High Speed ETL on Low Budget

Introduction

Data Acquisition & populating it in a warehouse has traditionally been carried out using dedicated ETL tools available in the market. An enterprise-wide Data Warehousing project would often budget a few hundred thousand dollars to invest in an ETL tool like Informatica or DataStage. These functionally rich, best-of-breed tools usually have their own ETL engine to process data along with programmer & user friendly GUI's to design and schedule the jobs.

However the underlining principle still remains the same: Use SQL to read data, transform, write to the target using SQL commands. The value-add such tools provide is ability to handle different kinds of data sources (flat files, XML, RDBMS, etc) with a standard interface/connectors, a responsive designer interface and orderly job monitoring screens.

But with recent enhancements in SQL and new ETL centric features brought in by the leading database vendors redefine the way to implement such data flows by using database itself as the ETL engine.

ETL focused enhancements in Databases

Let's take a look at the few recently introduced ETL focused features in the Oracle Database, these would also be found in other RDBMS sooner or later.

External Tables – *Virtual tables accessing external data residing in flat files*

Oracle 9i's external table feature allows data in external data sources like flat files to be exposed in the database like any other data residing in a regular table. This external table acts as "virtual table" which can be queried, joined directly using the full power of SQL, PL/SQL or JAVA without requiring the external data to be loaded in the DB. No DML operation or indexing is possible on this table.

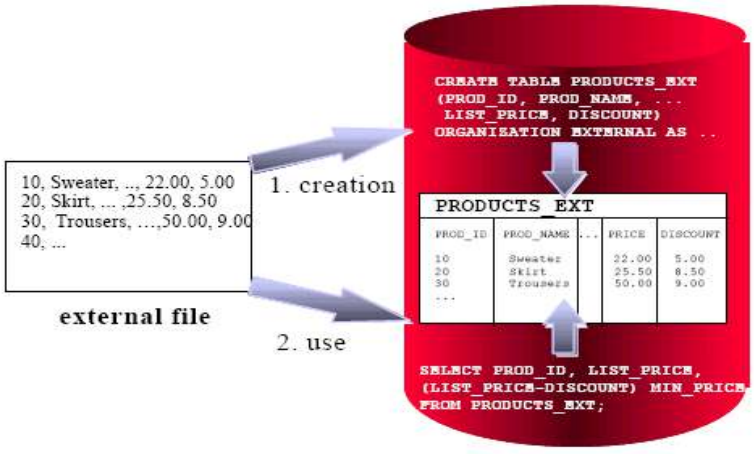

Figure 1 – Concept of an External Table [2]

Table functions – *Pipelined & Parallel execution of transformations implemented in PL/SQL, C or Java*

In a typical ETL process data extracted from a source system passes through a sequence of transformations before being loaded into the Data Warehouse. When results of a transformation are too large to fit in the memory, they must be staged intermediately in tables or flat files. This data is then read and processed as input to the next transformation in the sequence.

Such scenarios can however be done without requiring the use of staging tables, which interrupt the data flow through various transformation steps.

Table function is a function that can take a set of rows as input and produce a set of rows as output. These set of rows are processed iteratively in subsets, thus enabling a *pipelined* mechanism to stream these subset results from one transformation to the next before the first operation is finished. Further more they can be processed transparently in parallel.

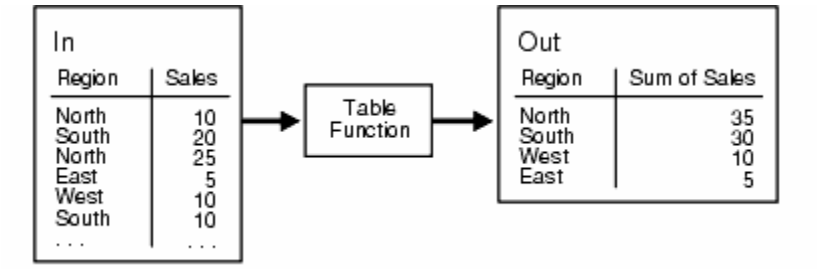

Figure 2 – Table Function Example [1]

Upsert Functionality – *Conditional Updates & Inserts in one pass*

Very often an Upsert logic needs to be incorporated in ETL. Insert a given a row if it's entirely new, if not then update the previously existing row in the target table. This functionality is delivered by the MERGE statement in SQL providing the ability to Update or Insert a row conditionally in a table. The MERGE syntax is now part of the SQL:2003 standard and is supported by most Database vendors in their latest releases.

Case Study

Let's instantiate this idea with a case study using these ETL focused features to drive home the point.

Figure 1 & 2 are screen shots of a mapping done with an industry standard ETL tool. It does the following –
Source data coming in flat files had to undergo some transformation before being "Upserted" (Insert if not exists, else Update) to the target DW tables.

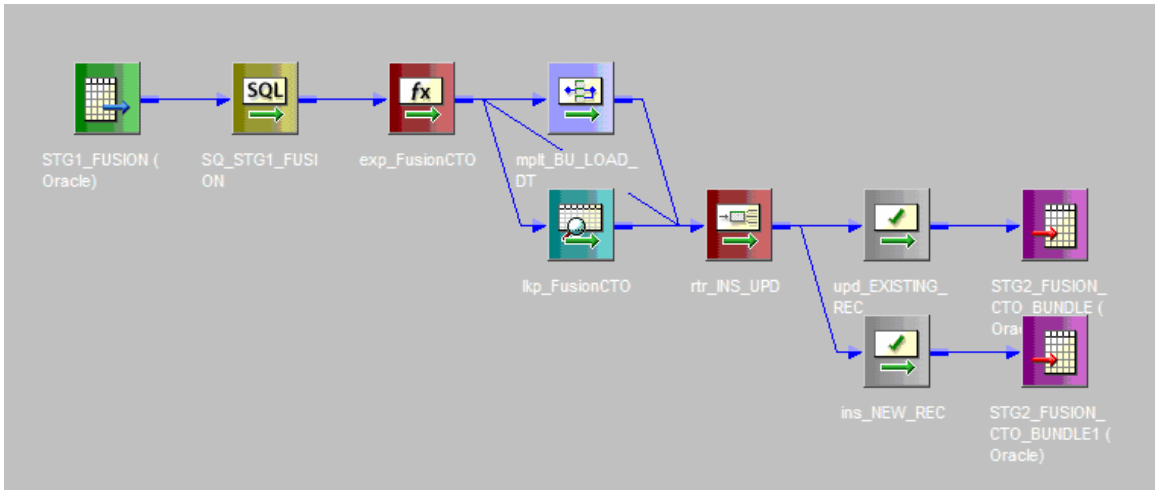

Figure 3 – Mapping for Step STG1 → STG2

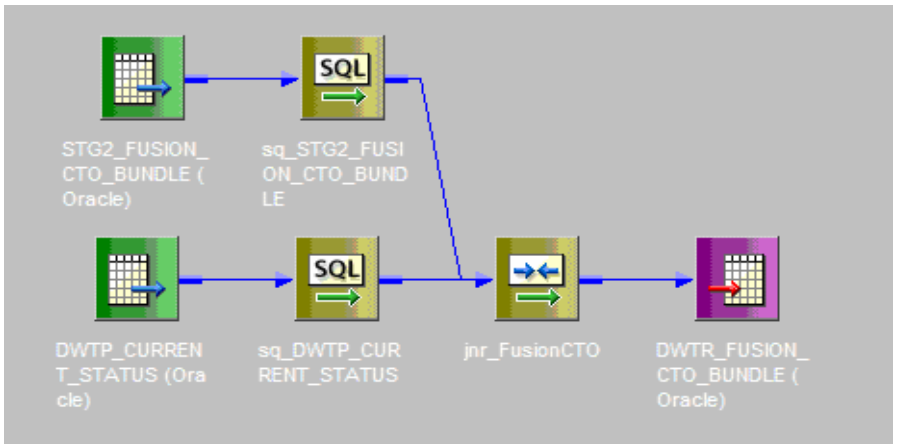

Figure 4 – Mapping for Step STG2 → DW

In line with environment setup at the customer, the loading process is phased in 3 steps: Flat file → STG 1 → STG2 → DW

Step 1: We used SQL* Loader to load STG1 i.e. Staging Area 1 (Some tool constraints prevented us from using the ETL tool to do this), alternatively an ETL mapping can do this too. Data from files is brought in "as is" to a table.

Step 2: STG1 to STG2 (Staging Area 2) is a mapping with transformations, followed by "Upsert" functionality on STG2 table.

Step 3: STG2 to DW mapping is a direct move to DW target with a current date lookup.

We tried to achieve the same using native SQL & PL/SQL features and were able to cover all three steps & execute it in one command!

Here's how we address each step of the ETL data flow using the alternative approach -

Step 1: Use *External tables* to read from flat files, no need to load in a physical table in STG1. Instead of having a populated STG1 table, just use *Select * from <External table>*

Step 2: The Select from external table will serve as an input to a *Pipelined Table function*. Any complex transformations can be coded in the table function. This table function spits out a table of records as soon as they are processed.

Step 3: Use a MERGE SQL statement for Upsert functionality. This SQL command calls the table function (which reads from External table, performs transformations on the fly and delivers a transformed data stream for the upsert) and updates records that already exist, if not then inserts.

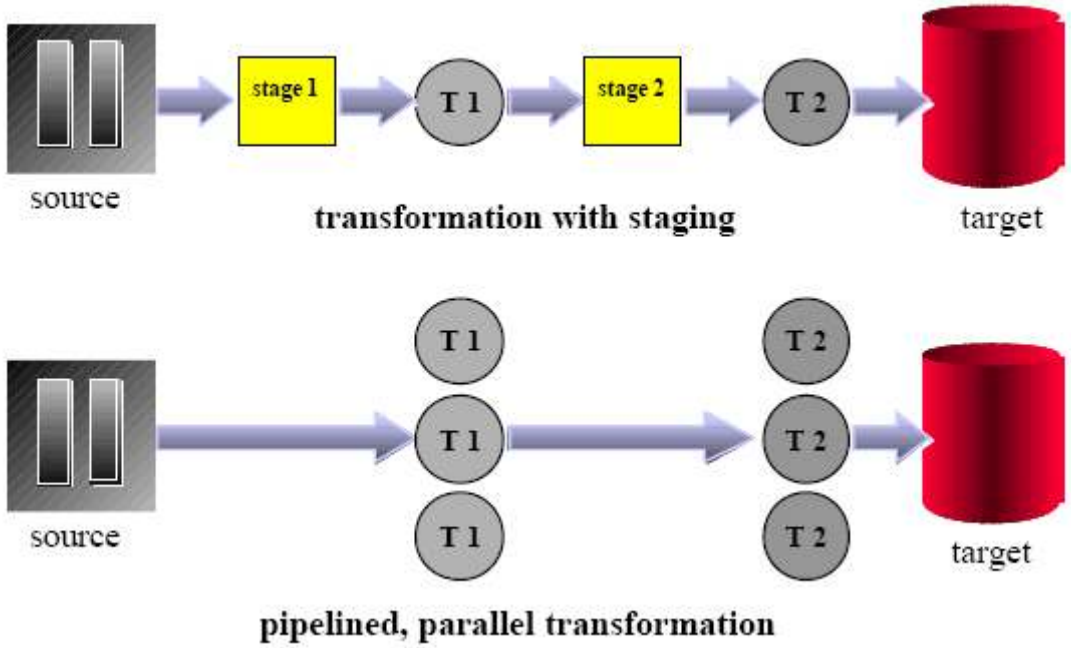

Figure5 – Schematic showing staged data flow v/s data flow with on-the-fly transformations [2]

As depicted in Figure 5, the contemporary process of using SQL*loader, mappings & Staging tables to populate the warehouse can also be achieved using a simple SQL. Below is the MERGE statement which delivers the entire process in one go -

```
MERGE INTO dwtr_fusion_cto_bundle d          -- Upsert command
USING (Select * From TABLE(fusion_feed)) f   -- Calling a table
function
ON (d.heart_order_number = f.heart_order_number AND
    d.heart_item_number = f.heart_item_number)
WHEN MATCHED THEN
UPDATE SET d.load_date = d.load_date
WHEN NOT MATCHED THEN
```

```
INSERT (heart_order_number, heart_item_number,
high_level_item_number, bundle_id, load_date, bu_load_date)
VALUES (f.heart_order_number, f.heart_item_number,
f.high_level_item_number, f.bundle_id, f.load_date,
f.bu_load_date);
```

Conclusion

Using the SQL versatility and new functionality introduced, any ETL task can be delivered at a fraction of the cost. Tools like Oracle Warehouse Builder and Microsoft DTS are based on the same fundamentals. They use the database as their processing engine and capitalize on the native features to offer a commendable ETL tool, not to mention they cost much less than the more established ones.

Gain in performance is another major plus. One, there is no data movement through a separate engine involved, second parallel / bulk loading and upserts can be implemented more promptly to your advantage.

Finally, there are no overheads in terms of setting up separate hardware or developer training. One can get started as long as the team is familiar with writing SQL.

"Guerilla ETL" scores well in terms of cost, performance & readiness.

References

[1] Oracle Data Warehousing Guide
[2] Hermann Baer, ETL Processing within Oracle9i, Oracle White Paper