Paper 771-2017

From Event Queues to Analytics

Bronwen Fairbairn, Collection House Limited

ABSTRACT

In the quest for valuable analytics, access to business data through message queues provides near real-time access to the entire data life cycle. This in turn enables our analytical models to perform accurately. What does the item a user temporarily put in the shopping basket indicate, and what can be done to motivate the user? How do you recover the user who has now unsubscribed, given that the user had previously unsubscribed and re-subscribed quickly? User behavior can be captured completely and efficiently using a message queue, which causes minimal load on production systems and allows for distributed environments.

There are some technical issues encountered when attempting to populate a data warehouse using events from a message queue. This presentation outlines a solution to the following issues: the message queue connection, how to ensure that messages aren't lost in transit, and how to efficiently process messages with SAS® software; message definition and metadata, how to react to changes in message structure; data architecture, which data architecture is appropriate for storing message data and other business data; late arrival of messages, how late arriving data can be loaded into slowly changing dimensions; and analytical processing, how transactional message data can be reformatted for analytical modelling.

Ultimately, using an Enterprise Service Bus to populate a data warehouse can require less development than accessing source databases; however a robust architecture is required.

INTRODUCTION

Data warehouses have two primary functions, they are:

- an auditable record of business data, they are an ideal tool for reconciliation of business systems and as a final check to ensure business standards are being adhered too; and
- responsible for providing any reporting or analysis that isn't easily exposed by the source systems themselves.

Currently, most data warehouses periodically snapshot the state of the business information from source systems. However, this means that information can be lost if multiple changes take place between snapshots and means the business can't use the analysis from the data warehouse to react in real time. Data warehouses can report on the past and hypothesize about the future, but they are generally incapable of incorporating current activity into either reporting or analysis.

The ability for the data warehouse to perform both these functions can be assisted by capturing data after every transaction or event and pushing information to the data warehouse for immediate loading. This is exactly what event queues give us the power to do, to push information after every significant event through to the data warehouse. Event queues enable the data warehouse to be more complete and deliver greater business value.

Dan Jahn of the SAS Institute espouses 6 tenets that will enable the data warehouse to facilitate better decision making; they are "(1) Aligned with your mission; (2) Complete; (3) Faster; (4) Accurate; (5) Accessible; and (6) Recurring, ongoing, or productionalized". Message Queue data feeds will assist your data warehouse to be more:

- comprehensive (2), as the data warehouse should reuse messages consumed by other system components and messages will be updated to reflect current business complexity;
- have less latency (3), messages can be transmitted with the initial transaction request or on transaction completion, this allows reporting latency to be minimized and the data warehouse to send out real time alerts;
- be more accurate (4), message queues allow all events to be loaded into the data warehouse and there should be no data lost from successive transactions; and
- be constantly updating and productionalized (6), once again the core nature of the messages mean that any business changes should be quickly reflected in the messages transmitted.

This paper will discuss some of the technical and architectural issues encountered when establishing a message queue data warehouse and outline possible approaches. The techniques outlined can be implemented independently, but that is not within the scope of this paper.

WHAT ARE EVENT QUEUES?

Event Queues, which are delivered via an ESB (Enterprise Service Bus), are software-engineering components used for inter-process communication and will often be implemented as architectural best practice within an organizations information technology systems. Messages and the ESB are becoming a favored means of communication, in effect the API, for systems everywhere. Event messages can be published to the ESB and any other systems can listen to and react to the events in a timely, but unrestrictive, fashion. They allow systems to communicate asynchronously; meaning that systems can operate independently, they can respond to requests in their own timeframe and with complete control over tasks.

Messages are generally transmitted using JSON (JavaScript Object Notation) which is a lightweight data format and places minimal overhead or constraints on the messages or the message queues. JSON messages have structure but minimal types or metadata instead relying on both the publisher and the receiver of the message to correctly understand the message content. A message contract generally exists to define and enforce the message structure.

Below is a typical example of a JSON message. This message is notification that a new transaction account has been created; this account has one account holder and one account signatory:

```
{
"destinationAddress":"rabbitmq://153.85.31.13/Message:AccountCreatedEvent",
"headers":{
  "Logging.Sent":"2017-02-16 05:50:23 +10:00",
  "Logging.RequestId": "2b34c601-77d8-460d-854f-3753fc99a405"
},
"message":{
  "accountId":2319,
  "accountType":"Transaction",
  "initialBalance":50,
  "accountHolders":[
    {"customerNumber":"4987","customerRole":"Owner"},
    {"customerNumber":"5032","customerRole":"Signatory"}
  ],
  "createdBy":"SusanD",
  "activityId":"a6af2f5d-c078-464c-bcb8-45fb3b26ef84"
},
"messageType":["urn:message:Message:AccountCreatedEvent"],
"sourceAddress":"rabbitmq://153.85.31.13/Account_Service"
}
```

An analogy is to imagine all the individual systems in your technical environment as different staff. A message queue allows all the staff to communicate; via commands, requests for other staff to do tasks;

and events, notifications for other staff that tasks have been done; all the staff have independent control over the databases they use, the way they optimize their work, they can even outsource responsibilities or have brief absences with minimal impact on the organisation as a whole. The only requirements are that every system correctly communicates via the service bus and that it completes its tasks in a reliable and timely manner.

But this architectural ideal breaks if the data warehouse is connecting to each individual systems database. Why isn't the data warehouse using the API that every other system has standardized on? The architectural justification for message queues can be applied to the data warehouse just as validly as any other system if not more.

Service Buses are particularly valuable to the data warehouse for several reasons:

- They enable data to be pushed to the data warehouse. This reduces the latency of data in the data warehouse and improves analytics and reporting.
- The data warehouse is a business wide reporting tool. It isn't interested in source system detail which can be distracting and complicate the data cleansing process.
- Historical information cannot be lost or overwritten, once a message is published to the ESB it will received by the data warehouse. The data warehouse will be an auditable, accurate record of business history, enabling its capacity for accurately reporting and modeling.
- The data warehouse system can access information even earlier and report on issues within the technical environment by processing command messages as well as events.
- Organizations that use message queues will often advocate that the message queue is the guaranteed record of history, rather than individual source systems, but extracting information from a message queue archive can be lengthy and complicated. Combining data warehouses with the message queue gives businesses the ability to easily report on and analyze message queue data.

DATA WAREHOUSE ARCHITECTURE

Processing messages requires a significant paradigm shift from the traditional snapshot methods of loading a data warehouse. Currently data warehouses include anything that uses a simple ETL (Extract Transform Load) process to extract data from source databases, transform it as per reporting requirements and load it into tables optimized for reporting. However, as messages are discreet transactions, they are of minimal benefit independently for reporting or analytics. Messages need to be amalgamated in the data warehouse and ESB data warehouses will generally require a more complex architecture, including:

- A staging area for messages before they have been processed;
- A comprehensive data warehouse to store all source data processed so far; and
- Reporting marts, optimized as per the current business requirements.

Using this architecture, ESB data warehouses:

1. Receive data in the form of messages (or data extracts where messages are unavailable);
2. Append or insert that data into the data warehouse;
3. Transform data from the data warehouse into the format required for reporting; and finally
4. Load the cleansed information into data marts for reporting and analysis.

Figure 1 is a representation of what this architecture might look like and some technologies used at each level.

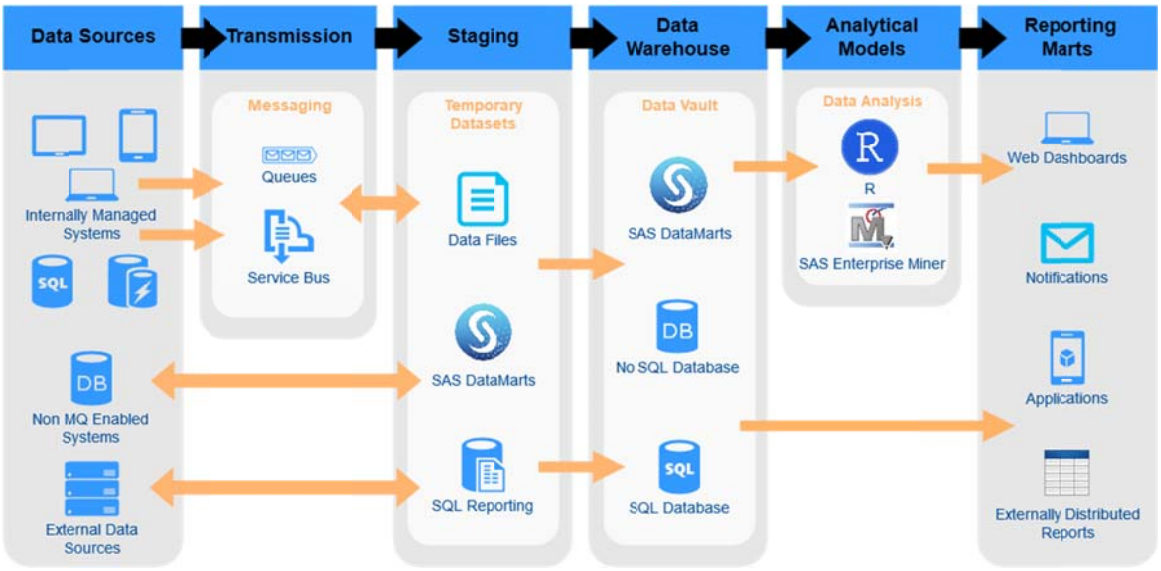

Figure 1. Message Queue Connection Process

This approach allows the data warehouse to build a comprehensive and auditable collation of an organization's data, while still delivering fast and effective reporting specific to the business definitions at the time (via the reporting marts).

MESSAGE QUEUE CONNECTION

The practice of processing ESB data, as opposed to snapshotting source systems has the potential to reduce processing of unchanged data, and disperse the processing load throughout the day. When an event occurs, the significant information is packaged into a message and sent to the data warehouse, which can process that small update quickly and provide near real time reporting to the users.

SAS Applications across the board are currently expanding their capability for connecting to ESBs and processing message data, although support for specific messaging technologies will depend on your environment. As shown in Figure 2, SAS® Data Integration Studio currently includes support for MicrosoftMQ and WebSphereMQ, while other SAS tools also support RabbitMQ, TIBCO Rendezvous, ActiveMQ and other JMS ESBs.

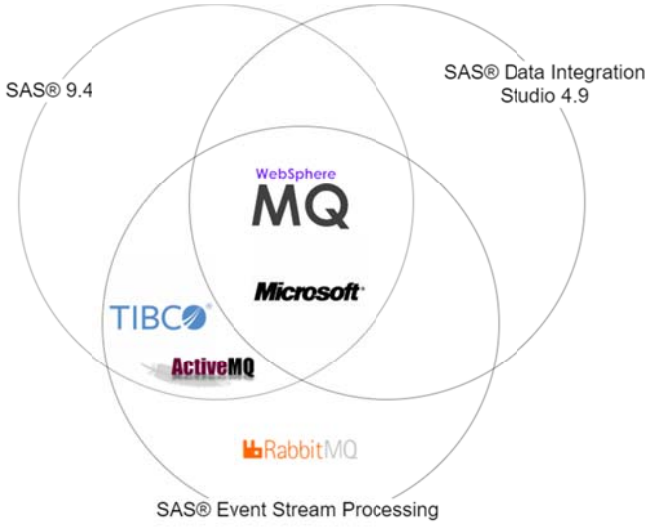

Figure 2. Support from SAS Applications for Message Queues

The standard architecture for most ESB systems is first-in, first-out; the message queues are a list of commands or events. An application retrieves a message off the queue, which prevents the message being accessed by any other process, processes the message and then removes the message from the queue. SAS® Event Stream Processing can process millions of events per second. However, many SAS environments don't have that capacity and will process data more efficiently in batches of similar messages. It is also likely that the priority of messages for the data warehouse is complex; depending on the type of message, the number of similar messages and its age; which would be best managed in SAS. Correctly prioritizing a set of messages, perhaps five Account Created messages up to 30 seconds old versus two Email Changed messages 10 seconds old and 1 Campaign Created message 42 seconds old, and executing the optimal load processes can be flexibly handled within a SAS program itself.

One solution is to use two message queues for the data warehouse and implement a five way handshake (Send, Acknowledge off 1st Queue, Acknowledgement Successful, Acknowledge off 2nd Queue, Acknowledgement Successful) as depicted in Figure 3. The traditional ESB approach described above uses a three way handshake.

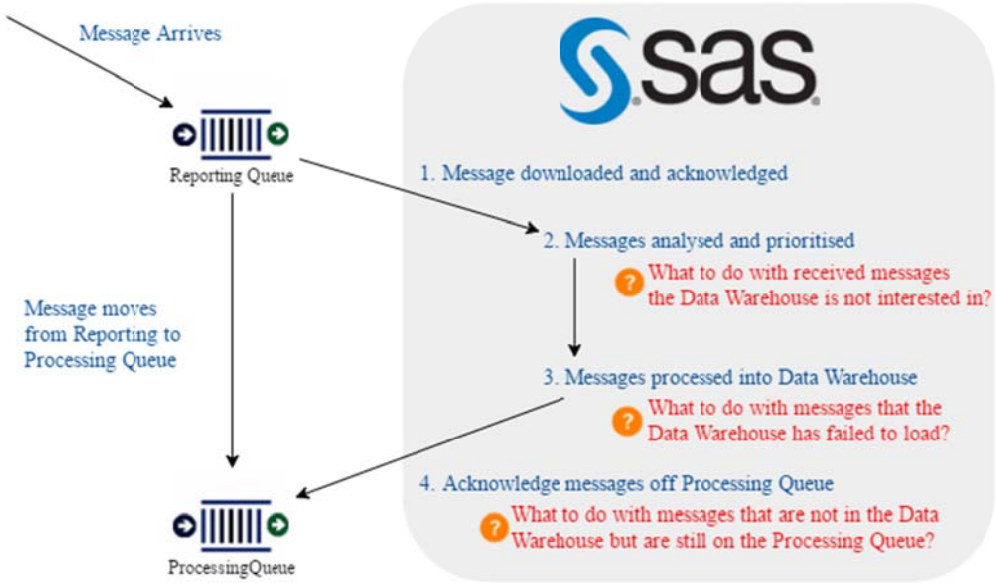

Figure 3 Diagram of Process

Under this approach:

1. One process is responsible for connecting to the message queue, retrieving the messages, acknowledging them from the reporting queue and saving them to the SAS Server. This process may also be responsible for deserialising the JSON;
2. Another process is responsible for looking at the available messages, prioritizing them and executing the appropriate SAS jobs for those messages;
3. Several processes exist for the actual processing of message data, this will generally be one process per message type; and
4. A final process is responsible for identifying the processed messages and removing them from the processing queue.

Under this approach the ESB is more than just a method of reliably transmitting information. It is a record of outstanding work and gives all systems complete visibility over the reporting backlog via the processing queue. Any failures to import messages will leave those messages on the processing queue and could place additional load on the messaging system. Therefore, it is essential that the processes that receive messages from the ESB and append them to the data warehouse be as robust as possible, with minimal delays and no data loss. As long as that objective is achievable the additional visibility is advantageous.

MESSAGE DEFINITION AND METADATA

Message queues generally use JSON to transmit event information. This is because JSON is a light weight and minimalist format for relaying information. Message structure is already well understood by both the transmitter and the receiver, and messages contracts exist to define and enforce that structure.

This presents several issues for the accuracy and auditability of data warehouse. Using the message connection architecture previously outlined, messages could be delayed and message contracts can change before the data warehouse processes that message. Additionally messages can include extensible features such as sub objects and arrays that are not easily defined using SAS Metadata; even string lengths, which the message contract may not specify, can easily cause truncation of data when imported into SAS.

These issues can cause data skipping or truncation at rates unacceptable for a reliable data warehouse. However, it is possible for the data warehouse to adapt to content of messages and automatically extend its data structure to prevent data loss.

Using Base SAS functionality messages can be imported into SAS and the structure checked prior to the data being loaded into the data warehouse. Using the message queue connection outlined previously, the message receiver is responsible for downloading messages and reformatting them so they can easily be imported into SAS. This process is entirely content agnostic, it may have no requirements other than that the message is well formatted JSON. The individual processes would be responsible for importing the message data files, analyzing the content, comparing the message structure to that already in the data warehouse and loading the data if possible.

Using SAS Proc Import with the maximum value guessing rows value will allow maximum flexibility in the initial SAS import, with the only restriction being that the imported data file is less than 32,767 characters. Alternatively, as of SAS® 9.4 M4 the JSON Libname Engine can be used to automatically parse a JSON file into SAS Datasets, however this loses the ability to auto format dates, as dates and times are not a recognized type in JSON. Regardless of your import method, Proc Contents and Data Steps can then be used to compare the import dataset with the data warehouse, and if possible extend the data warehouse tables so that the new data can be loaded without any data loss.

For example, this process imports a datafile and appends the data to a SAS table that already exists:

```
proc import datafile="XXX.dat" out=Input dbms=csv;
  guessingrows=max;
run;

proc contents data=Input out=Input_Cont(keep=varnum name type format length
formatl formatd) noprint;
run;

proc contents data=DW_Table out=DW_Cont(keep=varnum name type format length
formatl formatd) noprint;
run;

data DW_Cont;
  format fmt $50.;
  merge DW_Cont(rename=(varnum=cur_varnum type=cur_type format=cur_format
length=cur_length formatl=cur_formatl formatd=cur_formatd)) Input_Cont;
  by name;

  if type ne cur_type and type ne . and cur_type ne . then do;
    error 'ERROR: Variables in the message can not change type
automatically. Variable: ' name;
    abort;
  end;
  else do;
```

```
    length=max(length,cur_length);
    formatl=max(formatl,cur_formatl);
    formatd=max(formatd,cur_formatd);
  end;

  if type=1 or cur_type=1 then do;
    if formatl=0 then fmt=cats(format,'.');
    else fmt=cats(format,formatl,'.',formatd);
  end;
  else fmt=cats('$',length,'.');
run;

data _null_;
  format data_format $500.;
  retain data_format;
  set DW_Cont  end=last;
  if _N_=1 then data_format='';
  data_format=catx(' ',data_format,name,fmt);
  if last=1 then call symput('format',strip(data_format));
run;

data DW_Table;
  format &format;
  set DW_Table input;
run;
```

This code analyses the structure of the datafile and the SAS table and expands the SAS table to accommodate the incoming data if possible. Significant issues, such as a variable having a different type on the datafile and the SAS table, require manual intervention and throw an ERROR. The data step that generates the DW_Cont table will likely require modification if the DW_Table is not a SAS dataset, but this process can be adapted for most databases and environments.

This loosely defined process allows the data warehouse to automatically and efficiently adapt to changes in source data; it significantly reduces the risk of data loss and need for process rework. New fields should not automatically be exposed, in the reporting layer or to the end business users, until they have been reviewed. However, they will be recorded within the EDW (Enterprise Data Warehouse), greatly improving the EDW's status as an auditable system and enabling rapid development of reports using those fields at a later date.

MESSAGE CONTENT

As shown the data warehouse can adapt to the message structures used elsewhere in the organization and as much a possible messages should be shared and consumed by multiple systems to minimize variation. However, data warehouses are typically state based systems: for efficiency and reliability reasons they focus on recording the state of data over time, whereas preexisting messages may only capture the information that has changed or how it has changed.

Transactional or delta messages which are often used for source systems that are focused on the current state of data and can be of significant benefit when responsibilities are migrated to other systems, greatly complicate the task of building a data warehouse with an accurate record of history and particularly processing late arriving data, which will be discussed more in detail below.

For example, imagine the following messages about account balances are received and are loaded into an SCD (Slowly Changing Dimensions) table. The pre-existing messages here report the change in balance in the AccountBalanceChanged_Event message, so that components listening to that message can react differently if the balance change was positive or negative, large or small.

Account	Balance Change	Date
4987	$10	1Jun2014
4987	-$5	5Jul2014
4987	-$5	16Aug2014

Table 1. Tabular representation of Account Balance Changed Event messages off the Message Queue

SQN	Account	Balance Change	DTS	EDTS
1	4987	$10	1Jun2014	1Jun2014
2	4987	-$5	5Jul2014	.

Table 2. SCD Table that these messages could generate in the Data Warehouse

These messages and the resulting data warehouse table have several issues:

- despite the fact that there were two transactions withdrawing $5, that is not obvious in the data warehouse;
- it is not obvious what the balance at any point in time is; and
- calculating the balance at any point in time will produce errors if even one message is missed.

If the intention is to load transactions into the data warehouse the Balance Change field would be appropriate, as the state of the transaction is that it adds $10 or subtracts -$5. However, placing the actual account balance on the account table and its related messages, serves to make them more reliable and efficient. In this example two messages may be required, a Transaction_Event that has the transaction amounts on it and an AccountBalanceChanged_Event that reports the current account balances.

Messages used by the data warehouse will generally be events and as such should record the state of the information after a change has taken place, indeed the message should include the state of all information that could have been edited in this type of transaction, including the date of when the transaction took place.

For example, if a bank customer goes online to change their address, but only changes the ZIP code field that was entered erroneously (previously it was 00724).

Customer_ID	ZIP_Code
4987	07039

Table 3. Tabular representation of Account Balance Changed Event messages off the Message Queue

SQN	Customer_ID	Address	ZIP_Code	State	DTS	EDTS
1	4987	13 Garden St, Livingston	07039	NJ	18Feb2017	.

Table 4. SCD Table that these messages could generate in the Data Warehouse

This event is therefore responsible for reporting on the state of the information in the customer address system.

If the data warehouse was also interested on reporting on the success of address change attempts and the interface allowed the ZIP code field to be updated without editing or reviewing the state and address fields we could also have an message that resembled this:

Customer_ID	ZIP_Code	Date

Customer_ID	ZIP_Code	Date
4987	07039	18Feb2017

Table 5. SCD Table that these messages could generate in the Data Warehouse

This would be an appropriate message for the data warehouse if the address changed request on the 18Feb changed the ZIP code, but neither confirmed or corrected the address and the state fields.

Although this may require some message redevelopment initially, changes like having an effective date on all messages will be of benefit to the message queue as a whole.

DATA WAREHOUSE MODEL

A comprehensive data architecture such as Data Vault, originally developed by Dan Linstedt, is worthwhile to combine extensive message data together into accessible data warehouse. Data vault modeling is a database modeling method that is designed to provide long-term historical storage of data coming in from multiple operational systems. A Data Vault data warehouse is primarily focused on accurate and extensible data storage, rather than reporting efficiencies. As such it has the potential to combine all an organizations data into a comprehensive and cohesive data repository.

One advantages of using messages to populate a data warehouse is that the message data is intra-system; as such messages should use business level keys and reflects business functions as opposed to system complexity. Event data should be far easier to amalgamate into an EDW architecture, such as the Data Vault. Messages even serve to group attributes into change frequencies and will server to improve the efficiency of SCD in the EDW.

For example, a company using messages to load information on customer campaigning could build a data warehouse that resembles Figure 4.

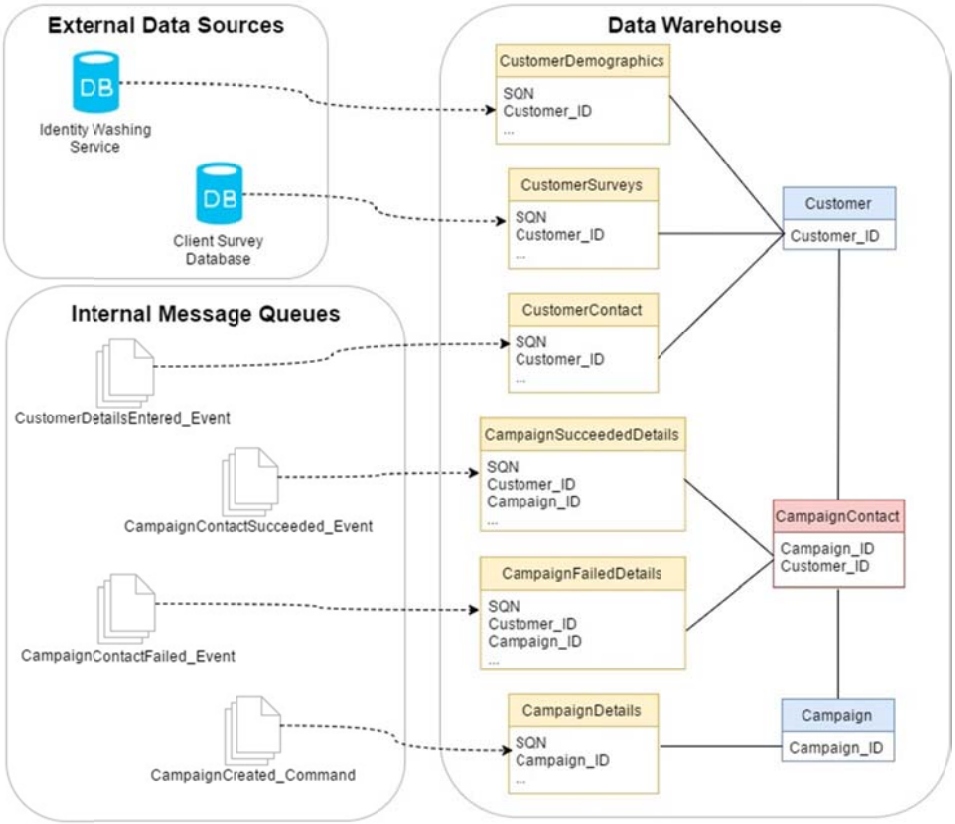

Figure 4. Diagram of Messages and the resulting Data Vault

LATE ARRIVAL

Any Data Warehouse wanting to process event data will probably be faced with the issue of processing late arriving messages. Messages can easily be delayed, have issues and require retransmission or simply be processed out of sequence. A message queue data warehouse should also have the ability to snapshot data, to resync missed messages or to verify that all data has been correctly captured and this should be ideally be done via the message queue itself.

The optimal approach for processing late arriving data will depend on the environment and data architecture. However the ability to process late arriving data will generally require an entirely lossless data warehouse, which common data warehousing concepts like SCD tables may not be.

For example, imagine the following messages are received about a customer's account balance. This data relates to a transaction account. The customer in question transfers money in an out of the account quickly, as they have other accounts for their savings.

The messages received about a customer's account balance are depicted in Table 6 and the resulting SCD table is depicted in Table 7

Account	Balance	DateTime
4396	$0.00	16Jul2016:13:16:54
4396	$0.00	23Jul2016:20:42:02
4396	$0.00	31Jul2016:23:59:59
4396	$3793.16	16Aug2016:07:03:50
4396	$0.00	16Aug2016:15:36:25

Table 6. Tabular representation of Account Balance messages off the Message Queue

SQN	Account	Balance	DTS	EDTS
1	4396	$0.00	16Jul2016:13:16:54	16Aug2016:07:03:50
2	4396	$3793.16	16Aug2016:07:03:50	16Aug2016:15:36:25
3	4396	$0.00	16Aug2016:15:36:25	.

Table 7. SCD Table that these messages could generate in the Data Warehouse

If however the message depicted in Table 8 was received late, how is the data warehouse to know the $2107.26 balance didn't apply all the way up to the 16Aug2016:07:03:50? An erroneous SDC table the data warehouse might generate is depicted in Table 9.

Account	Balance	DateTime
4396	$2107.26	23Jul2016:03:42:09

Table 8. Late Account Balance messages off the Message Queue

SQN	Account	Balance	DTS	EDTS
1	4396	$0.00	16Jul2016:13:16:54	23Jul2016:03:42:09
2	4396	$3793.16	16Aug2016:07:03:50	16Aug2016:15:36:25
3	4396	$0.00	16Aug2016:15:36:25	.
4	4396	$2107.26	23Jul2016:03:42:09	16Aug2016:07:03:50

Table 9. Erroneous SCD Table, record 4 should only last until 23Jul2016:20:42:02

Entirely lossless SCD tables need to be accompanied by reiteration tables. These are tables that record the effective date of all information received by the data warehouse and the SCD record it created or reiterated. Only combined with reiteration tables can SCD tables be an accurate record of all data received by the data warehouse, in the event of messages arriving late the entire message sequence can be recreated and the SCD tables updated.

Table 10 shows the correct SCD structure for the account. It is only by generating the reiteration table, depicted in Table 11, that the SCD table can be entirely lossless and late arriving messages can be correctly processed.

SQN	Account	Balance	DTS	EDTS
1	4396	$0.00	16Jul2016:13:16:54	23Jul2016:03:42:09
2	4396	$3793.16	16Aug2016:07:03:50	16Aug2016:15:36:25
3	4396	$0.00	16Aug2016:15:36:25	.
4	4396	$2107.26	23Jul2016:03:42:09	23Jul2016:20:42:02
5	4396	$0.00	23Jul2016:20:42:02	16Aug2016:07:03:50

Table 10. Correct SCD Table, this should be the result of the messages arriving in any order

SQN	DateTime
1	16Jul2016:13:16:54
2	16Aug2016:07:03:50
3	16Aug2016:15:36:25
4	23Jul2016:03:42:09
5	23Jul2016:20:42:02
5	31Jul2016:23:59:59

Table 11. Reiteration Table necessary for the data warehouse to be entirely lossless

ANALYTICAL PROCESSING

The amount of data generated by a message queue data warehouse can be significant. Message queues give the data warehouse access to a wealth of transactional and behavioral information that can significantly benefit analytical processing, but restructuring the data for analytics can be complex.

The first stage of analytics is summarization and reporting, if Data Vault has been used as the data architecture this can be relatively easy as all the information about an entity is easily linked together using the business keys. A message queue driven data vault data warehouse would even be able to deliver reporting on the error rates of source systems or system latency because of the ease of connecting to multiple commands or events in the information flow.

Reporting and analysis can be delivered will very low latency and even be pushed out to the business when a significant change has been detected.

The task of developing statistical predictive modeling of a message queue data warehouse is however harder, despite the fact that it can deliver far more accurate results. The reason for this is simply the amount of data collected. Whereas models developed off a traditional data warehouse may use current status and status 30 days ago, modeling developed from a message queue data warehouse should be able to sequence the last half a dozen statuses, the length of time spent in each one and a count of the length of time spent in each status since recording started. These sequences are a far more precise indication of past behavior and will provide a more accurate prediction of future behavior. They also enable correlations to be drawn between customers who are on similar paths but at different stages in their journeys.

As much data as possible should be extracted from the data warehouse and initially loaded into the modeling package, with a focus on behavioral indicators, such as times between events and the sequence of activities. It has already been shown that the data warehouse and the messages that load it should be state based, but when it comes to modeling the data should be transformed into deltas again. The fact that the customer recently deposited $2000 or changed their email preferences within an hour of subscribing, is going to be more indicative of their behavior than their current account balance or mailing list subscription.

Because of the event architecture the sequence and duration of information can be guaranteed to be accurate and this approach enables the models to incorporate more historical information. This does lead to a significantly wider dataset and will slow down the execution of any Enterprise Miner Models. Performance can be improved by running initial variable selection and cluster identification on reduced training and validation sets.

The larger amount of modeling information and the improved accuracy of the data should lead to more correlations in the model; these can be identified using the variable clustering tool in SAS Enterprise Miner. It is important, as with any predictive modeling to remove correlations. However identifying correlations can be a valuable modeling technique in itself. Identified correlations should be analyzed and investigated for any valuable causation. Studies have shown that data scientists spend less than 15% of their time actually analyzing data and identifying trends. The correlations identified through modeling are underutilized opportunities to understand behaviors and identify business improvements.

CONCLUSION

Using event queues and the enterprise service bus to populate a data warehouse can be hugely beneficial. It can provide faster access to business level data and allow the data warehouse to record all transactions so the entire business lifecycle can be modeled. Reporting latencies on an ESB data warehouse would generally be in the range of 1-10 minutes and reporting triggers can be used to push data with minimal delay all the way from the source system to the business users. An ESB data warehouse is better equipped to perform all data warehouse functions, because it has greater access to more business data, has minimal latency on reports and can even push alerts out to the business and because it has the full customer lifecycle can perform more accurate behavioral modeling.

Establishing an enterprise service bus architecture within your production systems is complicated. If the business does not have one already it may not be worth implementing one purely for the data warehouse. However, if an organization does have an ESB already it makes architectural sense to utilize it with the data warehouse also. ESB data warehouse's have greater distributed load meaning more computation requires less overhead and can be less development after the initial setup, because many of the processes are similar and templates can be used to save on development and ongoing maintenance.

If event queues are not established yet benefit can still be gained by frequently snapshotting transactional tables and loading them into a Data Vault. In fact, that could be a reasonable phase one of the development cycle. If the messages and the EBS architecture is going to take several months to develop, the data warehouse can still start snapshotting source databases and load historical data. The data warehouse can start delivering valuable analytics, just not with the same low latency and the messages can be connected as soon as they are developed.

REFERENCES

Jahn, Dan. 2016. "The Six Tenets of a Better Decision" Proceedings of the SAS Global 2016 Conference, Las Vegas, NV: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings16/SAS2900-2016.pdf

Linstedt, Dan "Data Vault Basics" Accessed March 5, 2017.https://danlinstedt.com/solutions-2/data-vault-basics/

Press, Gil "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says" Accessed March 2, 2017. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#71ed19c96f63

ACKNOWLEDGMENTS

Many thanks to Sarah Barry, Ned Martin, Nathan Johnston and Greg Nelson for their assistance.

RECOMMENDED READING

- *Using SAS DI Studio To Load A Data Vault* - https://communities.sas.com/t5/SAS-Communities-Library/Using-SAS-DI-Studio-To-Load-A-Data-Vault/ta-p/221697
- A Macro That Can Fix Data Length Inconsistency and Detect Data Type Inconsistency - http://support.sas.com/resources/papers/proceedings16/10000-2016.pdf
- SAS® Application Messaging: How to Integrate Disparate Processes in Your Service-Oriented Architecture - http://support.sas.com/resources/papers/proceedings10/314-2010.pdf
- Reading data with the SAS JSON libname engine - http://blogs.sas.com/content/sasdummy/2016/12/02/json-libname-engine-sas/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Bronwen Fairbairn
Collection House Limited
bronwen@bronwen.org
http://www.bronwen.org