SESUG Paper 139-2018

Data Integration Best Practices

Harry Droogendyk, Stratia Consulting Inc.

ABSTRACT

The creation of and adherence to best practices and standards can be of great advantage in the development, maintenance, and monitoring of data integration processes and jobs. Developer creativity is always valued, but it is often helpful to channel good ideas through process templates to maintain standards and enhancement productivity. Standard control tables are used to drive and record data integration activity. SAS® Data Integration Studio (or Base SAS®) and the judicious use of auto-call utility macros facilitate data integration best practices and standards. This paper walks you through those best practices and standards.

INTRODUCTION

The audacious use of “best practices” in the title of this paper, especially in the broad context of “Data Integration” demands an explanation. Since a comprehensive look at Data Integration (DI) would require a hefty tome, this paper will only touch on specific areas related to ETL standards, process automation and audit. The thoughts below come out of lessons learned over too many years of data work in multiple industries and diverse technical environments.

The paper will begin with a general overview outlining typical high-level DI stages, then progress to specific, helpful principles relating to standards and control tables that will be fleshed out in more detail. Coding standards per se will not discussed at length, but some musings in that line will appear in the details.

With the growing popularity of Data Federation and Data Virtualization products, it could be argued that DI is less important now than it has been in the past. While that argument may have some merit, standards, control and audit are equally applicable in contexts where Data Federation and/or Data Virtualization play a role.

Another challenge to the whole notion of standards, control and audit is the rapid pace of technological change. Tools, methodologies and throwaway solutions may last only several months or a few years before the next thing comes along offering bigger, better and faster. Sometimes it feels like we are always scrambling, too busy learning and trying to stay ahead of the curve to worry about standards and control. However, fast-paced environments must be managed and require discipline and control to be effective, accurate and accountable.

The material in the first two sections have a staccato “laundry list” flavor to them, but hopefully covering a number of related points in a concise manner will be helpful. Since the paper will concentrate on the technical side of Data Integration (DI), both DI and Extract, Transform and Load (ETL) will often be used interchangeably to refer to the processes involved.

DATA INTEGRATION OVERVIEW

There are a number of common elements that apply to typical DI projects, in any industry using any technology.

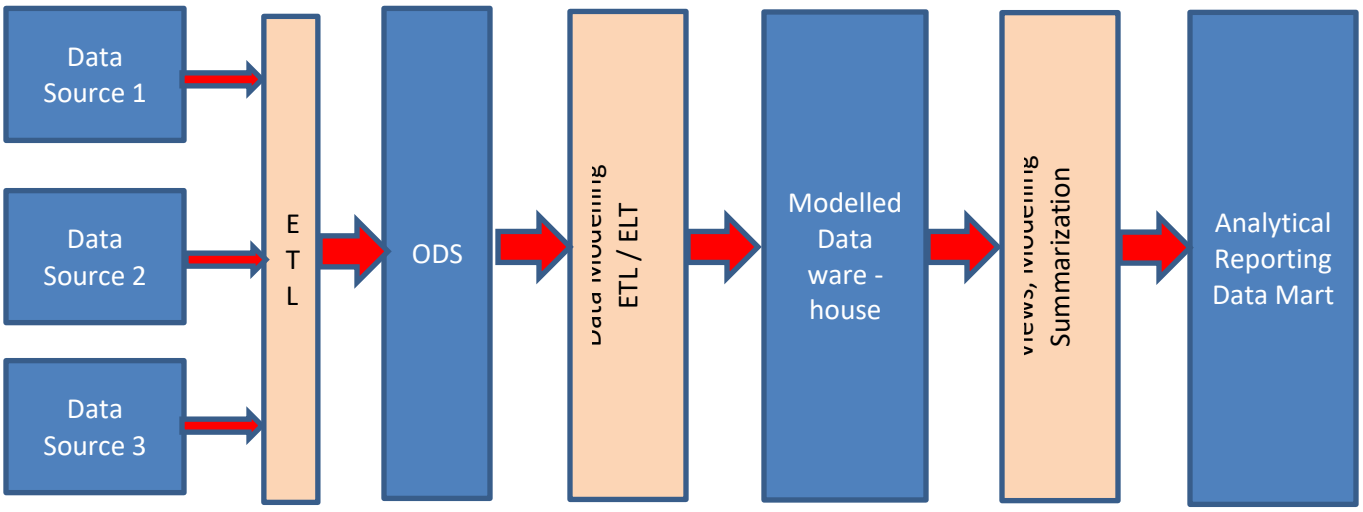

Figure 1 Typical Data Integration scenario

WHAT IS DATA INTEGRATION?

Most businesses have multiple activities that generate data in different areas, whether it be shop floor process data, accounting systems, web and social media data, or process logs. Rarely are the data for these varied subject areas stored in a single database. Transactional or operational data are most often captured in systems close to the activity, while enterprise accounting data is stored elsewhere. As a result, data is often scattered throughout an organization in various data stores. Since analytical or reporting exercises often require a holistic view of data assets found in different data "silos", it is often necessary to consolidate the required elements in a common database for convenient access. A helpful, concise definition of DI is found on the B-Eye Network website, "*Data integration is the process of combining data from a heterogeneous set of data stores to create one unified view of all that data*."

As the quantity of data increases, the need for robust DI processes increases as well.

DATA SOURCES

It is important that data sources be accurate, timely and reliable – or at least two out of the three. ☺ It may be that cleansed, operational data is available from an existing data mart, but the additional latency means *your* data mart will now be two days behind. If latency is an issue, extracting directly from the transactional database provides more timely data as long as any data cleansing issues can be handled effectively.

Knowledge of the source data environment is vital. The manner in which source data is loaded and processed will influence how DI processes are created. Ideally, the ETL jobs will perform delta extracts, only pulling data added or updated since the last extract was executed. A data dictionary and access to source database metadata will provide the information required to build efficient delta extract processes. To ensure delta queries run efficiently against source databases, take advantage of the functionality specific to each database vendor, e.g. Oracle hints, Teradata partitioned primary indexes.

Extracting data via direct access to the database tables is usually preferred. However, it is not always possible to convince database owners to allow remote access to their tables. Sometimes data suppliers

will opt to provide flat file extracts instead, requiring ETL developers to brush up on INFILE / INPUT processing to ingest the data. Flat file extracts can be pulled or pushed via secure FTP.

Where possible do not rely on manually updated sources. Excel spreadsheets are particularly troublesome since non-IT users do not always understand the importance of maintaining a consistent format and are unware that moving or renaming columns breaks the DI process.

Data sources are not always tabular or structured. Big data is playing an increasingly role, bringing order to unstructured data. Web services, streaming data and other non-traditional data source are becoming more common and present unique challenges.

OPERATIONAL DATA STORE (ODS) LOADING

In many cases, source data is stored initially in ODS tables in the target database environment. Most often ODS tables mimic the source data table and column structure. Some flexibility is warranted to make structure changes to account for database differences, e.g. Oracle stores dates in TIMESTAMP format, lop off the unnecessary time component and store target data as a DATA field, omit columns that are always null etc.

Though the target ODS environment is not modeled, it is still important to know the characteristics of the source data to process it accurately and efficiently. If daily delta extracts are to be loaded into the ODS layer, source data triggers or events must be available to ensure source data readiness before ODS jobs begin. Efficient delta extract criteria must be defined, whether it be last modified timestamps or ID columns that increase over time. If the source table does not have suitable columns to establish *reliable* delta criteria, it may necessitate reading the entire source table, truncating the ODS table and loading all rows each time the job runs. Source primary key components must be identified to ensure update / insert (upsert) operations into the ODS layer are accurate and eliminate opportunity for duplicate data. If the source database physically deletes data rows, extracts must leverage source database change data capture (CDC) logs, or schedule periodic recasts to true-up the ODS layer.

The target ODS tables must include ETL audit columns to record load events and load / update timestamps. Downstream operations will use the ODS timestamps to perform delta extracts when loading the modeled environment tables.

MODELED DATA ENVIRONMENT LOADING

While the source database itself may be a modeled environment, the source's data model may not be suitable for the target data store. This is especially true when multiple data sources are being integrated to populate the target database. The intended use of the target database will dictate the type of data model required. If the target database is an Enterprise Data Warehouse the model will likely be highly normalized. On the other hand, if a reporting data mart is being loaded, a different data model would likely be more suitable. **Do not minimize the importance of a robust data model!** If the data model is deficient, ETL development will be more difficult and data accuracy and maintenance will suffer.

Naming standards must be developed and adhered to. The SAS 32 character limitation for table and column names is a frustration in a world where database vendors are allowing much longer names. Naming standards can be a contentious issue, but once settled must be consistently applied to all names in the target database environment. It is equally important to generate a comprehensive data dictionary to ensure the business and technical definitions for all data items are available.

Often the modeled environment is loaded via delta extracts from the ODS layer. To ensure data accuracy, these delta extracts must use the ODS layer's load / update timestamps, and not any data value from the original source. Using ODS timestamps for delta operations will ensure source data that arrives "out of order" will still be processed correctly into the modeled environment. Modeled tables also have ETL audit columns to record load events and load / update timestamps.

ANALYTICAL / REPORTING ENVIRONMENT

Normalized data models optimized for a data warehouse may not be suitable for end user consumption.

Analytical use cases and reporting processes may demand different perspectives and views on the data, whether it be denormalization, summarization or transposition. These different perspectives are introduced in the analytical / reporting environment through further processing or via integrated views built on the modeled data.

HELPFUL PRINCIPLES

FOCUS FUNCTIONALITY

For program development and maintenance efficiency, it is helpful to focus each program or job on only one DI task. Monolithic code edifices may look impressive, but they make for longer development timelines, increased maintenance effort, complicate testing processes and hamper the ability to easily rerun failed jobs. While SAS has some object oriented functionality (and DS2 offering more in that line) the goal is not a completely modular architecture – it just isn't practical with the SAS language. User-written macros and functions can be utilized to increase modularity and centralize common functionality.

How focused should each job be? In the ODS layer, each job should populate one ODS table, most often from one source table. The programs loading the modeled environment tables should load a single modeled table. The delta criteria for each job is generally focused on the data most germane to the target table's contents.

Related single-task programs are then strung together in a flow and run together. If Billing data is being extracted from source, the initial flow will contain the source to ODS jobs. A second flow groups jobs loading the modeled Billing tables from the ODS layer. It is generally helpful to group jobs in this manner to determine when logical units of work are complete, e.g. the daily Billing extract load to ODS has completed, or the modeled Billing tables have been loaded today. Grouping or integrating in this way makes it far easier to build data readiness triggers for downstream processes.

TOOLS AND TECHNIQUES

SAS Data Integration Studio (DIS) has long been the tool of choice for data integration projects. In environments were DIS it not available, Foundation SAS is used with great success as well.

Whatever tool is selected, the features of the tool should be utilized when possible *and practical*. DIS has a large number of pre-built code transformations (GUI interfaces to capture options and generate SAS code) that simplify job development that is less prone to coding errors. Rather than developing and debugging custom macro code to loop over functionality in an ETL job, use the native LOOP transformation found in DIS and be assured robust code will be generated with almost no developer effort. There is no need to memorize the vagaries of specific SAS syntax to load a Teradata table, use the Teradata Table Loader transformation. Future DIS version updates are less painless when the tool is used in this manner.

On the other hand, some DIS transformations add an unhelpful layer between the developer and the finished task. In those cases, it is often more effective to use a user-written code node instead. The JOIN transformation falls into this category, especially when the query becomes more complex and non-SAS SQL functionality must be employed, e.g. database windowing or OLAP functions. In some cases, we do *not* want to use the tool's functionality.

Sensible standards, documented best practices, program templates and a well-stocked autocall macro library can help elevate the skill level and productivity of the development group. The gentle confines of a well-defined environment can bring new hires up to speed quickly. Rather than coding and developing to the lowest common denominator, it is usually possible to raise the skill level and throughput of the development team. Maintenance is less arduous when there is a familiarity with the architecture of the job templates used for development. Unit testing phases are often shorter when established autocall macros are used for common components of the ETL job, e.g. upserts, control table updates etc.

Where possible, create parameterized, data-driven processes, that use control tables and the application data itself to run themselves. Avoid hard-coding where possible. Rather than hard-coding userids, passwords, server IP addresses into programs, use a parameterized autocall macro to establish

connections. If SAS Management Console is available, use authdomains rather than userids and passwords, even in the autocall macro.

Do not be a technological bigot. We love SAS and its extensive functionality, but sometimes an explicit pass-through query utilizing native database SQL is a better solution than pulling the data down from the database into the SAS environment for processing. Use the strengths of the each tool in combination rather than forcing square pegs into round holes.

CONTROL TABLES

Data Integration activity and the associated control data must be recorded in a structured, accessible format for several reasons:

Control

Efficient delta extract processes will only retrieve new or updated rows of data. In those scenarios, record counts and the maximum delta column last-modified timestamp value (or identifier number) must be captured and stored in an extract control table at job completion. When the next delta job runs, the first step in that routine retrieves the saved maximum value of the source delta column from the extract control table, and then uses that value to extract only source rows that have a greater delta column value, i.e. last modified timestamp or identifier number.

Audit

Invariably it will be necessary to report on ETL job activity for formal audit procedures or to investigate data anomalies. The status of specific, individual DI processes, run dates and times, completion return codes, record counts and extraction criteria are very important historical information. Counts of rows inserted or updated, error and exception counts, job start and end times, elapsed times, job failures, reruns, data recasts – these items provide a good overview of the DI process, useful for reporting and performance tuning.

While ETL process data is captured in the control tables, each row in the ETL's target table must also record control data for auditing purposes. Each target table contains the following columns:

SYS_LOAD_DTM	– original load timestamp
SYS_LOAD_EXT_SEQ_ID	– latest CNTL_EXTRACT entry to load/update row
SYS_UPDATE_DTM	– latest update timestamp

Automation

As audit and control data is captured, opportunities for automation and finer control become more numerous. Conditional job processing, data quality checks, and sophisticated scheduling decisions can be driven from the control data captured by each ETL process.

The physical data model diagram on the next page illustrates the relationships of the various control tables. Some of those tables are integral to the capturing of ETL activity, others more concerned with data quality and SAS metadata relationships.

While the tables will be discussed under the three headings below for convenience, some tables could easily appear under more than one heading.

1. ETL control
2. Data quality and automation
3. History, audit and metadata

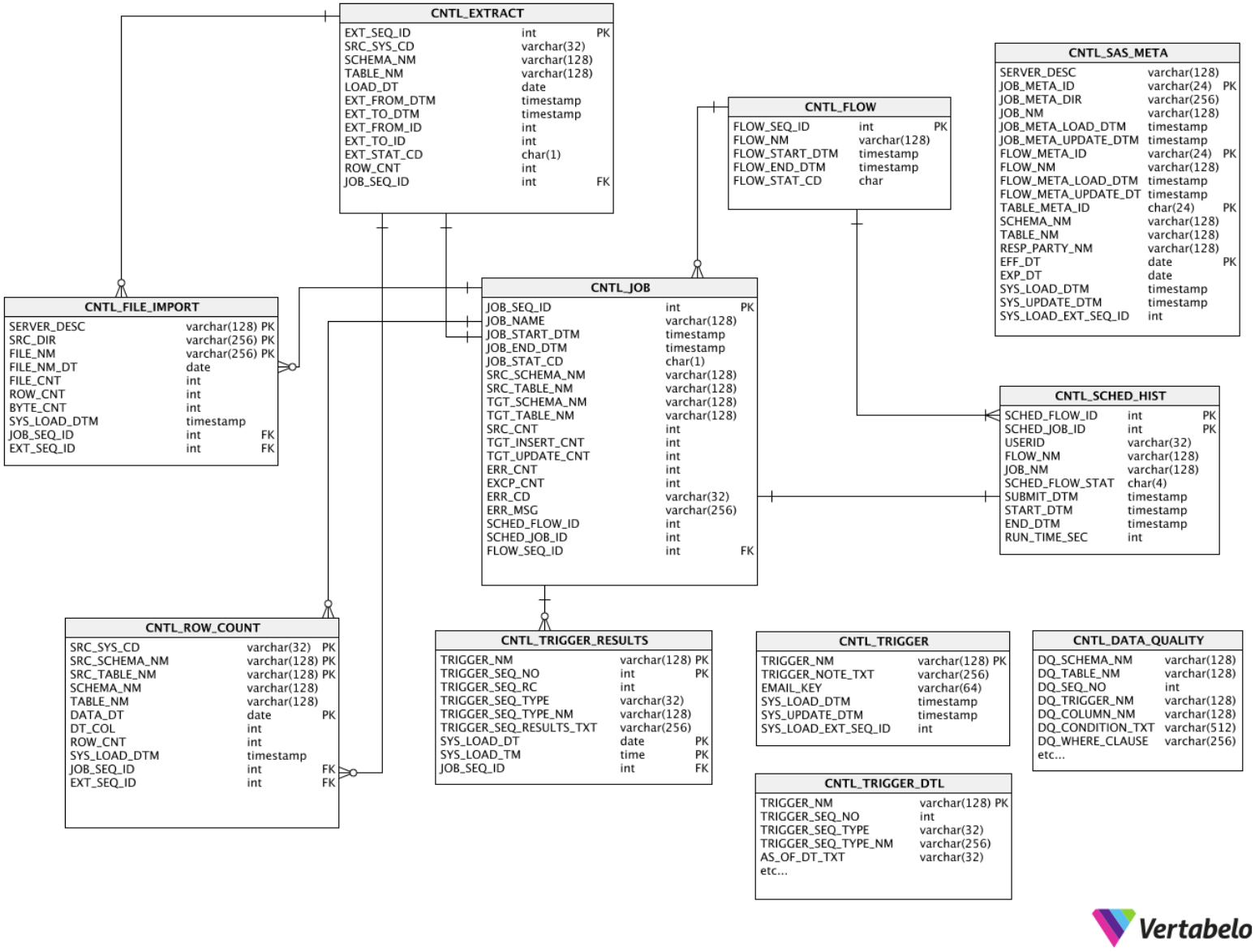

Figure 2 Control Tables - Physical Data Model Diagram

ETL CONTROL

CNTL_EXTRACT

Each ETL process creates an entry in the CNTL_EXTRACT table, eg:

- loading data from an SFTP site
- performing an incremental delta extract from a source database to ODS
- truncate and load
- populating modeled tables
- building analytical or reporting tables.

While the data model supports a 1:M relationship between CNTL_JOB and CNTL_EXTRACT, typically a 1:1 relation is maintained, one job to one extract, in keeping with the Focus Functionality principles.

Column	Description
EXT_SEQ_ID	auto-generated ID
SRC_SYS_CD	source system mnemonic
SCHEMA_NM	schema or directory name
TABLE_NM	table or file name
EXT_FROM_*	source extracted where last mod timestamp or ID > EXT_FROM_*
EXT_TO_*	source extracted where last mod timestamp or ID <= EXT_TO_*
EXT_STAT_CD	extract status, O=Open, C=Closed, F=Fail
ROW_CNT	source rows extracted
JOB_SEQ_ID	foreign key to CNTL_JOB

CNTL_JOB

ETL jobs create an entry in the CNTL_JOB table each time they run. Jobs executed in a flow will also populate the foreign key column FLOW_SEQ_ID and the SCHED_* columns retrieved from environmental variables established by the batch scheduler.

Column	Description
JOB_SEQ_ID	auto-generated ID
JOB_NM	job name
JOB_START_DTM	job start timestamp
JOB_END_DTM	job end timestamp
JOB_STAT_CD	job status, , O=Open, C=Closed, F=Fail
SRC_* columns	source data definition
TGT_* columns	target data definition
*_CNT columns	various counts, rows extracted from source, inserted / updated into target, exceptions, errors
ERR_*	error code and error message, recorded for job failures
SCHED_*	batch scheduler flow and job ID
FLOW_SEQ_ID	foreign key to CNTL_FLOW, associates jobs to flows

CNTL_FLOW

A job flow is means of grouping and running related jobs together. The first job in a flow is typically a trigger job to ensure data readiness, followed by an "open flow" job, the ETL or report job(s) and a "close flow" job. A row is created in CNTL_FLOW when the job flow is "opened". Each subsequent job in that flow will retrieve the opened flow's FLOW_SEQ_ID, storing it in CNTL_JOB, thus logically associating the each job with the flow. When all jobs in the flow have successfully completed the "close flow" job runs.

Column	Description
FLOW_SEQ_ID	auto-generated ID
FLOW _NM	job flow name
FLOW _START_DTM	flow start timestamp
FLOW _END_DTM	flow end timestamp
FLOW _STAT_CD	flow status, O=Open, C=Closed, F=Fail

Autocall macros are called to open and close CNTL_FLOW, CNTL_JOB and CNTL_EXTRACT entries.

DATA QUALITY AND AUTOMATION

CNTL_ROW_COUNT

While CNTL_EXTRACT and CNTL_JOB capture row counts applicable to a particular ETL run, data quality checks often require more specific detail. Transactional data generally has a transaction date, e.g. Orders have an order date, Interactive Voice Response (IVR) telephone systems have a call date. While seasonality may affect transaction volumes, activity will most often follow a pattern or a trend. If the IVR system records 100K calls on a typical business day, we should expect roughly that many calls each business day. If the source system hiccupped and reposted half of yesterday's calls but only 5K of today's calls we should probably not consider our IVR data up-to-date and useable. Data quality verification is more concerned with data rows per "data date", e.g. call date, than the general count of rows returned by today's extract.

Similar data quality requirements might apply to slowly changing dimension tables like Customer Master. On a typical business day 5% of our Customer Master data may change, triggering a row with a new effective date in the SCD2 table. If that drops to 0.1% one day, we may have a data issue, either at source or our ETL process.

Column	Description
SRC_SYS_CD	source system mnemonic
SRC_SCHEMA_NM	source system schema name
SRC_TABLE_NM	source system table name
SCHEMA_NM	our schema name
TABLE_NM	our table name
DATA_DT	transaction date, e.g. IVR call date value
DT_COL	transaction date column name, e.g. IVR CALL_DT
ROW_CNT	row count for this DT_COL's DATA_DT value for this EXT_SEQ_ID
SYS_LOAD_DTM	timestamp when CNTL_ROW_COUNT row loaded

JOB_SEQ_ID	foreign key to CNTL_JOB for extract job
EXT_SEQ_ID	foreign key to CNTL_EXTRACT for extract

CNTL_DATA_QUALITY Tables

Data integrity is the most important characteristic of any successful Data Integration project. The data warehouse can contain a host of subject areas, each with a comprehensive set of dimensional and fact data. However, if the data is not accurate, what good is it? Unfortunately, data accuracy is surprisingly easy to compromise. Software is written, or at least designed, by fallible humans and hardware can fail – stuff happens!

It is impossible to verify every data point – time and budget limitations simply do not allow for it. However, the most critical of data items should be examined and approved before important downstream processes consume them. Row counts by transaction data date are one component of that examination, but often more complex verifications are required. For example, if the percentage of rows with NULL activation date exceeds a certain value, it is indicative of a data anomaly. Or, weekly data that hasn't been refreshed by Tuesday is out of date. More complex verifications are driven by a set of data quality tables. CNTL_DATA_QUALITY defines the checks to be performed, other tables capture the various data quality metrics such as minimum, maximum values, NULL vs populated percentages etc…

Column	**Description**
DQ_SCHEMA_NM	schema name
DQ_TABLE_NM	table name
DQ_SEQ_NO	sequence number
DQ_TRIGGER_NM	trigger to be satisfied before data quality check is performed
DQ_COLUMN_NM	column to be verified
DQ_CONDITION_TXT	SQL code to generate more complex verifications
DQ_WHERE_CLAUSE_TXT	when specific subsetting is required, e.g. for a single line of business
etc…	

Business-defined data quality requirements are imported from developer-maintained Excel spreadsheets and loaded into the CNTL_DATA_QUALITY table. SAS autocall macros perform the actual data quality checks and update the data quality metric control tables.

CNTL_TRIGGER Tables

Having captured ETL control data and data quality metrics, it is now possible to regulate downstream processes more effectively. As discussed in the Focus Functionality section, similar processes are grouped together into flows, e.g. Billing source to ODS jobs are in one flow, Billing ODS to modeled environment jobs are a second flow. To maintain data integrity in the modeled environment, it is vital that *all* the ODS tables required by the modeled environment jobs are up-to-date before the flow loading the modeled environment starts. In similar fashion, reporting jobs and analytical processes should not run with incomplete or inaccurate modeled data.

In the Billing data example, before the modeled environment flow initiates, we want to ensure the Billing source to ODS flow is complete. In similar fashion, since the modeled Billing data also requires Customer Master data elements, that flow requires that specific Customer dimension jobs must also have finished

successfully. In the reporting realm, the Sales dashboard should not be updated unless the sales data is up-to-date and accurate.

CNTL_TRIGGER_DTL defines the requirements for dependent processes. In addition to the basic columns listed below, the control table includes many more columns defining the thresholds for more complex checks, such as ROW_COUNT and DQ_VALIDATION checks, e.g. AVERAGE_10 for rolling average of last 10 entries, +/- percentage / count allowances etc.

<table>
<tr><th>Column</th><th>Description</th></tr>
<tr><td>TRIGGER_NM</td><td>trigger name, standards require it to be the same as the dependent job flow</td></tr>
<tr><td>TRIGGER_SEQ_NO</td><td>number trigger conditions</td></tr>
<tr><td>TRIGGER_SEQ_TYPE</td><td>define the type of check to be performed, e.g.
<table>
<tr><th>Value</th><th>Use case</th></tr>
<tr><td>FLOW_COMPLETE</td><td>successful completion of entire flow</td></tr>
<tr><td>JOB_COMPLETE</td><td>successful completion of single job</td></tr>
<tr><td>FILE_DROP</td><td>presence of X files in specified directory with specified file name pattern</td></tr>
<tr><td>ROW_COUNT</td><td>row count check, e.g. rolling average of last 10 days</td></tr>
<tr><td>DQ_VALIDATION</td><td>data quality check</td></tr>
</table></td></tr>
<tr><td>TRIGGER_SEQ_TYPE_NM</td><td>value pertaining to TRIGGER_SEQ_TYPE, e.g. one condition of the modeled Billing flow requires that the Billing ODS flow be complete, TRIGGER_SEQ_TYPE = ”FLOW_COMPLETE” TRIGGER_SEQ_TYPE_NM = “BILLING_SOURCE_TO_ODS”, i.e. the name of the Billing source to ODS job flow</td></tr>
<tr><td>AS_OF_DATE_TXT</td><td>text describing data recency requirements, e.g. ‘CURRENT_DATE-1’, ‘MONTH_END’, translated into an actual date when the trigger is checked</td></tr>
<tr><td>etc…</td><td></td></tr>
</table>

ETL HISTORY AND JOB METADATA

CNTL_SCHED_HIST

Batch scheduler log data is extracted through the day and loaded into CNTL_SCHED_HIST. While some job execution data is available in CNTL_JOB, the scheduler data provides additional information that is useful for audit purposes and reporting. The CNTL_JOB.SCHED* columns are parsed from batch scheduler environmental variables at job run time and stored in CNTL_JOB to facilitate table joins to provide a comprehensive view of the ETL process.

Column	Description
SCHED_FLOW_ID	sequential flow ID assigned by batch scheduler
SCHED_JOB_ID	sequential job ID assigned by batch scheduler
USER_ID	job’s functional user id
FLOW_NM	job flow name
JOB_NM	job name

SCHED_FLOW_STAT	status of job, DONE, EXIT, RUNNING
SUBMIT_DTM	timestamp of job submission
START_DTM	actual start timestamp of the job
END_DTM	timestamp when job ended (successfully or unsuccessfully)
RUN_TIME_SEC	job execution time

CNTL_SAS_META

A daily process extracts flow, job and table data from the SAS metadata to populate the SCD2 table CNTL_SAS_META. SAS metadata has a number of uses, not the least of which is the "responsible party", aka the last person to change the job – that comes in very handy when jobs fail. ☺ See the PDM diagram in Figure 1 for the columns in this table, all of which are self-explanatory for those familiar with SAS metadata.

FLOW AND JOB TEMPLATES

FLOW TEMPLATES

Job flows are made up of related ETL processes that run together, e.g. Billing ODS data to modeled environment jobs. Typically, the first job in the flow is a trigger job to check that all the required data is ready for consumption before the real work begins. Each time the trigger process runs, the pass / fail / warning outcome is recorded in CNTL_TRIGGER_RESULTS for audit purposes. If the trigger check fails, the trigger job aborts, then invoked again in X minutes by the batch scheduler to check again. When the required data is ready, the trigger job completes normally and the flow continues.

After the trigger job completes successfully the next job opens the CNTL_FLOW entry. Subsequent jobs run in series or parallel as required to complete the flow requirements. When all ETL / reporting jobs in the flow have completed, the final job closes the CNTL_FLOW entry and the flow completes successfully.

Since shop standards require the trigger name and the flow name to be the same value, generic trigger and flow open and close jobs can be included in every flow. These generic jobs reference the batch scheduler environmental variables to retrieve the flow name and execute autocall macros to check trigger conditions and open / close flows.

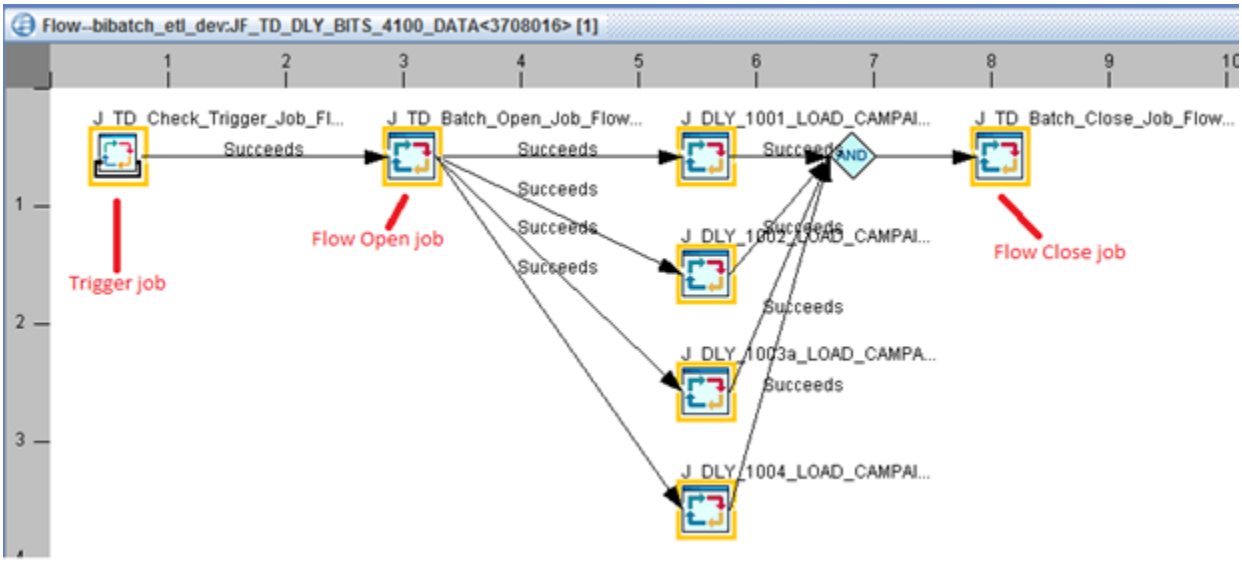

Figure 3 Job Flow Diagram

JOB TEMPLATES

Developer creativity is a wonderful thing, but not at the expense of compromised standards. To enhance developer productivity and reduce future maintenance effort, ETL jobs are built from standard templates, each designed for a specific application.

- flat file import, trunc and load
- source to ODS delta extract
- ODS to modeled tables

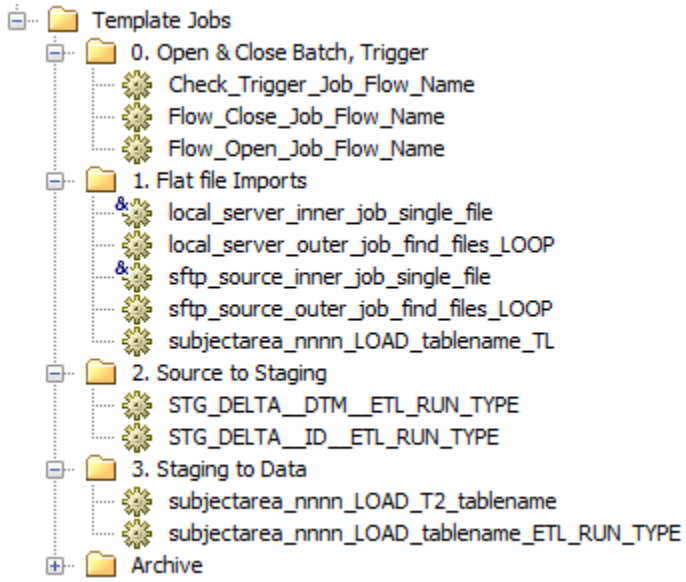

Figure 4 Job Templates

While each of the job templates differ according to application, common elements in each ensure standards are maintained and control tables are properly maintained. Each job template ensures:

- job pre-code establishes job parameters and initializes ETL control table entries:
    - defines database connections, SAS libnames via autocall macro
    - sets macro variables to define source, staging and target schema / tables and ETL type: delta or initial load
    - calls autocall macros:
        - %getCurrentFlow to obtain FLOW_SEQ_ID for this flow
        - %jobOpen to create a CNTL_JOB entry
        - %extOpen to create a CNTL_EXTRACT entry
- subsequent job transformations / job code uses macro variables assigned in job pre-code to avoid hard-coding
- common ETL activities are performed using autocall macros
    - table truncation, error handling, upsert / SCD2 functionality, row counts
    - %extClose to update extract max timestamp, row counts at job completion
- autocall macros are invoked as necessary to update CNTL_ROW_COUNT, CNTL_FILE_IMPORT and populate data quality metrics
- job post-code calls %jobClose to update CNTL_JOB with job ending statistics

Example job pre-code, %extClose and job post-code may be found in Appendix 1. Job palette screen shot below for Source to ODS job template:

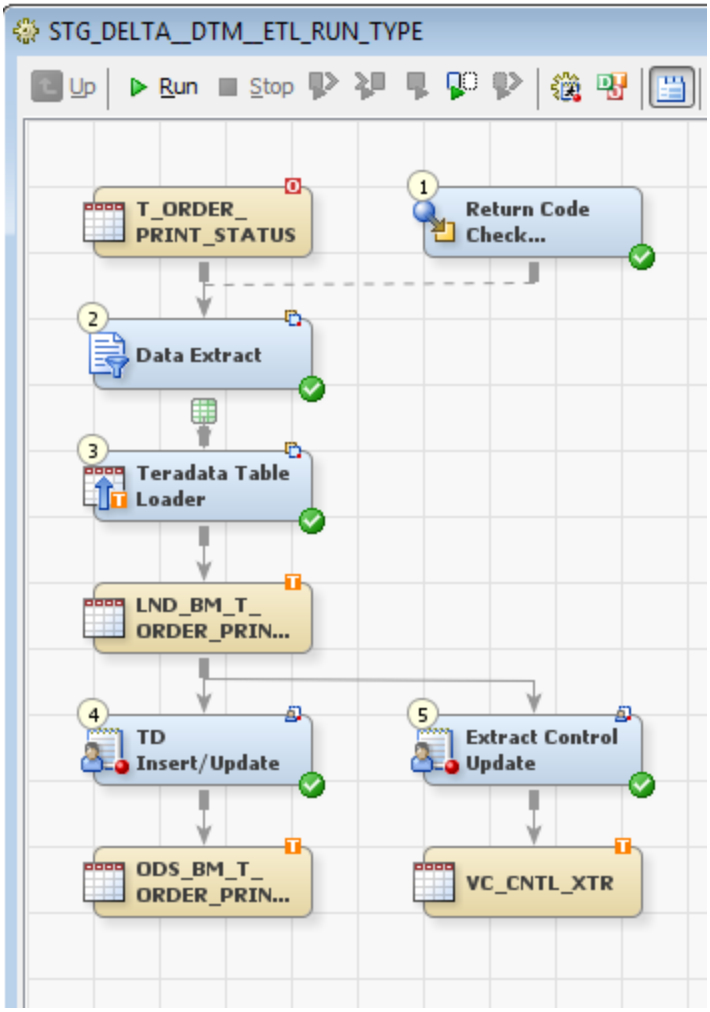

Figure 5 Source to ODS Job Template

CONCLUSION

KISS, keep it simple & standardized. ☺ The effort to think through and design basic standards and reliable control structures pays dividends through the *entire* software development lifecycle. Job templates help enforce coding standards and provide a robust skeleton upon which to build, ensuring control tables are properly updated to provide for appropriate audit, automation and data quality checks. Gains from increased productivity and reduced maintenance effort along with the advantages of audit and automation functionality make the upfront investment extremely worthwhile.

Beyond the obvious examples, control tables have repeatedly proven themselves useful in a number of scenarios, a sampling below:

- identifying the initial developer of the job
- identifying jobs that use / load specific tables
- self-serve dashboards to show data readiness and quality for end users
- isolate data gaps
- identifying when data anomalies began to occur
- isolating Teradata fastload errors to a specific host on the grid
- daily reports of batch exceptions, e.g. Teradata fastload errors, job failures

REFERENCES

Rupinder Dhillon, Darryl Prebble. 2015-04 "Automating Your Metadata Reporting", *Proceedings of the SAS Global 2015 Conference*, Dallas, TX: Available at https://support.sas.com/resources/papers/proceedings15/2060-2015.pdf.

B-Eye Network. 2010. "Clearly Defining Data Virtualization, Data Federation, and Data Integration" Accessed March 5, 2018. http://www.b-eye-network.com/view/14815

"Creating and Scheduling SAS Job Flows with the Schedule Manager Plugin in SAS Management Console", https://communities.sas.com/t5/Ask-the-Expert/Creating-and-Scheduling-SAS-Job-Flows-with-the-Schedule-Manager/ta-p/399190?attachment-id=12523

Anjan Matlapudi, Amar Patel and Travis Going, "Let the Schedule Manager Take Care of Scheduling Jobs in SAS® Management Console 9.4", *Proceedings of the SAS Global Forum 2016 Conference*, Las Vegas, NV: Available at: http://support.sas.com/resources/papers/proceedings16/11202-2016.pdf

ACKNOWLEDGMENTS

I have worked with a number of very talented folks through the years, each of whom have contributed to the formation of my thoughts on this topic. Most of them are probably blissfully unaware of the positive impact they have been.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author via the website contact:

Harry Droogendyk
Stratia Consulting Inc.
www.stratia.ca

APPENDIX 1

JOB PRE-CODE

```
%db_connect(WAC MITSTG);                    %* assign DB connections ;

%global _etl_run_type;
%macro get_etl_run_type;                    %*INITIAL or DELTA run ;

      %if %sysfunc(sysexist(ETL_RUN_TYPE)) %then
            %let _etl_run_type = %sysget(ETL_RUN_TYPE);
      %if &_etl_run_type eq %then %let _etl_run_type = DELTA;

%mend;

%get_etl_run_type

%put Executing with %nrstr(&_etl_run_type)=&_etl_run_type..;

%let _batchSeqId=0;
%getCurrentFlow             /* batch name = JF name, no parm reqd */

/* job audit input parameters     */
%let _batchSeq      = &_batchSeqId;
%let _jobName       = &etls_jobname;
%let _loadDT        = "&sysdate"d;
%let _procStg       = ODS;

%let _srcSys        = WAC;
%let _srcSchema     = BMWACM;
%let _srcTable      = T_ORDER_PRINT_STATUS;

%Let _dtmType       = 0;

%let _stgSchema     = MI_STAGE;
%let _stgLib        = MITSTAGE;
%let _stgTable      = LND_BM_T_ORDER_PRINT_STAT;

%let _tgtSchema     = MI_STAGE;
%let _tgtLib        = MITSTAGE;
%let _tgtTable      = ODS_BM_T_ORDER_PRINT_STAT;

%let _errLib        = MITSTAGE;
%let _errTable      = &_stgTable._E;

%let _srcRecCnt     = .;
%let _tgtInsRecCnt  = .;
%let _tgtUpdRecCnt  = .;
%let _errRecCnt     = .;
%let _excRecCnt     = .;
%let _errCD         = ;
%let _errMsg        = ;
```

```
/* job audit output parameters    */
%let _jobSeqId      =;
%let _jobStartDTM   =;
%let _jobEndDTM     =;
%let _jobStatusCD   =;

%jobOpen(
      batch_seq_id        = &_batchSeq,       /* batch sequence id             */
      job_name            = &_jobName,        /* job name                      */
      proc_stage          = &_procStg,        /* process stage                 */
      load_dt             = &_loadDT,         /* business load date            */
      src_sys             = &_srcSys,         /* source system                 */
      src_schema_name     = &_srcSchema,      /* source schema name            */
      src_file_table_name = &_srcTable,       /* source file or table name     */
      tgt_schema_name     = &_tgtSchema,      /* target schema name            */
      tgt_file_table_name = &_tgtTable        /* target file or table name     */
      );

%put NOTE: Job instance ID [&_jobSeqId] for job [&_jobName].;
%put NOTE: Job start date time [%sysfunc(putn(&_jobStartDTM, datetime20.))].;
%put NOTE: Job status [&_jobStatusCD].;

/* extract control */
%let _seqId         =;
%let _xtrFromDTM    =;
%let _xtrToDTM      =;
%let _schema        = &_srcSchema;
%let _table         = &_srcTable;

%extOpen(     schema        =&_schema,
              table         =&_table,
              srcSys        =&_srcSys,
              jobSeq        =&_jobSeqId,
              loadDT        =&_loadDT,
              dtmType       =&_dtmType
      );

%macro do_initial_dates;          %* If INITIAL run, adjust dates;
      %if &_etl_run_type = INITIAL %then %let _xtrFromDTM  = 2000-01-01 00:00:00;
%mend;

%do_initial_dates

%put NOTE: Extracting for extract [&_seqId] with dtmType of &_dtmType.;
%put NOTE: Extract from date time [&_xtrFromDTM] to [&_xtrToDTM].;
```

EXTRACT CLOSE CODE NODE

```
%let _rowCount          =0;
%let _maxXtrDTM         =;
%let _dtmTypeNumFormat = %eval(20 + &_dtmType).&_dtmType;
              %* results in 20.0 for &_dtmType=0, 26.6 for &_dtmType=6;

proc sql noprint;

       connect to teradata ( &bi_db_connect_string );

       select coalesce(max_date, "&_xtrFromDTM"DT) format &_dtmTypeNumFormat.
       ,      row_count
         into :_maxXtrDTM, :_rowCount
         from connection to teradata (
              select max(CREATE_TIME) as max_dtm,
                     count(*) as row_count
                from &_stgSchema..&_stgTable
              );
quit;


%extClose(    xtr_seq_id    =&_seqId,
              schema        =&_schema,
              table         =&_table,
              srcSys        =&_srcSys,
              closeDTM      =&_maxXtrDTM,
              rowCount      =&_rowCount,
              dtmType       =&_dtmType
       );

%let _tgtInsRecCnt  = &_rowCount;       /* target insert cnt for %jobClose      */
```

JOB POST-CODE

```
%jobClose(
       job_seq_id          = &_jobSeqId,       /* job sequence                  */
       src_record_cnt      = &_srcRecCnt,      /* source record count           */
       tgt_ins_record_cnt  = &_tgtInsRecCnt,   /* target insert record count    */
       tgt_upd_record_cnt  = &_tgtUpdRecCnt,   /* target update record count    */
       err_record_cnt      = &_errRecCnt,      /* error record count            */
       exc_record_cnt      = &_excRecCnt,      /* exception record count        */
       err_cd              = &_errCD,          /* error code                    */
       err_msg             = &_errMsg          /* error message                 */
       );

%put NOTE: Job instance ID [&_jobSeqId] terminating at
       [%sysfunc(putn(&_jobEndDTM, datetime20.))].;
%put NOTE: Job terminating with error code [&_errCD].;
%put NOTE: Job terminating with error message [&_errMsg].;
```