ETL DEVELOPMENT GUIDELINES

DEVELOPMENT GUIDELINES and BEST PRACTICES

ETL, SQL and Batch programming – Data Team

Document History

Revision	Section(s)	Description	Editor	Date
1.0	-	Draft	Jaime Medina	21/06/2010
1.1		Small addition	Enrique Godino	10/12/2010
1.2		Logging section added	Enrique Godino	17/12/2010
1.3		Major revision, still in progress	Enrique Godino	10/03/2011
1.4		SQL Commit control added	Enrique Godino	15/04/2011
1.5	DB	Adding guideline to use (NOLOCK)	Enrique Godino	13/05/2011
1.6	JTI Data Rules	UTD Date Time fields	Enrique Godino	23/05/2012

Review and Approval

Team	Name	Signature	Date
GDC	Enrique Godino		
DBA	Brigitte Alonso		
MRU (External)	Arnaud Grimont		

Table of Contents

1. Description

This document will describe the Data development guidelines that should be followed by MRU Data Team and Operation Team, following the quality guidelines provided by GDC and DBA.

1.1. Purpose

The purpose of this document is to provide guidelines for future developments in order to use performance, traceability and maintenance characteristics.

New developments and modifications of current process should be following these standards.

1.2. Structure

This document will be structured following the different tasks that the Data Team works on:

- DB Objects → Tables, views, alias and triggers
- DB Code → Store Procedure and functions
- Informatica → Mappings, Sessions and Workflows
- IFB → Scripts to execute the EIM tasks
- Scripts → Other scripts (file transfer mechanisms)

2. DB Objects

The following section refers to OLAP database mainly; it is focused on the specific objects created exclusively for ETL purposes. However this may include some tables created in OLTP not to be used by the Sales applications but as a backup or parameter table.

DB Objects are expected to be equal in all environments, except for indexes that will be created based on use and performance.

2.1. Staging Database

The staging database will be used for Backup tables and temporal tables required for a specific test or data correction. The naming convention to be used for backup tables is the following:

ORIGINAL_TABLE_NAME + INC/CR # + DATE_OF_BACKUP (YYYYMMDD) + DEVELOPER_INITIALS.

E.g. WC_SALES_NONDAILY_F_UA _INC9999999_20101210_EG

The DBA team will periodically remove backup tables with a creation date (based on its name) older than 3 months. A message will be sent by DBA to all involved parties (GDC Siebel DL mainly) to inform that a delete will happen and the delete could be stopped if the GDC Siebel team confirms the backup has to stay. Further actions which would require accessing the deleted table should be done restoring the data from tape backup.

Additionally, whenever a new backup table needs to be created an excel template could be updated in SharePoint, collecting:

- Name
- Functional Reason for Creation
- Incident Related
- Team Responsible
- Person Responsible
- Date of Creation
- Status: In Use / Pending Validation / Backup
- Expected Date for Deletion → This will allow to drop tables no longer needed

The staging database will not require defragmentation process

2.2. Table Naming Convention

Table naming convention will follow the standard OBIEE guidelines.

Those guidelines do not include the non-staging tables for ETL purposes, such as:

- Parameter tables→ Name as WC_PARAM_FUNCTIONALNAME
- Delta tables → Name as WC_DELTA_ETL_FUNCTIONALNAME
- Pre step tables → Name as WC_PRE_ETL_FUNCTIONALNAME
- Log Tables → OLAP Logs will be placed on the existing WC_PROCESS_EXECUTION_LOG

2.3. Store Procedure Naming Convention

Store procedure should be named as sp_DESCRIPTIVE_NAME by default

The store procedure that executed by the ETL load will be named sp_ETL_SESSION_NAME, matching the Informatica task that is executing them, the Session Name should be a Descriptive name when possible.

The store procedure that are executed for DRP will be named sp_DRP_DESCRITIVE_NAME

If for any reason a store procedure is developed for a unique use (like initial loads, migration, data update) they should be named sp_UNI_DESCRIPTIVE_NAME, if there is a CHG/INC related sp_UNI_DESCRIPTIVE_NAME_INCNumber

For the store procedure that are developed to be executed from another store procedure, to re-use data or to provide market behaviour difference will be named following the main sp_DESCRIPTIVE_NAME_sub_REASON being this REASON the re-usable code or the market name

2.4. Description Comments

2.5. Structure

3. DB Code

The following section refers to OLAP database only

Store Procedure and functions are expected to be equal across all environments, if any difference is expected the code must be the same but commented.

3.1. Description Comments

All store Procedure and functions, must include the following comments

```
/**************************************************************************
Name: Name of the Store Procedure
Execution: Name of the process(es) that executes it
Description: Functional Description of the procedure

Release Date: Creation Date and Release (or Release Independent)
Process ID: CHG or INC
Development Team:
Author:

Versions Change History:

Number:
Description of Changes:
Release Date: Modification Date and Release (or Release Independent)
Process ID: CHG or INC
Development Team:
Author:

Number:
Description of Changes:
Release Date: Modification Date and Release (or Release Independent)
Process ID: CHG or INC
Development Team:
Author:

**************************************************************************/
```

On modification of existing store procedure the comments needs to be included, with all information available

3.2. Process Log

All OLAP store procedure will leave trace on the log table using the existing process `INSERT_WC_PROCESS_EXECUTION_LOG`

3.3. Performance Guidelines

The following guidelines have been identified in order to enhance the DB performance:

- Implement the following property within the SP
  a) At the Beginning of SP -- SET NOCOUNT ON
  b) At the End of SP -- SET NOCOUNT OFF
- Do not use the WITH RECOMPILE option
- Using the value to a variable for any repeated use of SQL functions .Ex: getdate() function.
- Do not use Temp tables, create tables on the OLAP database and include the required indexes
- Use LEFT OUTER JOIN instead of using not exists(i.e. remove the correlated sub query)
- Avoid cursors when possible → The cursor is the resource intensive in SQL Server, it makes delay(wait) the other process. In general, we try to avoid the
- Place Commit at the end of each transaction

3.4. Modularity in TRANSACT SQL

Many process written in SQL Stored Procedures have the downside that the repeat the same statements over and over or that similar processes (e.g. Automatic Objective Batch and Manually Executed Objective Batch) do not have common code for the common functionality. It is therefore mandatory to apply modularity so that functions are defined when applicable, and these functions will be reused by the different processed. Modularity has to be documented in HLFDs.

3.5. Commit control on Stored Procedures (!)

It is a must to apply commit control in every single piece of TRANSACT SQL that inserts, updates or deletes data of any sort. The commit frequency or period has to be parameterized and it should be read from a parameter table to be documented in the process HLFD.

3.6. Always use READ UNCOMMITTED mode (!)

It is a must to always use the 'WITH (NOLOCK)' clause when instantiating whatever table in a SQL query. Without using this clause, there's room for potential locks caused on base tables by queries that do not allow dirty

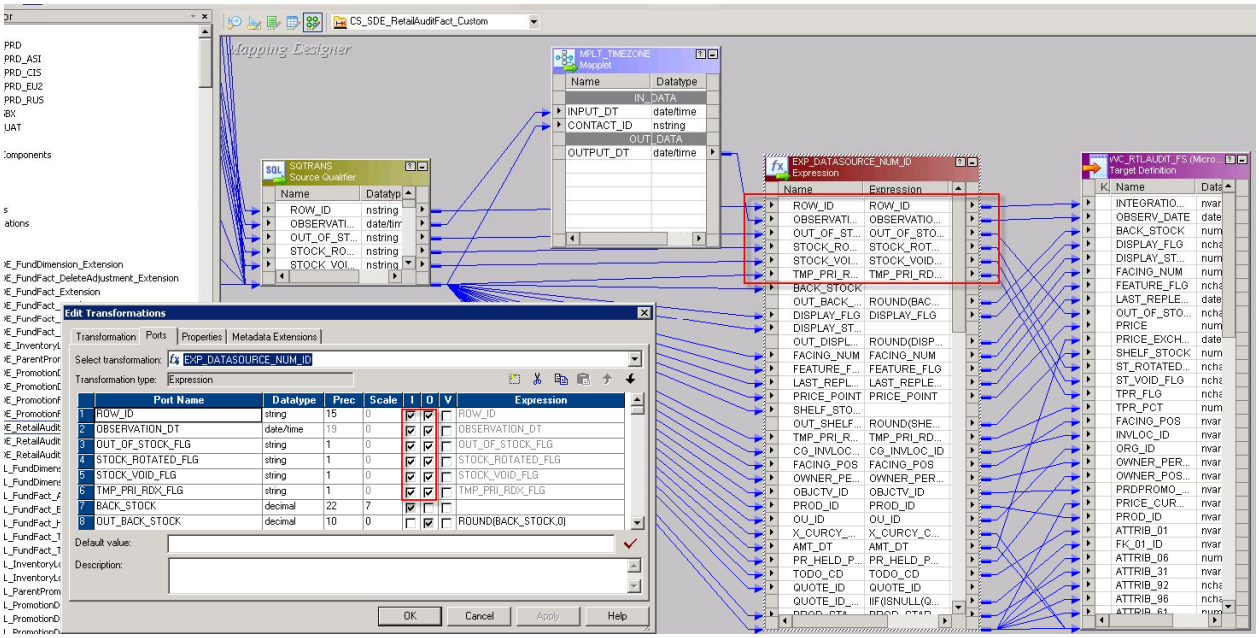
reads, and subsequently apply locks on the tables that are being read. This is especially catastrophic in EU2 OLTP, where Call Center agents suffer frequent locks that keep them unable to work for many minutes.

4. Informatica

4.1. Design considerations

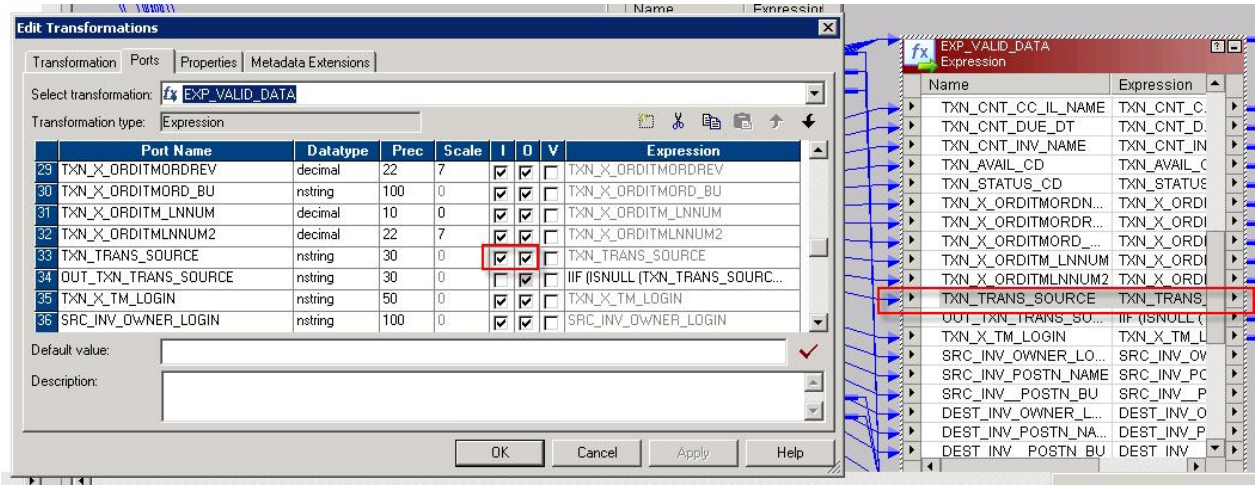
4.1.1. Performance: Avoid connecting ports that are not used in transformations

Please check the following printscreen:



It shows that e.g. ports 1 to 6 are sent from the Source Qualifier to the Expression and they are defined as I/O. All these ports do not suffer any transformation and are not used as variables either so they should never be sent to the Expression. Instead they should be sent directly to the next transformation, which in the above case is the Target Table. This shows much better performance in Informatica as greatly reduces the memory needed to execute each transformation.

It is also important that if ports are not really used as output ports, they are not marked as such in the Port properties when editing the transformation:



In the example above, TXN_TRANS_SOURCE is not used as output, but anyway is defined as output port. This increases the memory needed and therefore negatively affects performance.

4.1.2. Performance: Avoid doing unfiltered lookups on big tables and use Joiners instead or reduce the records retrieved by applying filters (!)

It is unfortunately frequent to see some lookups being done on tables which contain a lot of data (Fact tables). Moreover sometimes no filters are applied in the lookups so the lookup size is huge.

As a rule of thumb Joiners should be used instead of Lookups when accessing big tables and when the number of records moved by mapping is much smaller than the number of records retrieved by the lookup

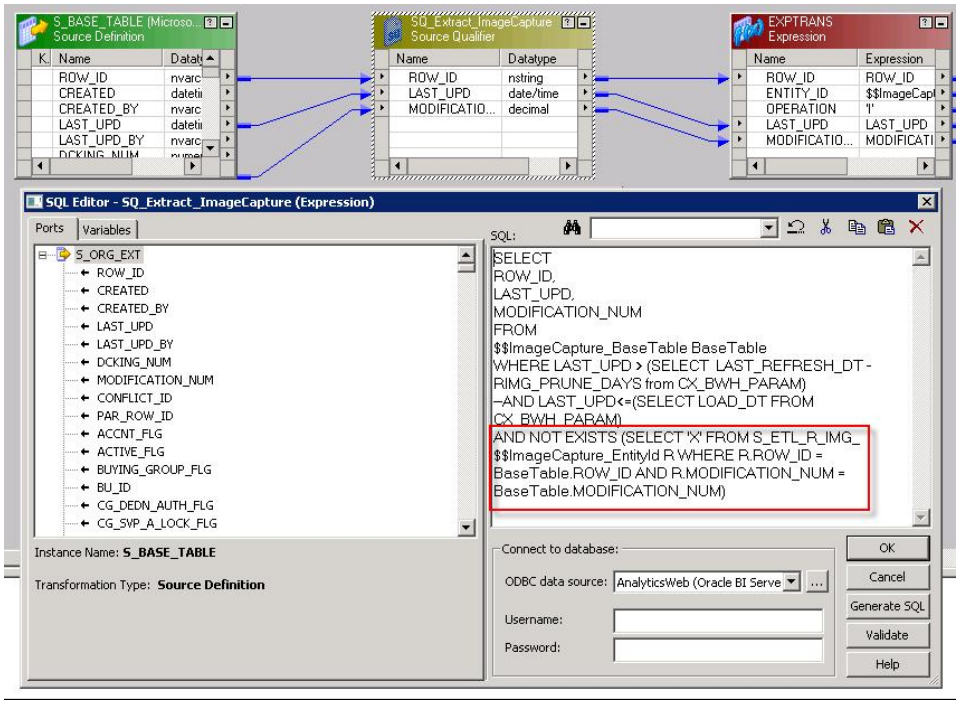
4.1.3. Performance: use disconnected Lookups when possible

Disconnected Lookups are possible only when the lookup returns one single field. Whenever possible, it is better to use disconnected lookups and invoke them from Expressions rather than placing the Lookup transformation in the mapping. This improves mapping performance.

4.1.4. Performance: avoid bottlenecks in SQ Joiners (!)

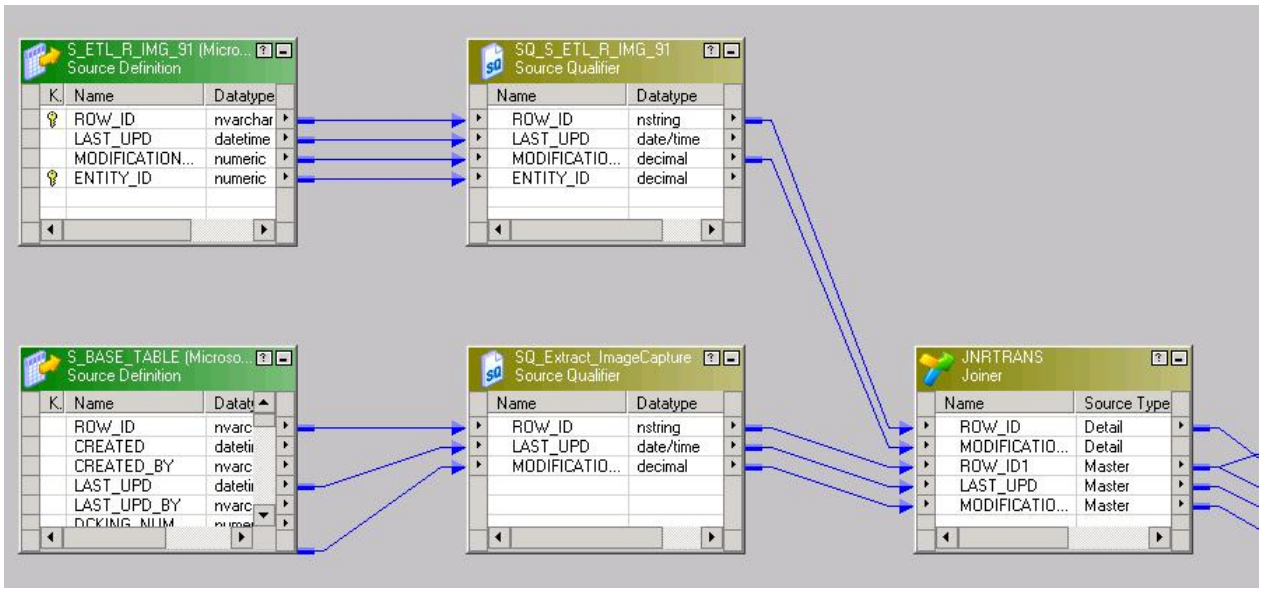
It is an extended practice to place the data extraction effort solely on the SQ Joiner. E.g. a big query like the one present in the Retail Audit Image Capture is reading from the base table joined the with the R image capture via a Subquery. Due to the size of both tables and to the fact that the RDMS will not start to return data until the query is completed on its side, the Informatica process will be idle for a long time (up to 50 minutes in PROD!).

The following printscreen depicts this:



The alternative would be to avoid giving all the effort to the DB and assign part of the work to the ETL Server. For example, using two parallel ordered extractions from the Fact and the R image capture and using a joiner the time for capturing the RA change goes from 50 to 16 minutes.

The following printscreen depicts an alternate way to capture change:



Both Source Qualifiers are ordered by ROW_ID in the same way. This way, the joiner starts processing data very soon, as soon as the queries start to return data, and as long as data enters in the Joiner, data is processed and committed in the target(s), therefore increasing the overall performance.

4.1.5. Performance: define caching parameters according to real needs (!)

In almost all cached transformations in JTI's ETLs, standard values for Data and Index Cache sizes have been used (2MB and 1MB respectively). The end result is that in practice the memory needs are much bigger and leads to creation of idx cache files. This is definitely worse for performance, since it uses slower Disk I/O instead of quicker RAM. Therefore, in DEV it should be investigated e.g. an average amount of data that will be retrieved by a lookup so that the cache parameters are set accordingly. It is not possible to find the ideal value for all situations, but definitely 2M and 1M values are not ideal at all for almost all cases.

4.1.6. Performance: use an efficient commit frequency (!)

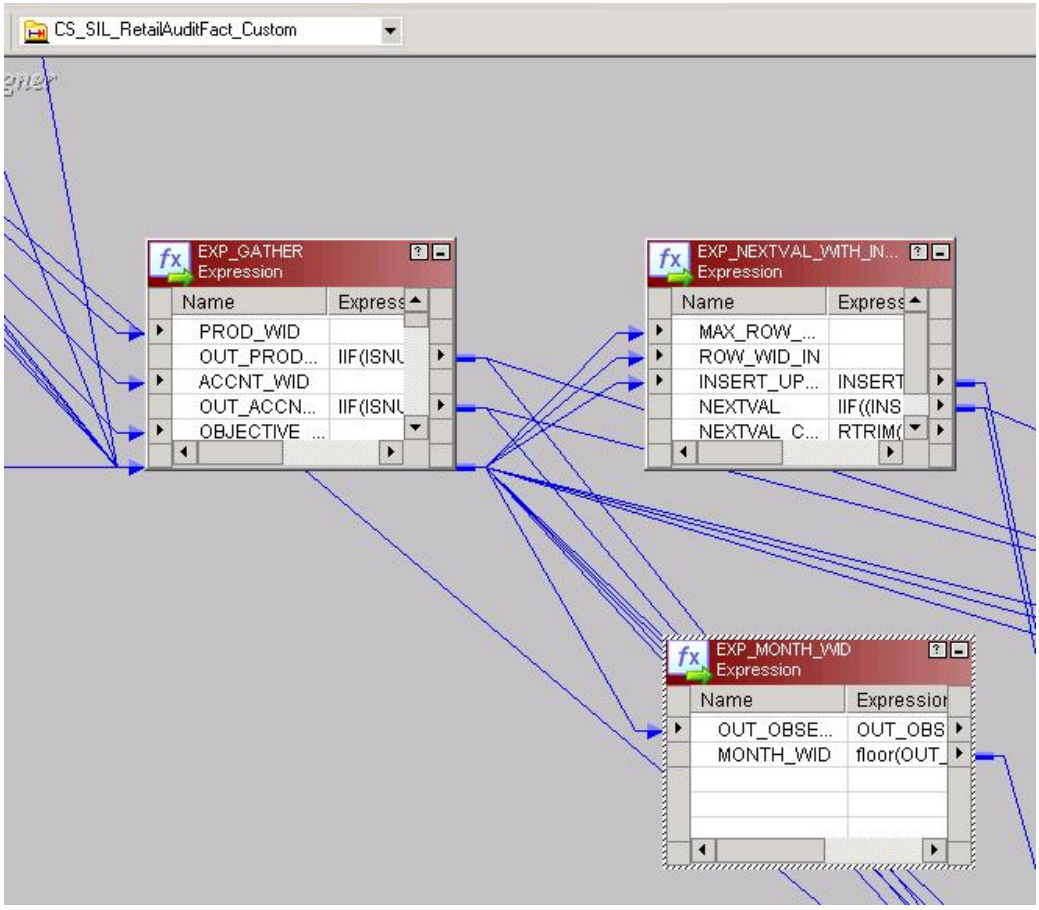
The default commit frequency of 10.000 records is far too small to be efficient when writing in OLAP Databases. It should be increased and tested in an iterative way, so that increases are not very big to avoid having issues with the usage of the temp DB. The approach will be to set in PROD the commit period to a higher value and the performance and DB Temp DB usage stability will be observed during a period of time. If OK, a higher value will be used until performance is acceptable or issues appear. The chosen value will be communicated to Data DEV for them to apply the same in DEV so that after releases the correct value is kept.

4.1.7. Performance: build small mappings, worklets and workflows

For both clarity and performance it is better to avoid building mappings with many transformations and it is preferred to build simple yet clear and well performing mappings. This is a concept similar to modularity in programming: to avoid big monolithic pieces of code by using functions or smaller pieces.

4.1.8. Performance: Avoid using N Expressions in a row and try to accomplish all in one instead (!)

Please check the below printscreen:



It is definitely possible to perform the three Expressions in a single one. This leads to less data flow that in the end improves performance.

4.1.9. Performance: try using Bulk mode for populating Staging or PRE tables

For ETL entities normally containing hundred thousands records on a daily basis (Sales Collection, Retail Audit, Activity) it is worth checking whether Bulk Mode can be used. This means that the DB mode is set to Bulk mode, so no rollbacks are possible (which does not matter for staging inserts), and inserts are much quicker. This can be combined with a big commit period (100K or bigger).

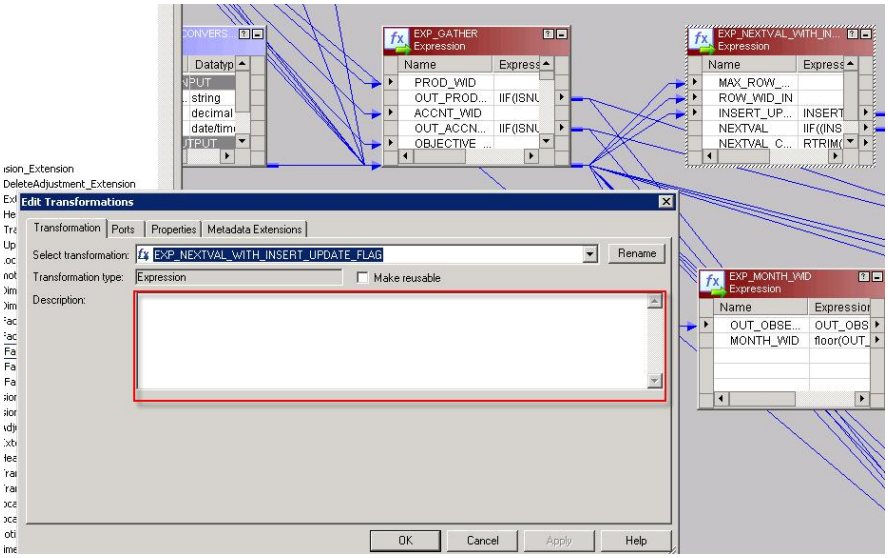
However please note that Bulk mode can't be used with ODBC connections and is only supported when using Native drivers. Please check with Tech Arch to ensure availability of Native connections.

4.1.10. Clarity: Avoid crossing connectors

When possible, connectors connecting transformation ports should not be crossing one with another so that clarity in mappings is improved.

4.1.11. Clarity: Use comments (!)

All transformations should include comments about its rationale, the CR/INC that requested its creation or modification and basic ideas about the design concepts used.



4.1.12. Stability: Avoid using the Stored Procedure Transformation (!)

It has been found that the Stored Procedure transformation in Informatica leads to poor stability and the ETL process has big chances to fail. Therefore, in case it is really needed, stored procedures will be invoked via Pre- or Post-SQL.

4.1.13. Stability: Use Native MS SQL Drivers instead of ODBC (!)

ODBC drivers are far less stable than native ones and also can have worse performance so it is advised to use Native drivers in every relational connection.

4.1.14. Config Mgmt: use parameters when possible

A lot of things can be parameterized in Informatica, from DB Connection names to entire SQL Queries. It is advised to use parameters as often as possible since this avoids the need to do further configuration in different environments and also helps to avoid making some changes release dependant (e.g. an SQ that is somehow improved to optimize explain plan can be defined as a parameter and if it has to be changed, it is enough to change the parameter file elsewhere and avoid having to change N repositories.

4.2. JTI Data Rules

4.2.1. Transfer of Boolean Flags from OLTP to OLAP

Since in Siebel Sales it is visually and even functionally equivalent when a Boolean flag is 'N' or NULL, there are many occurrences in which these fields are left NULL in OLTP. However this has a negative effect in OLAP, since in OBIEE we are able to see the real values and nulls have to be avoided.

Therefore all Boolean fields moved from OLTP to OLAP shall be defaulted to 'N' in case they are NULL.

This default has to be documented in whatever DTD and TD documents produced as part of the implementation.

4.2.2. Transfer of Boolean Flags from Interface to OLTP

Since in Siebel Sales it is visually and even functionally equivalent when a Boolean flag is 'N' or NULL, there are many occurrences in which these fields are left NULL in OLTP as the value is empty on the Interface.

Therefore all Boolean fields moved into OLTP shall be defaulted to 'N' in case they are NULL.

This default has to be documented in whatever HLFD and TD documents produced as part of the implementation.

4.2.3. Transfer of UTC Date Time fields from OLTP to OLAP

OLTP UTC Date Time fields should always be transformed to Local Time Zone based on criteria to be analyzed case by case. Data Team should request this info at Analysis Phase after assessing the type of the OLTP field, instead of not transforming it, because it systematically leads to New Requirements when markets test in OLAP.

5. Logging

5.1. Event and Error logging in Informatica

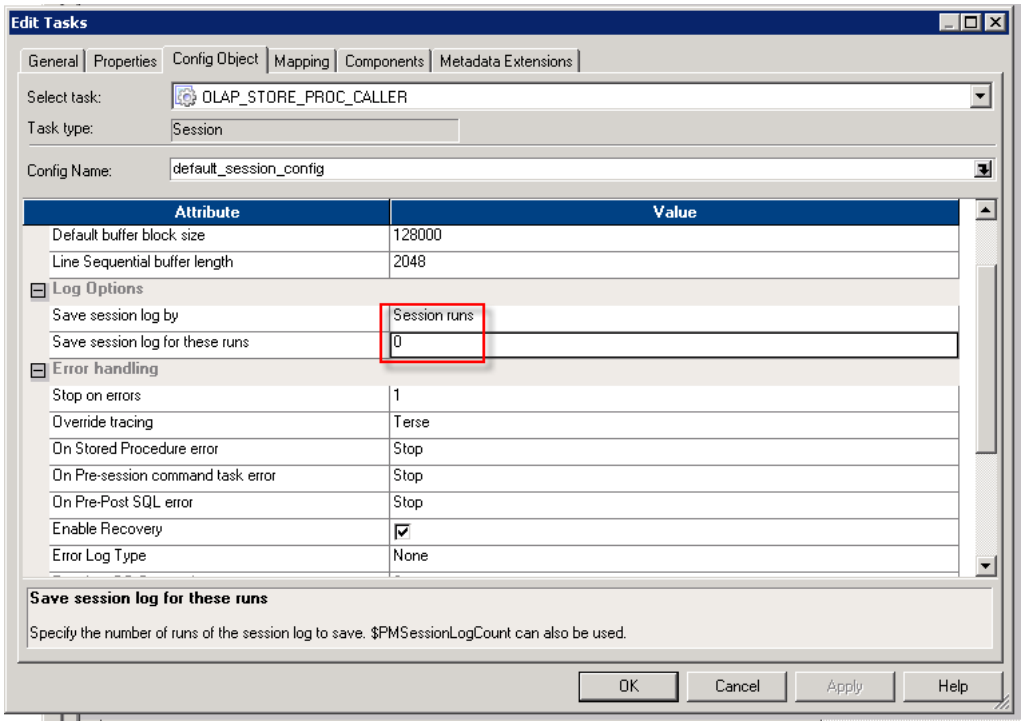
5.1.1. Log level

All PROD Sessions should have a low log level (Terse or Normal) except when a higher level is needed to trace issues (only when executed against a limited amount of data).

The same applies to UAT in general

5.1.2. Log naming and archiving

Currently JTI's ETLs overwrite the previous log files with each new run, so valuable information can be lost when trying to trace issues after some days they happen. Therefore it is suggested to change the approach of naming and archiving logs.



The above highlighted parameters define the log naming and archiving convention. Current settings imply that only one log is kept. It is advisable to change these setting so that

- Either 'Save session log for these runs' has a value of around 24 (6 ETLs per week x 4 weeks) so that we have the log history of one month time. More log space will be used than now, but Informatica controls that this is limited to a maximum amount.

- Or 'Save Session log by' has a timestamp value, but in this case it is required to have an external archiving and cleansing tool to prevent high disk usage by logs.

5.2. Event and Error logging in batch processes

It is a must to have a unique place for finding ALL the information related to the execution of a given batch and this has to be explained in the HLFD.

A unique section is needed in the HLFD to explain in which table it is possible to find log information and how to exploit that table to get appropriate information.

The table used should be as standard as possible (e.g. WC_PROCESS_EXECUTION_LOG is the standard log table for OLAP processes) so that there is a unique table that collects log information from the different processes. A similar table should be defined in OLTP to be used with the same purpose.

5.2.1. Log levels

Logging events and errors is different when a process is running in Production mode and when a process is running in testing mode. Therefore a minimum of three log levels must exist, so that it can be defined the detail of event/error information the process will generate in a table (or file when appropriate):

- Trace mode: this log level will be used when a process is being developed, tested or traced for debugging. Every single step performed by the process will be logged and ideally values of data structures being used in the process need to be included in the records. This logging mode includes:

    - General Information messages: process Start or End messages fall into this category. They can be used to trace general performance.

    - Warning messages: these messages will contain info about 'handled' exceptions, that is, they refer to abnormal process flow that is not considered an error and needs to be known for later process optimization.

    - Error messages: these messages will contain info about 'unhandled' exceptions, that is, they refer to abnormal process flow that is considered an error, causing some parts of the normal flow not to be executed successfully, and needs to be known for later process correction.

    - Simple Step information: every single step that is performed by the process will be outputted with information of variable values, INSERT/UPDATE/DELETE statements executed and also offers information useful to associate the given message with the piece of code that has generated that log entry. (Note: the *Print* statement present in many SQL process is subject to be substituted by

calls to the log function and these category of events will be logged only in case the log level of the execution instance is such that has to include this level of detail)

- Detailed Production mode: this log level will be used for stable processes running in Production (or UAT in case the process is not currently being developed or traced) and the information it will generate will be simple and concise yet powerful to know if the process is working fine. This logging mode includes:
    - General Information messages: process Start or End messages fall into this category. They can be used to trace general performance.
    - Warning messages: these messages will contain info about 'handled' exceptions, that is, they refer to abnormal process flow that is not considered an error and needs to be known for later process optimization.
    - Error messages: these messages will contain info about 'unhandled' exceptions, that is, they refer to abnormal process flow that is considered an error, causing some parts of the normal flow not to be executed successfully, and needs to be known for later process correction.
- Silent Production mode: this log level will be used for stable processes running in Production (or UAT in case the process is not currently being developed or traced) and the information it will generate will be simple and concise yet powerful to know if the process is working fine. This logging mode includes:
    - General Information messages: process Start or End messages fall into this category. They can be used to trace general performance.
    - Error messages: these messages will contain info about 'unhandled' exceptions, that is, they refer to abnormal process flow that is considered an error, causing some parts of the normal flow not to be executed successfully, and needs to be known for later process correction.

The way to accomplish this logging strategy is to define a reusable logging function or procedure that takes as input a general parameter specified on process startup to define the log level and also a parameter which defines the category of the event that is being logged in the code (i.e. for a SQL statement execution, the event category will be 'Simple Step' whereas for an error that is captured the event category will be 'Error').

General Log levels and Event Categories can be seen as grades in a severity scale:

0. Error and General Information events
1. Warning events
2. Simple steps

The 'log' procedure can be specified in a way that if the general log level of the process is smaller than the category of the event being logged, the procedure will return silently doing nothing and no entries will be logged. In case the general log level of the process is greater or equal than the category of the event being logged, the procedure will add an entry in the log table (or file when appropriate).

5.2.2. Logging is a must

As a last comment to logging, if a log function or procedure fails when trying to insert log information, the parent process invoking it has to be aborted or terminated for security.