WhereScape®

>Near Real Time Processing

>Raphael Klebanov, *Customer Experience at WhereScape USA*

>Definitions

1. Real-time Business Intelligence is the process of delivering business intelligence (BI) or information (DW) about business operations as they occur. "Real time", while unachievable, means near-to-nil-latency and access to information whenever it is required. In this blog we utilize the term "Near Real Time" (NRT) to discuss this type of processing.
2. All NRT systems have some latency; the goal is to minimize the time from the business event to a corrective action or notification. There are three types of latency in any business:
   a. Data latency is the time taken to collect and store the data;
   b. Analysis latency is the time taken to analyze the data and turn it into useful information;
   c. Action latency is the time taken to react to the info and take action.
3. NRT technologies are designed to reduce all three latencies to as close to zero as possible, while traditional BI only seeks to reduce data latency and does not address analysis or action latencies since both are administered by manual processes.
4. Real-time BI systems are Event Driven (ED), and may use various (in memory) techniques to enable events to be analyzed without being first transformed and stored in a database. This subject is omitted from this blog.
5. An alternative approach to ED architectures is to speed-up the refresh cycle of an existing DW to update the data more frequently. This can achieve NRT update of data, with data latency is in the range from minutes to hours. The analysis of the data is still manual, so the total latency is significantly higher compared to ED approaches.

>Change Data Capture (CDC)

CDC is used to load only new or changed data from a source system and has three major types:

1. Data source-driven CDC uses the timestamp or sequenced IDs (e.g. guid) to identify the last loaded rows and store them in DW audit tables. The audit tables hold data for all jobs and all tables that need to be in a consistent state. For all tables the last processed keys (source/target) and the statuses are saved. These audit tables work complementary to RED metatables including ws_job_log, ws_wrk_audit_log, and other ws_wrk_% tables.
2. Snapshot-driven CDC is used when no proper timestamps or IDs are available - or when some records might have been updated. Snapshot-driven CDC stores a copy of the loaded data in a work table(s) and compares it, record-by-record, to an upcoming dataset via delta SQL queries, DB functions, or transformations. Using the hashed value of the "Change Detection Fields" accelerates the CDC. For example:

```
CAST(HASHBYTES('SHA2_256',(dbo.MULTI_HASH_FNC('tblname','schemaname'))) AS
VARBINARY(32));

CONVERT(CHAR(32),HASHBYTES('MD5',CAST(UPPER(CAST(CASE WHEN col1 IS NULL THEN
'UNKNOWN' WHEN col1 LIKE '' THEN 'EMPTY STRING' ELSE LTRIM(RTRIM(col1)) END AS
VARCHAR(150))) AS VARCHAR(150)),2) + '|' +
CONVERT(CHAR(32),HASHBYTES('MD5',CAST(UPPER(CASE WHEN col3 IS NULL THEN CAST(-1
AS CHAR(2))+' - '+LTRIM(RTRIM(col2)) WHEN col3 LIKE '' THEN CAST(-2 AS CHAR(2))+'
- '+TRIM(RTRIM(col2)) ELSE CAST(CAST(col3 AS INT) AS VARCHAR(10))+' - '+
TRIM(RTRIM(col2))END) AS CHAR(500))), 2); … and so on

SELECT CHECKSUM(*) FROM schemaname.tblname WHERE columnname = ?;
```

3. Trigger-driven CDC is the closest to "real time"; however it introduces an additional layer of complexity and maintenance for the DBA. This solution creates triggers to write the changed data to a separate table.

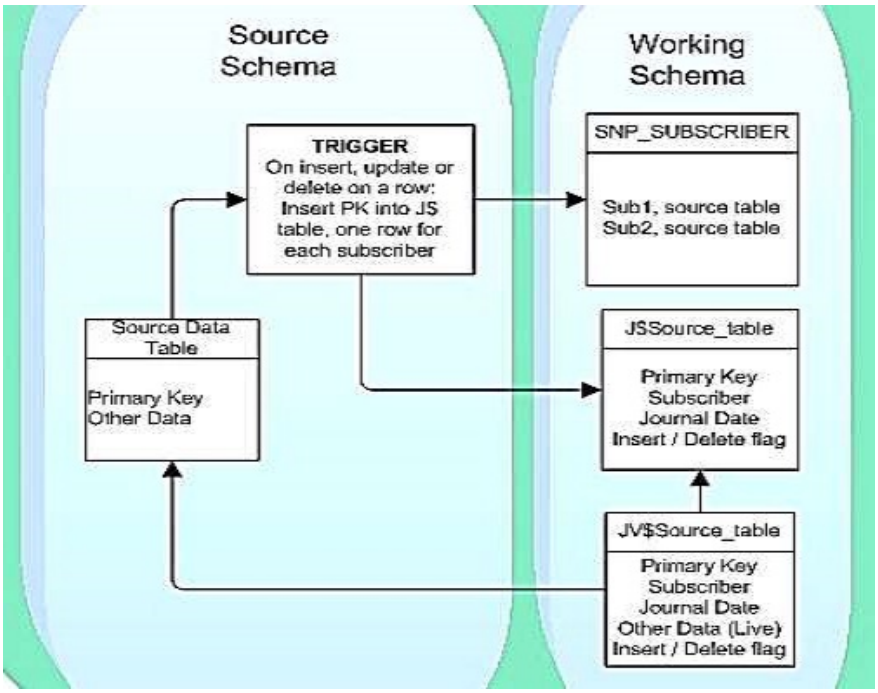

Figure 1 Trigger-based CDC

>Batching

1. Mini-Batching: Data is loaded incrementally using intra-day loads.
   a. Approximate latency is 15 to 30 minutes
   b. Change captured by WHERE clause, e.g.
   c. WHERE [datetime] BETWEEN $Plastrun$ AND $P currrun$
2. Micro-Batching: Source changes are captured and accumulated to be loaded in intervals.
   a. Approximate latency is 2 to 10 minutes
   b. Change captured by the CDC process
3. Use Batch Processing option for proc. code. This allows for the selection of a field to batch process on, e.g. a date field.
4. "Bursts" via RED Scheduler
   a. Maximum Threads. If more than one thread is allocated then they will run in parallel
   b. Logs Retained set to 1 to reduce build-up of the scheduler log.

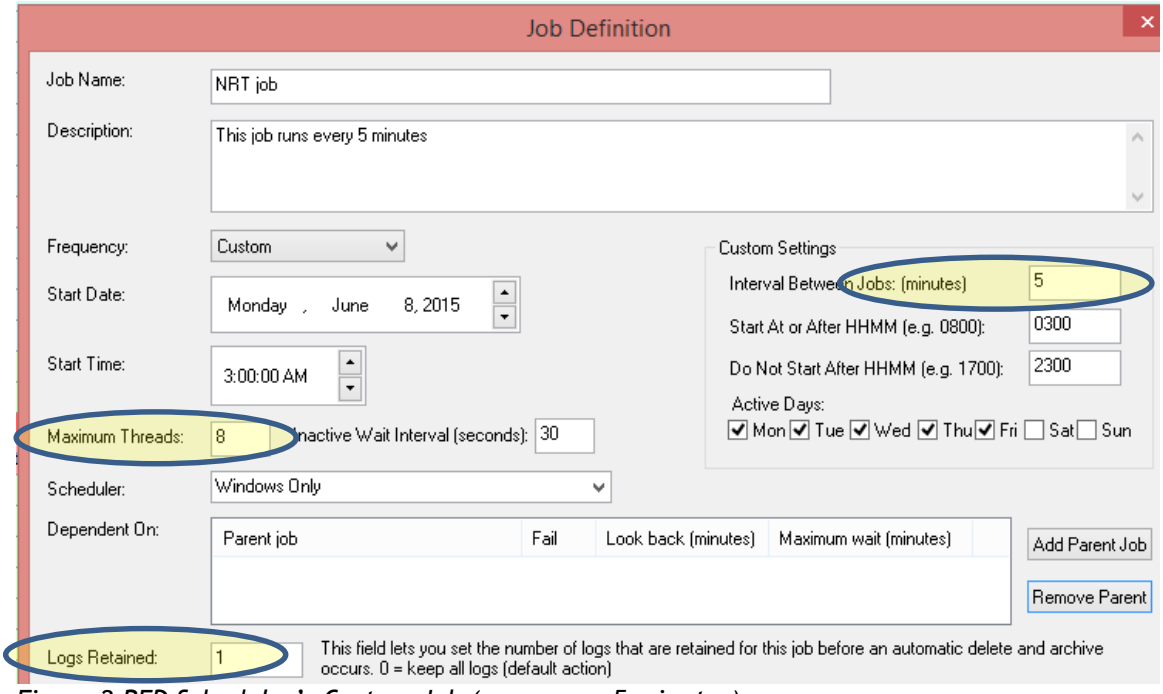

Figure 2 RED Scheduler's Custom Job (runs every 5 minutes)

5. Use set of audit/control tables to capture stats.
6. Decrease Impact on Source System:
   a. Transactional Replication
   b. Hints (locking mechanism) e.g. TABLOCK, NOLOCK, READCOMMITTED, etc.
   c. Read-only connections
   d. Indexed views on top of tables as data source
7. Use Batch Processing option for proc. code. This allows for selecting a field to batch process on, e.g. orderdate field

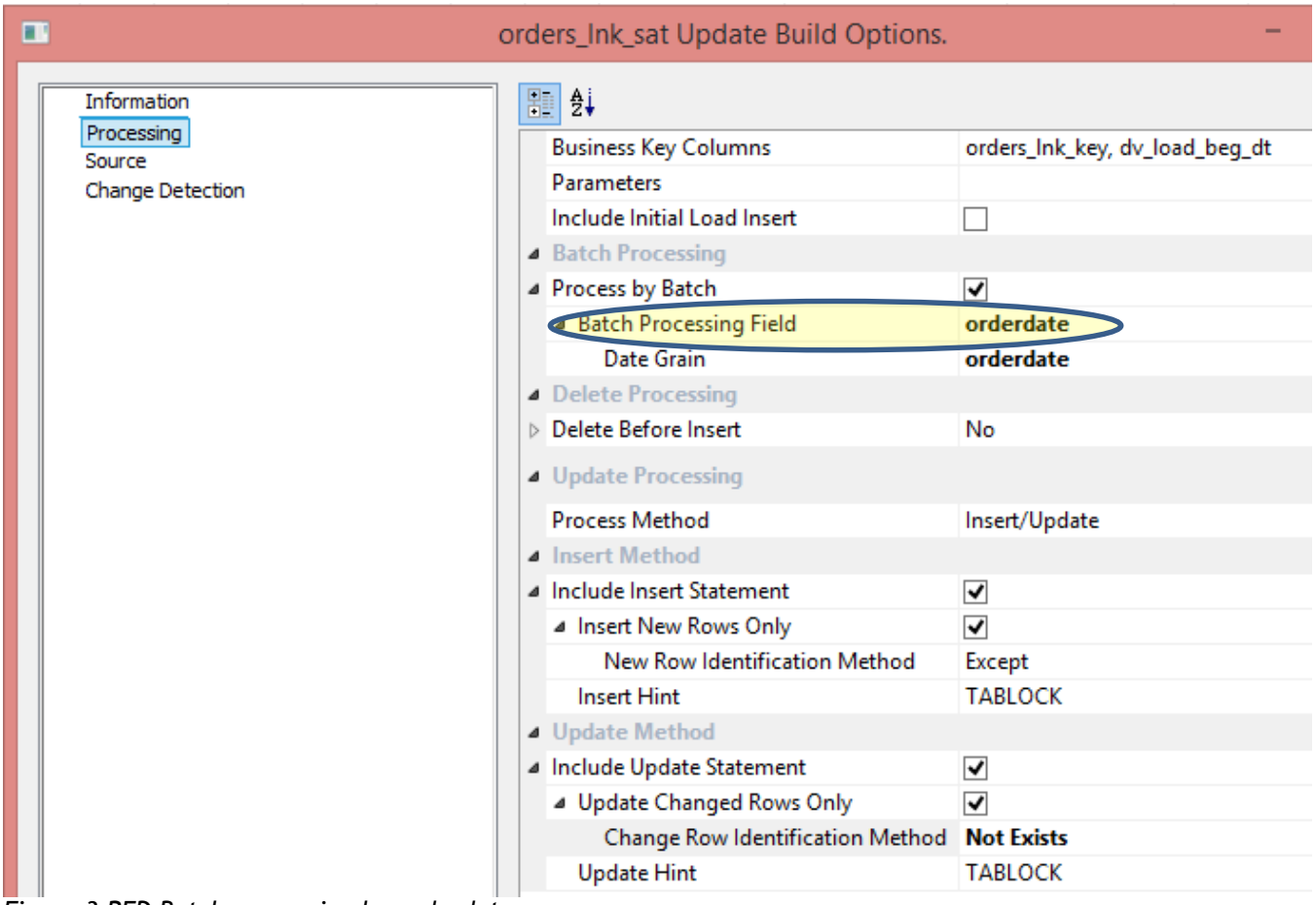

Figure 3 RED Batch processing by orderdate

8. Because job runs very frequently, you might want to add this check:

```
BEGIN --Ensure that the job isn't already running or about to run;
      --if so, exit and wait for this job to run again in 5 minutes
EXEC Ws_Job_Status @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id,
null, 'NRT_test', null, null, @v_return_code OUTPUT, @v_return_msg OUTPUT,
@v_result OUTPUT, @v_job_status_simple OUTPUT, @v_job_status_standard OUTPUT,
@v_job_status_enhanced OUTPUT
IF  @v_result in(1, -1)  -- the status check was successful
--Check the result:
  BEGIN
    EXEC Ws_Job_Release @p_sequence, @p_job_name, @p_task_name, @p_job_id,
@p_task_id, 'NRT_test',p_task_id, 'NRT_test',@v_return_code OUTPUT, @v_return_msg
OUTPUT, @v_result OUTPUT
    IF @v_result = 1  --See if the job was released successfully:
      BEGIN
      StartJob Set StartJob = 'N' …
      END
    END
END
```

>Parallelism

Parallelism is important architectural consideration for DV2.0. All the same types can be loaded in parallel via WhereScape Scheduler.

Databases and appliances built upon the concept of MPP (Massively Parallel Processing) by nature have a vast advantage over single-threaded processing. RED supports such platforms as Teradata, Netezza, Greenplum and, recently, MS PDW.

Below are two diagrams showing the WhereScape implementation in case of Data vault DW "flavor"

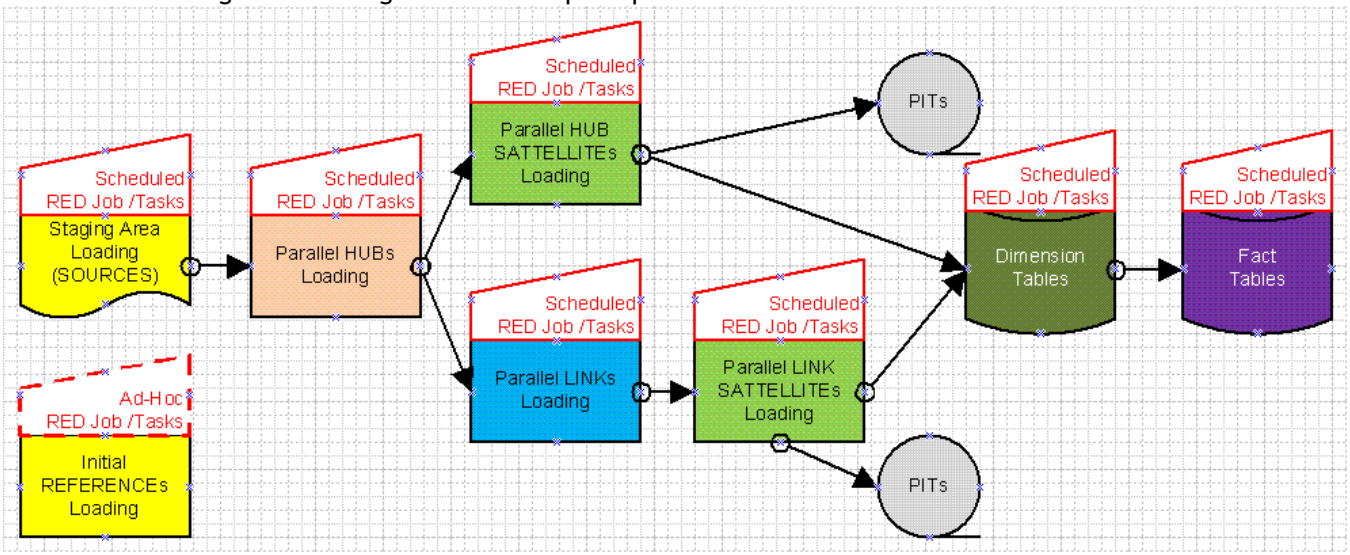

Figure 4 RED implementation of Data vault parallel tasks

Figure 5 RED Implementation example

>Compression

1. Reduce the size of the database
2. Improve performance of I/O intensive workloads because the data is stored in fewer pages and queries need to read fewer pages from disk
3. Extra CPU resources are required on the database server to compress and decompress the data, while data is being exchanged.
4. The following objects can be compressed:
   a. A whole table that is stored as a heap; a whole table that is stored as a clustered index; a whole nonclustered index; a whole indexed view.
   b. Each partition in a partition table or a view; a whole columnstore table or a clustered columnstore index; a whole nonclustered columnstore index.
   c. Each partition is a partitioned columnstore table or index.
5. WhereScape supports all the objects that can be compressed

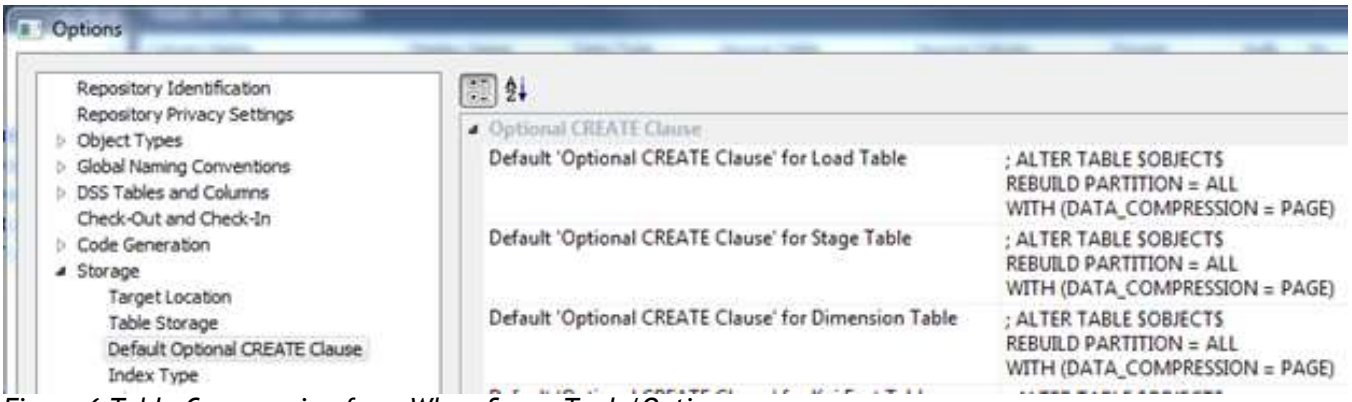

Figure 6 Table Compression from WhereScape Tools/Options

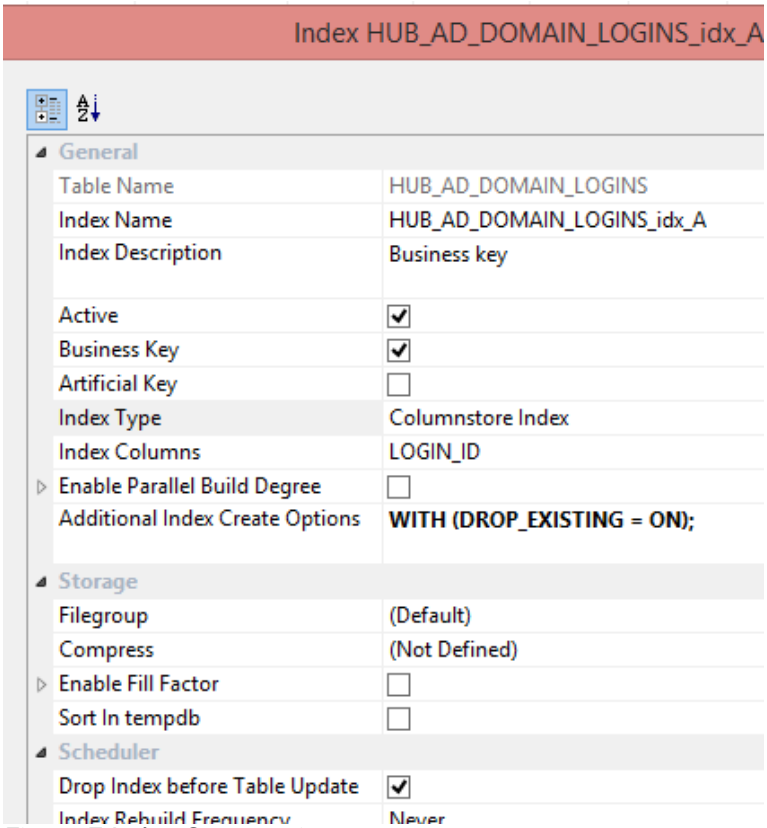

Figure 7 Index Compression

>Partitioning

1. The data of partitioned objects is divided into units that can be spread across more than one filegroup in a database
2. This enables access to subsets of data quickly and efficiently, while maintaining the integrity of a data collection
3. Maintenance operations can be performed on one or more partitions more quickly. E.g. compressing data in one or more partitions or rebuilding one or more partitions of an index
4. Improves query performance. For example, the query optimizer can process equijoin queries between 2 or more partitioned tables faster because the partitions themselves can be joined
5. Cube partitioning -- each measure group in a cube is divided into partitions, where a partition defines a portion of the fact data that is loaded into a measure group
6. Performance and scalability: partitions can be processed separately and in parallel, diff. aggregations. SSAS automatically scans only the partitions that contain the necessary data for a query
7. Partitioning minimizes index re-creation time.
8. Partitioning in WhereScape RED
9. Automated partitioning via wizards: Detail partitioning, Rollup partitioning, Exchange partitioning.

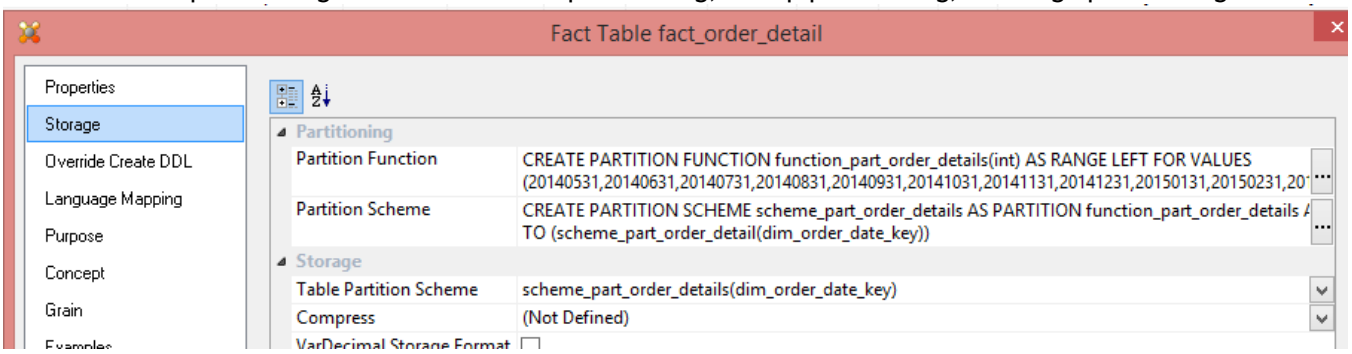

Figure 8 Automated partitioning of a Fact table

10. Automated cube partitioning (via wizards)

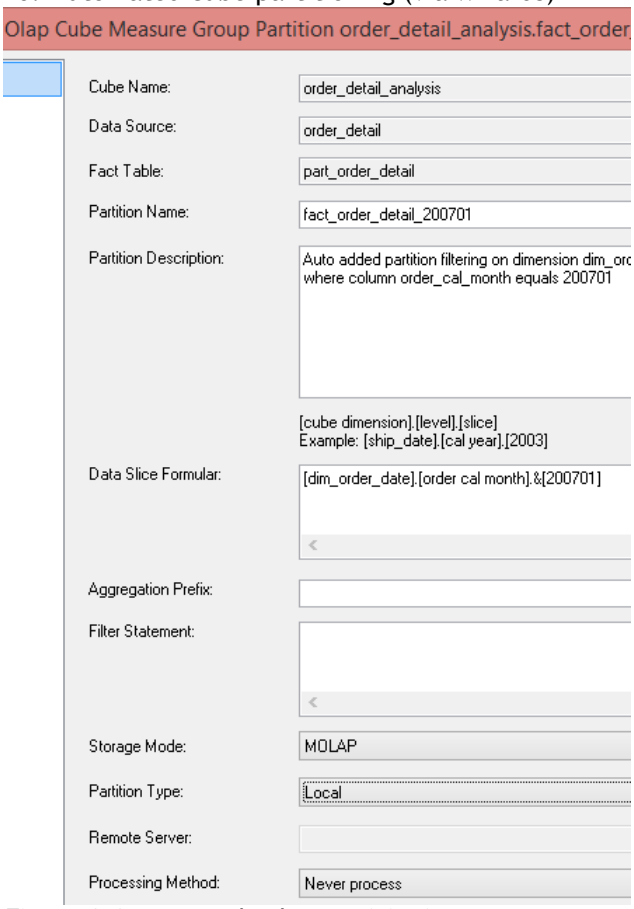

Figure 9 Automated cube partitioning

11. Other supported features include:
12. Partitioned indexes creation and maintenance;
13. Exchange partition code;
14. Intelligent aggregating processing;
15. Only processing aggregates that need reprocessing.

>Big Data

The Big Data shows ability to store and analyze huge amounts of structured and unstructured data promises ongoing opportunities for businesses, academic institutions, and government organizations. Intel has shown through this research that significant performance gains are possible from Hadoop through a balanced infrastructure based on well-selected hardware components and the use of the Intel® Distribution for Apache Hadoop* software.

WhereScape provides a number of functionality supporting Big Data NRT solutions:

1. Extended Hive as a target
2. DB-specific loaders for Hadoop to
3. RDBMS (Oracle/Teradata/Greenplum)
4. Moving of data between RDBMS and HDFS or Hive.
5. Processing of ELT on Hive for standard object types – Load, Stages, Dims, Facts, Aggregates, etc.
6. New "File" object = 'HADOOP'. Load tables created from HADOOP file directly.

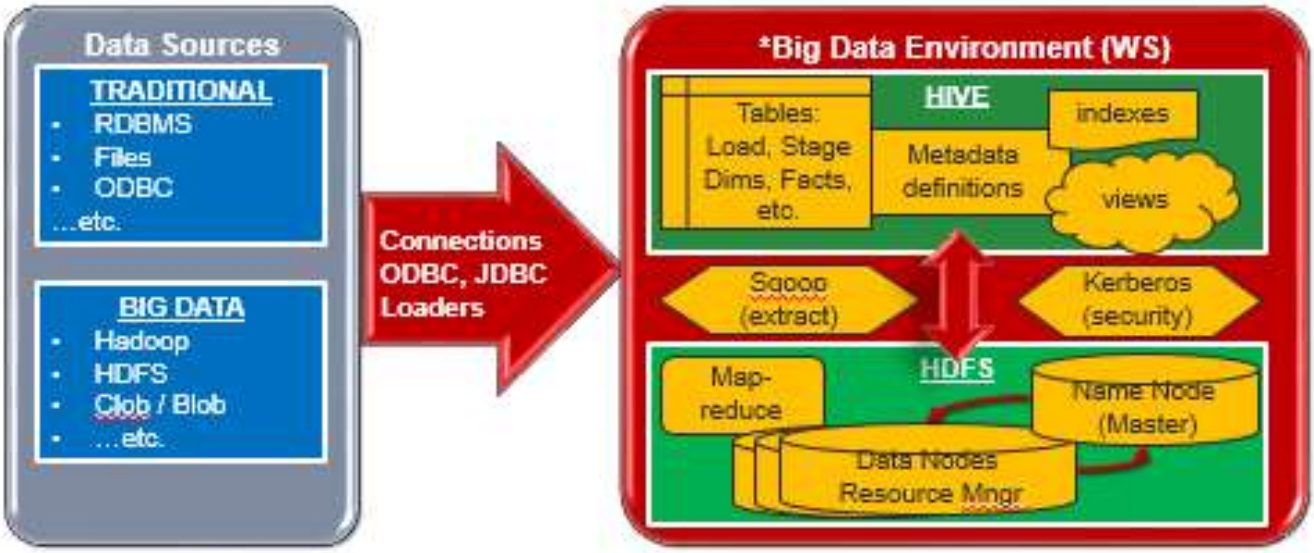

Figure 10 WhereScape Big Data solution (WIP)

>Conclusion

1. A traditional DW architecture should not assume a real time context keeping the system online while updating its data.
2. However, it is possible, under certain circumstances, to have near real time (NRT) processing in traditional data warehouse architectures, if we are able to implement:
   a. Change Data Capture (CDC) architecture, optionally with unloading data to offline;
   b. Use of mini- or micro-batching;
   c. Parallelism while processing information systems' homogeneous objects;
   d. Partitioning the data and index by partition;
   e. Data and indexes compression;
   f. Minimizing of the impact of the source system while querying data;
   g. Use of hashkey for primary keys and change detection;
   h. Utilizing Hadoop and distributed data environments;
   i. The application that accepts short offline periods.