|  |
| --- |
| EPAM Systems, RD Dep. |
| MTN.BI.08 ETL and SQL Review |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | [Elias Nema](mailto:Elias_Nema@epam.com) | 20-JAN-2014 |  |  |
|  |  |  |  |  |  |

Contents

[1. Data Warehousing Basics 3](#_Toc384328259)

[2. Loading Data into the Data Warehouse 3](#_Toc384328260)

[2.1. Is Cleanliness Best? 4](#_Toc384328261)

[3. Oracle Data Load 5](#_Toc384328262)

[3.1. Prepare for data loading 6](#_Toc384328263)

[3.1.1. Database objects necessary for data loading 6](#_Toc384328264)

[3.1.2. Create an External Table 6](#_Toc384328265)

[3.1.3. Considerations for the target table 7](#_Toc384328266)

[3.1.4. DML versus DDL 7](#_Toc384328267)

[3.2. Analyzing and monitoring your load operation 7](#_Toc384328268)

[4. Source Books and Articles 8](#_Toc384328269)

# Data Warehousing Basics

Why build a data warehouse? Why is the data in an online transaction processing (OLTP) database only part of a business intelligence solution? Where does Big Data fit in a data warehousing deployment strategy?

Data warehouse relational database platforms are often designed with the following characteristics in mind:

* *Strategic and tactical analyses can discern trends in data.* Data warehouses often are used in creation of reports based on aggregate values culled from enormous amounts of data. If OLTP databases were used to create such aggregates on the fly, the database resources used would influence the ability to process transactions in a timely manner. These ad hoc queries often take advantage of computer-intensive analytic functions embedded in the database. Furthermore, if data volumes of this size were entirely pushed to in-memory databases in a middle tier, the platform cost would be prohibitive.
* *A significant portion of the data in a data warehouse is often read-only, with infrequent updates.* Database manageability features can make it possible to deploy warehouses containing petabytes of data, even where near real-time updates of some of the data is occurring.
* *The data in source systems is not “clean” or consistent across systems.* Data input to transactional systems, if not carefully controlled, is likely to contain errors and duplication. Often, a key portion of the data warehouse loading process involves elimination of these errors through data transformation. Since multiple source systems might differ in data definitions, data transformations during the ETL (extraction, transformation, and load) process can be used to modify data into a single common definition as well as improve its quality.
* *The design required for an efficient data warehouse differs from the standard normalized design for a relational database.* Queries are typically submitted against a fact table, which may contain summarized data. The schema design often used, a star schema, lets you access facts quite flexibly along key dimensions or “lookup” values. (The star schema is described in more detail later in this chapter.) For instance, a business user may want to compare the total amount of sales, which comes from a fact table, by region, store in the region, and items, all of which can be considered key dimensions. Today’s data warehouses often feature a hybrid schema that is a combination of the star schema common in previous-generation data marts with third normal form schema for detailed data that is common in OLTP systems and enterprise data warehouses.
* *The data warehouse often serves as a target for meaningful data found on Big Data platforms that optimally solve semi-structured data problems.* Big Data can be described as semi-structured data containing data descriptors, data values, and other miscellaneous data bits produced by sensors, social media, and web-based data feeds. Given the amount of irrelevant data present, the processing goal on a Big Data platform is to map the data and reduce it to data of value (hence “Map Reduce” callouts in programs written using languages such as Java and Python that refine this data). This subset of Big Data is usually fed to a data warehouse where it has value across the business and might be analyzed side by side with structured data.

# Loading Data into the Data Warehouse

Experienced data warehouse architects realize that the process of understanding the data sources, designing transformations, testing the loading process, and debugging is often the most time-consuming part of deployment. Transformations are used to remove bogus data (including erroneous entries and duplicate entries), convert data items to an agreed-upon format, and filter data not considered necessary for the warehouse. These operations are often used to improve the quality of data loaded into the warehouse.

The frequency of data extraction from sources and loading into the data warehouse is largely determined by the required timeliness of the data in order to make business decisions. Most data extraction and loading takes place on a “batch” basis and data transformations cause a time delay. Early warehouses were often completely refreshed during the loading process, but as data volumes grew, this became impractical. Today, updates to tables are most common. When a need for near real-time data exists, warehouses can be loaded nearly continuously using a trickle feed if the source data is relatively clean, eliminating the need for complex transformations. If real-time feeds are not possible but real-time recommendations are needed, engines such as Oracle’s Realtime Decisions are deployed.

## Is Cleanliness Best?

Once the data in the warehouse is “clean,” is this version of the true nature of the data propagated back to the originating OLTP systems? This is an important issue for data warehouse implementation. In some cases, a “closed loop” process is implemented whereby updates are provided back to the originating systems. In addition to minimizing some of the cleansing that takes place during future extractions, operational reports become more accurate.

Another viable option is to avoid cleansing by improving the quality of the data at the time of its input into the operational system. As noted previously in this chapter, this is critical if OLTP systems are to be directly accessed for business intelligence. Improving data quality at the source also enables high-speed loading techniques to be used in near real-time data warehouses (since transformations can be eliminated).

Improving data quality at the source can sometimes be accomplished by not allowing a “default” condition as allowable input into a data field. Presenting the data-entry person with an array of valid options, one of which must be selected, is often a way to ensure the most consistent and valid responses. Many companies also provide education to the data-entry people, showing them how the data they’re keying in will be used and what the significance of it is.

Key Oracle products and database features that often help facilitate this process include:

* **Oracle Data Integrator (ODI).** Acquired by Oracle in 2007, this extraction, transformation, and loading (ETL) tool that handles heterogeneous sources and targets is sometimes referenced as an ELT tool since transformations are pushed into the target data warehouse. This product has replaced Oracle Warehouse Builder as Oracle’s primary offering for ETL. ODI features Knowledge Modules that define integration capabilities, including extraction with change data capture, loading and unloading utilities, SQL-based loading and unloading, and transformation logic SQL. Data Quality options include data quality profiling, batch processing, and address verification. The Knowledge Modules are modifiable. The product architecture includes a development environment that makes use of the Knowledge Modules as templates in declarative design processes and an orchestration agent. ODI can be used for data-based, event-based, and service-based data integration.
* **Oracle GoldenGate.** Acquired by Oracle in 2009, GoldenGate has replaced Oracle Streams as the primary software recommended for log-based replication. Often used for zero downtime software upgrades, during software migrations, and for low latency transaction replication and recovery, GoldenGate supports a wide variety of data sources and targets. It is often used to load Oracle-based data warehouses where the need for data transformations is minimal and near real-time updates of the data in the data warehouse are desired.
* **Transparent Gateways and Heterogeneous Services.** Provide a bridge to retrieve data from non-Oracle sources using Oracle SQL to load an Oracle Database. Heterogeneous Services provide ODBC connectivity to non-Oracle relational sources. Gateways can optionally provide a higher level of performance when extracting data from non-Oracle sources.
* **Transportable Tablespaces.** Another feature for data movement, Transportable Tablespaces enable rapid data movement between Oracle instances without export/import. Metadata (the data dictionary) is exported from the source and imported to the target. The transferred tablespacecan then be mounted on the target. Oracle Database 10g introduced cross-platform Transportable Tablespaces, which can move a tablespace from one type of system (e.g., Solaris) to another (e.g., Linux).
* **Transportable Partitions.** Oracle Database 11g introduced Transportable Partitions for rapid data movement between Oracle instances.
* **Data Pump Fast Import/Export.** Added in Oracle Database 10g and enabled via external table support, Data Pump is a newer import/export format. Parallel direct path loading and unloading are supported.
* **Oracle Big Data Connectors.** First available in 2011, Oracle’s Big Data Connectors include an Oracle Loader for Hadoop that pushes the preprocessing of data to be loaded into an Oracle data warehouse to the source Hadoop Big Data cluster. The result is lessened CPU and I/O impact on the target Oracle Database platform. An Oracle Data Integrator Application Adapter for Hadoop provides ODI Knowledge Modules optimized for Hive and the Oracle Loader for Hadoop. An Oracle Direct Connector for HDFS and an Oracle R Connector for Hadoop are also provided.

The Oracle Database helps the ETL and loading process in a variety of ways. For high-speed loading of flat files, Oracle SQL\*Loader’s direct path load option provides rapid loading by bypassing the buffer cache and rollback mechanism and writing directly to the datafile. You can run SQL\*Loader sessions in parallel to further speed the table loading process (as many warehouses need to be loaded in a limited “window” of time).

Many popular ETL tools, including ODI, generate SQL\*Loader scripts. The core Oracle Database engine features embedded ETL functions that ETL tools support to varying degrees. Examples of these features include support for external tables, table functions, merge (i.e., insert or update depending on whether a data item exists), multitable inserts, change data capture, and resumable statements.

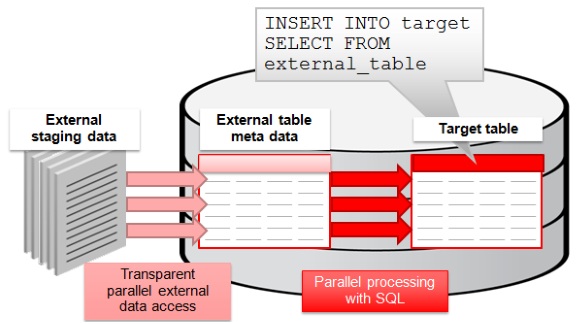
# Oracle Data Load

The main goal when loading data into a data warehouse, whether it is an initial or incremental load, is often to get the data into the data warehouse in the most performant manner. Oracle Database 11g supports several embedded facilities for loading data including: external tables, SQL\*Loader, and Oracle Data Pump.

If you are loading from files into Oracle you have two options, SQL\*Loader or external tables. Oracle strongly recommends that you load using external tables rather than SQL\*Loader:

* Unlike SQL\*Loader, external tables allows transparent parallelization inside the database.
* You can avoid staging data and apply transformations directly on the file data using arbitrary SQL or PL/SQL constructs when accessing external tables. SQL Loader requires you to load the data as-is into the database first.
* Parallelizing loads with external tables enables a more efficient space management compared to SQL\*Loader, where each individual parallel loader is an independent database sessions with its own transaction. For highly partitioned tables this could potentially lead to a lot of wasted space.

Oracle's external table functionality enables you to use external data as a virtual table that can be queried directly and in parallel without requiring, the external data to be first loaded in the database. The main difference between external tables and regular tables is that an external table is a read-only table whose metadata is stored in the database but whose data in stored in files outside the database. The database uses the metadata describing external tables to expose their data as if they were relational tables, so you always have to have the external data accessible when using an external table. Since the data resides outsides the database in a non-Oracle format you cannot create any indexes on them either. Figure 1 illustrates the architecture and the components of external tables.



**Figure 1 Architecture and components of external tables**

## Prepare for data loading

Oracle's best performing approach to loading data in a high-performant parallel manner is through the use of flat files and external tables. An external table allows you to access data in external sources (staging files) as if it were a table in the database. This means that external files can be queried directly and in parallel using the full power of SQL, PL/SQL, and Java. To load data you have to provide the following: an external data file (a.k.a. staging file) in a known format that is accessible from the machine(s) where the database instance(s) runs on, plus two objects in the database: a directory object, creating a logical pointer to a location outside the database where the external data files reside and an external table, holding the metadata of both the data format in the external files as well as the internal database representation of this “table”.

### Database objects necessary for data loading

To define an external table users define Oracle directory objects that contain the paths to the OS directories holding the staging files, log files (information about the external data access is written there), bad files (information about data access errors is written there), and finally in Oracle 11.2 directories where the pre-processing programs reside, as shown in Figure 1.

For example:

REM directory object to point to the external data files

CREATE DIRECTORY data\_dir1 AS '/dbfs/staging/hourly/data';

REM for the log files of the external table access

CREATE DIRECTORY log\_dir AS '/dbfs/staging/hourly/log';

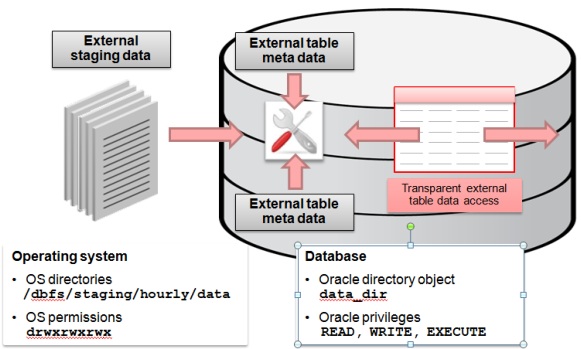
REM for the rejected records of the external table access

CREATE DIRECTORY bad\_dir AS '/dbfs/staging/hourly/bad';

REM directory to point to the pre-processing executable

CREATE DIRECTORY exec\_dir AS '/myloadfiles/exec';

Oracle directory objects are important for Oracle controlled security. Appropriate privileges (READ, WRITE or EXECUTE) must be given to the Oracle directories for a database user to be able to read, write and execute files from them.



**Figure 2 Operating system-database relationship for external tables**

### Create an External Table

An external table is created using the standard CREATE TABLE syntax except it requires an additional clause, ORGANIZATION EXTERNAL. This additional clause specifying the following information (which is necessary to access the external data files):

* Type of the access driver used for reading the external staging files. It can be ORACLE\_LOADER or ORACLE\_DATAPUMP. For data warehousing loading the most common case is using ORACLE\_LOADER.
* Access parameters that define the behavior of the access driver, including the description of the external data format, for example record delimiters, field separators, and default value for missing fields.
* The directory objects that contain the staging files and define the location for the log and bad files for the load.
* The definition of the columns as seen by the database.

### Considerations for the target table

An important aspect for loading - and querying - large amounts of data is the space management used for DW tables. You want to ensure to load data as fast and efficient as possible, while guaranteeing a physical data storage that is not detrimental to the future data access. Space management is either controlled on a table level or inherited from the tablespace level where the table (or partition) will reside later on.

If the redo recovery of the table after load is not critical, we recommend to CREATE TABLE or INDEX with the NOLOGGING clause. This minimizes the redo information generated during direct loads. For an index, Oracle minimizes redo logging only for CREATE INDEX and it logs redo for index maintenance during a direct load INSERT.

Loading large volumes of data always beg the question of whether or not to compress the data right at data load time. However, any kind of compression always imposes additional CPU processing on the loading process. Since loads most commonly are CPU and memory bound, applying compression always has a negative impact on the load performance.

You have to make the decision between maximum performance and immediate space saving (note that compression also provides a benefit for scanning the data since less data has to be read from disk).

For optimal load performance, consider loading the data in an uncompressed format and to apply compression later on in the lifecycle of a table (or partition). Especially with Oracle Partitioning the separation between “older” data and the new data to being loaded is an easy accomplishment: the majority of the older existent data is stored in a compressed format in separate partitions (or subpartitions) while the most recent partition(s) are kept in an uncompressed format to guarantee optimal load performance. For example, you can load the data of the most recent day into an uncompressed partition while the rest of the table is stored in a compressed format; after some time this partition can also be compressed (and be potentially moved to a different storage tier) by using an ALTER TABLE MOVE PARTITION command.

### DML versus DDL

Data can be loaded using DDL or DML operations. In the case of a DDL statement, the target table will be created as part of the load process itself using a CREATE TABLE AS SELECT (CTAS) command. In the case of using a DDL statement, the target table is always empty (since it is not existent prior to the “load”), and there will be no indexes, thus no index maintenance. A CTAS operation does a direct path insert (a.k.a. direct load) where the load process directly writes formatted Oracle data blocks on disk; it bypasses the buffer cache and the conventional SQL layer and is the fastest way of loading data.

Alternatively, data can be loaded into an existing table, using either an INSERT APPEND or MERGE APPEND statement; the target table can either be empty or already contain data, and it can have indexes. Using the APPEND hint invokes a direct load operation for DML. Note that in the case of a MERGE statement, only the insert portion of the statement will be a direct load operation; the update of already existent data will be done using a conventional update operation since you actually change existing records.

For performance reasons it is recommended to load data using a direct load operation whenever possible.

## Analyzing and monitoring your load operation

To analyze a load operation you can analyze it prior to running it, using EXPLAIN PLAN.

There are four important items in an EXPLAIN PLAN for a load statement: whether (or not) a load runs in parallel, the degree of parallelism used, the estimated size of the load, and the distribution method. For example given this load statement:

EXPLAIN PLAN FOR

INSERT /\*+ APPEND \*/ INTO sales

SELECT \* FROM sales\_external;

Its EXPLAIN PLAN may look as follows:

SELECT \* FROM TABLE (DBMS\_XPLAN.DISPLAY);

--------------------------------------------------------------------------------------|Id| Operation | Name | Bytes| TQ |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------------- |0 | INSERT STATEMENT | | 100G | | | |

|1 | PX COORDINATOR | | | | | |

|2 | PX SEND QC (RANDOM) | :TQ10000 | 100G | Q1,00 | P->S | QC (RAND) |

|3 | LOAD AS SELECT | SALES | | Q1,00 | PCWP | |

|4 | PX BLOCK ITERATOR | | 100G | Q1,00 | PCWC | |

|5 | EXTERNAL TABLE ACCESS FULL | SALES\_EXTER| 100G | Q1,00 | PCWP | |

-------------------------------------------------------------------------------------- Note

-----

- Computed Degree of Parallelism is 24

- Degree of Parallelism of 24 is derived from scan of object SH.SALES\_EXTERNAL

Ensure that the actual data load is running in parallel (Id #3). You can do this by ensuring that you do not see a parallel-to-serial distribution before the load actually takes places („P->S‟ in the „IN-OUT‟ column for Ids higher than the load Id). You can also verify a parallel load by having the final „PX SEND QC‟ operation (Id #2) above the load operation.

# Source Books and Articles

1. Powell G. Oracle Data Warehouse Tuning for 10g. Oxford: Elsevier Digital Press, 2005.
2. Kimball R., & Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition. Indianapolis: John Wiley & Sons, 2013.
3. Lane, P. Oracle Database Data Warehousing Guide, 11g Release 2 (11.2). Redwood City: Oracle, 2013.