ISSN 0975-3303
Mapana J Sci, 9, 2 (2010), 35-46
https://doi.org/10.12725/mjs.17.5

ETL AUTO RECONCILIATION

Jibrael Jos and Bragadishwaran U***

ABSTRACT

Extraction, transformation and Loading (ETL) is the process of storing data into the data warehouse. Errors in the ETL process can result in wrong data being stored. This paper introduces an automated approach to reconcile the source data with data stored in the data warehouse. This ensures that data in the warehouse is consistent with the source data and all stakeholders have clarity on the quality of the data. The ETL Unit presented in the paper can be considered as a template/ design pattern to implement this strategy. The pattern will help to for implementing both integrated and independent Auto Reconciliation.

***Index Terms**—Data Warehouse, ETL, ETL Unit, Testing, Auto Reconciliation.*

I. Introduction

As the information needs of organization grow, Organizations are increasingly turning towards "Data Warehouse/Business intelligence" applications to satisfy their demands. The key challenges with these applications are the correctness of data being stored and the calculations done on the data.

* *Christ University, jibrael.jos@christuniversity.in*
** *ETL Lead, MindTree Limited, bragdishwaran_u@mindtree.com*

This paper aims to provide a generic approach to consistently address the data quality issues faced in projects which involve data movement.

A regular, ongoing data movement process ideally needs a minimal maintenance overhead once the solution is deployed. The ETL code would have been unit tested, integrated tested and undergone scrutiny of the UAT team. What is sometimes forgotten that even if a single line of code has **not** been changed, the daily waves of data which flow into the warehouse can have an impact on the quality of the data.

ETL Process shifts, cleans and transforms data. Indexes are dropped, partitions made and other such operations which actually needs a closer check. Given the size of the process and possible points of failure, an Integrated ETL Auto Reconciliation Framework is something which cannot be avoided.

II. Related work

Ralph Kimball in his article "38 Sub System of ETL" discuss about the need for different sub system, below is the three which have some relevance to the proposed work.

Sub System 7: Quality screen handler. In line ETL tests applied systematically to all data flows checking for data quality issues. One of the feeds to the error event handler.

Sub System 8: Error Event Handler. Comprehensive system for reporting and responding to all ETL error events. Includes branching logic to handle various classes of errors, and includes real-time monitoring of ETL data quality.

Sub System 27: Workflow monitor. Dashboard and reporting system for all job runs initiated by the Job Scheduler. Includes number of records processed, summaries of errors, and actions taken.

In his article "Is the Data Correct?" he has suggested how one can test using some simple SQL to test whether the underlying data is correct.[3]

Jeff Theobald in his article on "Strategies for Testing Data Warehouse Applications" talks about using statistical function to check for the validity of data seen on a report. [4]

III. Proposed work

An ETL Auto Reconciliation System which will run test cases for each ETL Unit in tandem with the ETL Process. These unit test cases would be able to be triggered by a Test Engine to check either the whole system or relevant sub systems independent of the ETL Process.

These test cases will be run not just during release of the ETL System but will be run automatically for each and every load.

The test cases will exist for every ETL Unit, it will need to be developed at the time of Development of ETL System. The test case itself will be developed by a Business Analyst or a dedicated Tester but to create the SQL / Stored Procedure/Script will need some involvement of a programmer.

There will be other test cases which check the sanctity of the system on the whole. (These are explained in more detail in subsequent sections)

A. ETL Design

- An ETL Design has the following steps
- Files to Staging
- Staging to Dimension
- Staging + Dimension to Fact Tables
- Fact to Aggregated Facts
- Facts to Cross Functional Data Marts

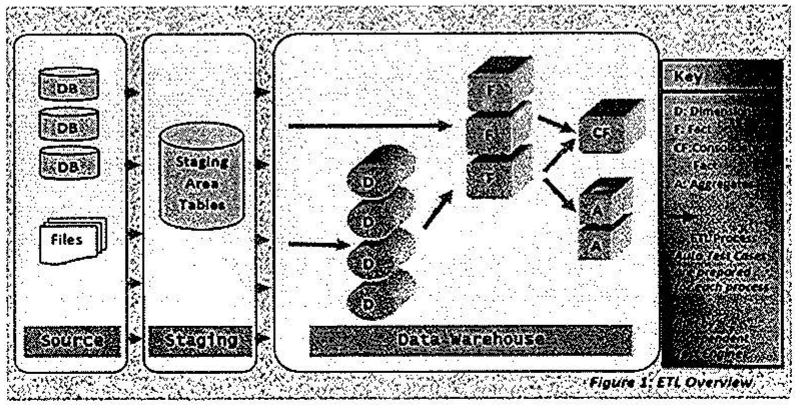

Figure 1: ETL Overview

ETL design will take care of logging, status updates, reject handling and reports to monitor the overall flow of the whole process.

B. Need for Ongoing Testing

ETL coding can get quiet cumbersome and the process which take multiple hours can have unforeseen error creeping in. The absence or corruption of a file, two processes getting out of sync and other reason like server disruption. The multi threading to optimize the data load while making use of the multiple CPU can make it difficult for the DWH Quality manager to make a quick assessment of the how the ETL operations have under gone.

Before the Data Warehouse Quality Manager approves the load for the month, he needs to evaluate which Data Marts have been synchronized for the latest data load. For the data marts which have failed he needs to see the dependencies which have failed.

Changes in any of the source systems for the data warehouse can have cascading impact on the warehouse. Take for example a certain table structure gets modified and loading of the data into staging table fails, which in turn may fail a Fact and in the process a Data Mart. We cannot possibly release a data mart in which we say that everything is fine except for say X measure.

There has been many implementation which have had more than a 1000 individual packages which run everyday and only one person to manage the ETL movement (and that too most probably part time).

C. ETL Unit

One ETL unit can be defined as a single data movement from Source A to Destination B. If ETL process can be considered as a chain of ETL Units, the Figure 2 shows the two layers which normally are involved in the design (ETL and Metadata Layer).

The third Testing Layer is what we are recommending. This Testing Layer will have two interfaces with the ETL unit, one to run the test cases and one to check the results.

Results are checked based on the ETL Dependencies for the ETL unit which we are about to run. So for example while running one Fact from Staging to DWH we will check all the dependent dimensions are okay and even the Staging Table itself is verified.

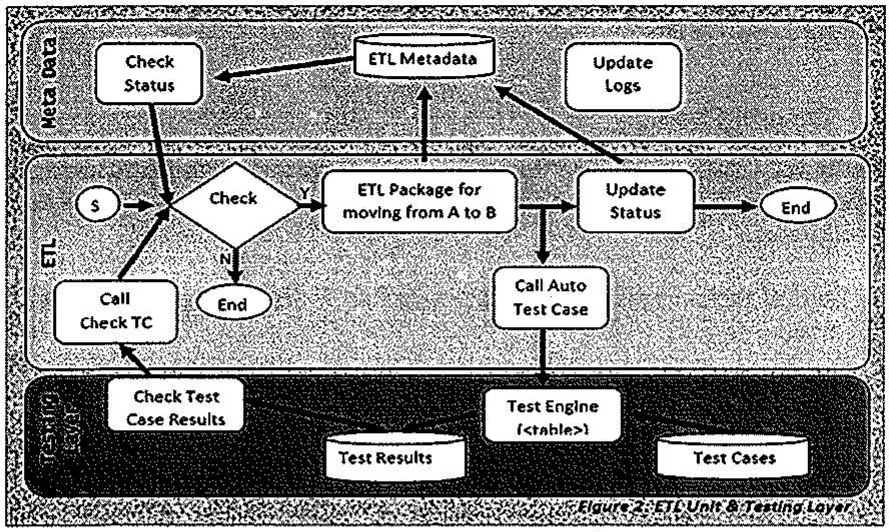

Figure 2: ETL Unit & Testing Layer

Once the ETL unit's main task is over then the process will call the test engine to run the test cases for the unit currently processed.

Like logging level should be configurable, so should the Testing Layer. Running can be triggered off at a unit level or at a group level. Some test cases may be considered as warning and some as critical enough to stop dependent units.

D. Test Engine

The Test Engine is completely Data Driven and every new test case will just need an entry in the ETL Auto Test Case Table.

Testing Layer is made in a generic manner and all test cases will be some DB SQL which will run dynamically. Whether it is a Fact, Dimension or Staging, the pattern would be the same.

This helps us run all the test cases along with ETL and if we choose also independently. This ETL Design pattern guarantees that there is minimal cohesion between the two layers and development can work independent of each other.

IV. Test Cases

A. General Checks

Inserts: Test Cases which have been converted to queries such that source and target tables in a process are evaluated to ascertain if there has been an error in moving data.

- Sum of certain amount for certain account in Staging is equal to sum of amount in Fact
- Row Counts for certain period

Updated columns: Updates needs a more detailed consideration as insert can be checked by row counts. But updates can be more tricky incase of change in a dimension key or attribute. SQL can be written to check cardinality but queries tend to be slow to run. If audit trail is not available in source system it can prove to be very cumbersome though not impossible but feasibility will need to be studied for specific columns on a case to case basis.

Deletions: Incase of hard deletion we will not have any reference data to verify the deleted record, but incase of soft deletion (flag or archive table) can be handled with a suitable query.

B. Specific Checks

Each Fact will have more than 5 test cases, each to evaluate some formulae's, joins, aggregates.

- For a certain keys (account, employee etc)
- Sum of specific measures
- Cardinality of a certain dimension id in a Fact
- Business logic based measures

C. Sanity Checks

- Existence of DB specific statistics
- Existence of Expected Tables
- Existence of Expected Partitions
- Existence of Expected Indexes
- Existence of Expected Materialized/Indexed views

- Parameters which effect performance (in case tampered by some over zealous administrator)
- Partitions wise count (Is a month missing)
- Approximate Checks (Transaction table normally has around 40 million rows coming for the current month, amount in account not above a certain expected limit) and Statistical Checks [4]

D. Performance Test Case

Apart from existence of partitions, indexes and materialized views some additional test cases could be related to database query times, like expected runtime for certain SQL

Pre Processing

Running individual SQL on source system can be time consuming. We may be able to get all the source system table counts in the extraction window and store in the Testing Layer

- SQL which does row count of each data object
- Scripts which count rows of every file loaded
- Scripts which get Performance Parameters
- SQL Test Procedures which evaluate if a file is loaded completely or not

V. Testing Metadata

DWH Objects to keep a tab the ETL Engine stores the Last ETL date for every Database Object. This helps answering questions like:

- Which all tables were not loaded today?
- Which tables have been failing in the last week?

To keep a tab of which Fact to run because a DIM has failed one needs to maintain the dependencies of all objects. This is tracked in ETL Dependencies. So for example we would know that

Fact B is dependent on

- Fact A
- Staging B
- DIM (9 of them)

Fact A itself is dependent on

- STAGING_A
- DIM (12 of them)

And Staging A is dependent on a certain set of

- A1 to A9 source files

If source files could not be loaded there is no point trying to run the process which is loading the Facts. This knowledge of dependencies saves some critical processing time.

The test cases itself will be stored in ETL Test Case Master and results in ETL Test Case Results. The columns will need to include related DWH object, Source test case SQL and destination test case SQL. In certain cases there may be some expected difference due to some known data issues, this number may need to be stored to avoid test case failing daily.

DWH Object Type

Field Name	Description	Remark	Sample
Object_Type_Id	Description of the Test Case	Primary Key	3
Object Type	Fact/ Index/ Staging / Consolidated Fact / Mat Views		FACT

DWH Objects

Field Name	Description	Remark	Sample
ID	Unique Identifier	Primary Key	12
Object Name			FACT_SALES
Object Type ID		FK: DWH_Objects	3
Last ETL Date			20-Feb-2010
Row Count		Based on time cost	23,432,123

ETL Dependencies

Field Name	Description	Remark	Sample
ID	Unique Identifier	PK	12
Object Name Id		FK –DWH_OBJECTS	25
Dependent Object Id			12,423,312

ETL Test Case Master

Field Name	Description	Remark	Sample
ID	Unique Identifier	PK	12
Object_Id	Description of the Test Case	FK –DWH _OBJECTS	25
Test Case Name	Test Case Name		Total Row Count
Source SQL	Query to be run to get Expected Result	Can be SQL, Procedure name to call	SELECT COUNT(*) FROM STG_BILLING
Destination SQL	Query to be run to get Actual Result	*	SELECT COUNT(*) ROM FACT BILLING WHERE DT=?
Expected Difference	Incase there are differences which are known in the current system due to certain anomalies		11
Severity	Whether Test Case failure should stop the load		

ETL Test Case Results

Field Name	Description	Type	Sample
ID	Unique Identifier	PK	123
Test Case Id		FK – Test Case Master	6127
Expected Result	Output of the Query from Destination		12,423,312
Actual Result	Output of the query from Source		12,423,300
Expected Difference			11
Status	Match/UnMatch (Detect/ Reviewed / Approved)		Unmatched
Test Date / Time			23 Sep 2010
Test Batch Id	Incase multiple runs take place we will need to store the batch for which the Test Case run. This will have special significance in when a error run takes place		538
Approved User Id		May need to be a separate table which may track the failure cases	
Approved Date			

VI. Testing Dashboard

It is assumed that the underlying ETL Solution has already provided process driven logging, reports on batch outputs, job failures, rejects on this log table to give the Data Quality Manager a quick assessment of the ETL load for the day. Additional

reports which would be useful the ETL Data Quality Manager could be taken from the proposed testing metadata.

A. Reports

- Data Marts / Cubes Status
- Critical Alerts
- Failed ETL Units
- Dependent Units Not Processed
- Test Case Results for specific ETL Objects
- Test Case Results for type of ETL Object
- Performance Related Test Case Report

B. Report Implementation

Using a simple set of ASP.NET or Java Script some generic interfaces can be made to display the test case results.

The independent run test cases given by the Auto Recon will give the much needed confidence in the data so that the DQM can go ahead and let the bells toll

VII. Future Work

The test cases considered are only those related to running of an ETL, additionally Test Engine can be extended for development and enhancement testing.

ETL Admin Dashboard can be integrated more completely with existing ETL system by creating a Re-Usable Component for the Testing Layer.

VIII. Conclusion

The advantages of this auto reconciliation strategy in data warehousing projects are

- This is more proactive than reactive in nature. The system does not create wrong data, which in turn will not lead to wrong reports, which leads to loss of user confidence in the underlying system.

- Business users are aware of the quality/correctness of the report. This is critical as it prevents disaster scenarios in Data Warehousing projects (when critical business decisions are taken on wrong report data).
- This is a step towards Embedded Testing. Generally a change in the source system introduces data which generates wrong report measures. Conventional "Development Testing" cannot identify these errors. These errors can only be identified by embedded testing which are periodically executed.
- This approach once developed into a generic component will reduce the Development time and increase the quality of the project. The development team does not have to worry about automating test cases.

This process is not the silver bullet for all your maintenance problems, but if there is one way we need to move, that is towards a better process. And we believe the ETL Auto Recon Strategy can be one of the key pieces in the Overall ETL Design Framework.

References

1. Kimball, R./Joe Caserta "The Data Warehouse ETL Toolkit: Practical Techniques for extracting cleaning, conforming, and delivering data". John Wiley and Sons 2004. ISBN 81-265-0554-0
2. Kimball, R. "The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouse extracting cleaning, conforming, and delivering data". John Wiley and Sons 1996. ISBN 0-471-15337-0
3. Kimball, R. "Is Your Data Correct?" http://intelligent-enterprise.informationweek.com. Dec 5 2000
4. Jeff Theobald. "Strategies for Testing Data Warehouse Applications" http://intelligent-enterprise.informationweek.com. June 2007.