; end;', -- Procedure to execute
  v_date2, -- start running at
  'SYSDATE + 7' -- interval of jobs
  );
 COMMIT;
 END;
/
```

2. Check the job status by running this query:

```
select job, next_date, next_sec, broken, interval, what from user_jobs;
```

Inter-Job dependency process

To further enhance the utility of the scheduler, you could create a process that would handle error handling if a process during the procedure failed. You would need to create two items:

- table to store the process control
- modular program to handle step-by-step flow

Create Table

Create a table that will accommodate the command of a flow using the following table DLL:

```
create table flow_process(
fp_id number, --Flow Process unique ID
fp_name varchar2(100), --Flow Process Name
fp_command varchar2(400), --Command
fp_seq number, --Sequence Process
fp_stat number --Sequence status
);
```

These tables define a certain flow of a process. The fp_command column is used to store the command of a flow.

The column fp_stat will read the status of a process, and frs_desc provides a description of that status.

In this case, the fp_stat column will be filled with:

1 = Done

9 = Error

```
Insert into flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
  values(1, 'Run Fact', 'RunPieces(''fact
C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.ini'')', 1, 1);

Insert into flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
  values(2, 'Run SortCL', 'RunPieces(''sortcl
/spec=C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\item_sort.scl'')', 2, 1);

Insert into flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
  values(3, 'Run Loader', 'RunPieces(''sqlldr userid=iriwb/iriwb@orcl
control=C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\load_agg.ctl
DIRECT=true'')', 3, 1);

Commit;
```

Modular Program

Since we already have a procedure to handle each ETL process respectively, we can modify our current runFlow process to handle the errors.

```
   CREATE OR REPLACE procedure runFlow as
      retVal varchar2(10);
   begin
      dbms_output.put_line ('Start Process at :' || to_char (sysdate, 'yyyy-mm-dd
   hh24:mi:ss'));
      --start fact
      retVal := null;
      dbms_output.put_line ('Start Fact Processing...');
      retVal := runCommand ('fact
   C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.ini');
      if retVal is null or retVal <> 'OK' then
            dbms_output.put_line (retVal);
      Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
      values(1, 'Run Fact', 'RunProcess(''fact
C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.ini'')', 1, 9);
            GOTO endProgram;
      end if;
      Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
      values(1, 'Run Fact', 'RunProcess(''fact
C:\IRI\CoSort95\workbench\workspace\DemoOracle\ETL\items.ini'')', 1, 1);
      dbms_output.put_line ('End Fact Processing...');
      -- Call sortcl and the scl file
      dbms_output.put_line ('Start Aggregate Processing...');
      retVal := null;
      retVal := runCommand
   ('sortcl/spec=C:/IRI/CoSort95/workbench/workspace/DemoOracle/ETL/item_agg.scl');
      if retVal is null or retVal <> 'OK' then
            dbms_output.put_line (retVal);
             Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
         values(1, 'Run SortCL',
   'RunProcess(''sortcl/spec=C:/IRI/CoSort95/workbench/workspace/DemoOracle/ETL/item_ag-
   g.scl'')', 1, 9);
            GOTO endProgram;
      end if;
      Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
         values(1, 'Run SortCL',
   'RunProcess(''sortcl/spec=C:/IRI/CoSort95/workbench/workspace/DemoOracle/ETL/item_ag-
   g.scl'')', 1, 1);
      dbms_output.put_line ('End Aggregate processing...');
      -- Start SQLLoader
      dbms_output.put_line ('Start Load Processing...');
      retval := null;
      retval := runCommand ('sqlldr control=C:\IRI\CoSort95\workbench\workspace\DemoOr-
   acle\ETL\items_agg.ctl userid=scott/tiger DIRECT=TRUE.');
      if retVal is null or retVal <> 'OK' then
            dbms_output.put_line (retVal);
   Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
         values(1, 'Run SQL*Loader', 'RunProcess('' sqlldr control=C:\IRI\CoSort95\work-
   bench\workspace\DemoOracle\ETL\items_agg.ctl userid=scott/tiger DIRECT=TRUE.)', 1, 9);
            GOTO endProgram;
      end if;
      Insert into scott.flow_process(fp_id, fp_name, fp_command, fp_seq, fp_stat)
         values(1, 'Run SQL*Loader', 'RunProcess('' sqlldr control=C:\IRI\CoSort95\work-
   bench\workspace\DemoOracle\ETL\items_agg.ctl userid=scott/tiger DIRECT=TRUE.)', 1, 1);
      dbms_output.put_line ('End Load processing...');
      dbms_output.put_line (retVal);
      --Terminate program if error occurred or terminate regularly
      <<endProgram>>
      dbms_output.put_line ('End process at :'||to_char (sysdate, 'yyyy-mm-dd
   hh24:mi:ss'));
      null;
   exception
      WHEN OTHERS THEN
      dbms_output.put_line (sqlerrm);
   end;
   /
```

Note that the only part modified was the added insert lines of the ETL process respectively. If during the ETL process that step fails, then the insert into table line will set the status column to 9. If the process however executes with no errors, then the status is set to 1.