Enterprise Data Warehouse approach - definition and scope

The Enterprise Data Warehouse (EDWH) approach tries to integrate all in-house Business Intelligence apps into one consolidated Data Warehouse. The goal is to provide a one-stop source in which all relevant business data is available in high quality, independant from the original source systems, in it's whole history and aligned to a common business directory and metadata model.

The primary attraction of an enterprise data warehouse is that all the data is constantly available for analyzing and planning purposes. The alternative is for a business to have different databases for each major branch or organizational division, leading to a complex schedule of data reporting to allow for higher level analytics and planning. An enterprise data warehouse imposes a standard treatment of data and can grow with a business's needs by adding classifications as they emerge in the business model.

Data Warehouses are central repositories of integrated data from one or more disparate sources. They store current and historical data in one single place and are used for creating analytical reports for knowledge workers throughout the enterprise. The data stored in the warehouse is uploaded from the operational systems.

At DNVGL, the EDWH approach is trying to enhance to effects of Business Intelligence by integrating existing on premise BI solutions, the new Oracle BI Cloud (BICS) initialtive and OTBI reporting. Oracle Transactional Business Intelligence (OTBI) is focused on quick delivery of insight on specific Oracle cloud apps data using a custom, closed-shop data model. EDWH is focused on integrating enterprise data independent of source system using a consolidated data model.

Layers and architecture

For data to be used for Enterprise Business Intelligence, it has to be fetched from the source system, integrated, consolidated, aligned and presented to BI tools. This is done moving the data between several architectural layers. This process is called ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform).

STAGE layer

In the STAGE, data is fetched from the source systems. This can be done by using a database connection (database link) or by reading files delivered in XML or other formats. The source system data is imported to the stage layer as it is modeled in the source system. The only addition may be DWH-specifiy metadata columns for each stage table: a column identifying the original source system, and another column that states the time this data was first seen in the data warehouse.

Data is loaded incrementally, if possible. It is present in the stage layer per source system.

In addition to the stage (STG) layer, a pesistant stage layer (PSA) may be established. In this layer, every load iteration is preserved by partitioning using the DELTA DATE column. This enables the ability to reload every iteration of source system data into the subsequent DWH layers.

CORE layer

In a Data Warehouse it is recommended to implement a central Core layer. The core layer is where integration, consolidation and historization are done. To integrate data from various heterogenous sources, it has to be loaded into a common, unified data model. The Core is the only source for the Data Marts that are used for BI applications.

The Core data model can either be derived from the known analytical requirements (which sometimes are very specific requirements for reports) or from the data models of the source systems. These two strategies are known as Reporting-Driven and Data-Driven Analysis.

In an ideal world the Core layer design is neither driven by specific source systems nor by currently known report requirements but by the definition of your business and your overall analytical requirements. With a pure data-driven strategy, the Core model must always be changed whenever the source system changes. With a reporting-driven strategy, you will not be able to fulfill new requirements without changing the Core data model. The data model of the Core should ideally reflect the business processes in the best possible way. This is often a combination of the two strategies. For a stable and flexible Core data model, the current source systems as well as the known analytical requirements should be considered when designing the Core data model.

Most analytical requirements need historic data. Comparison of current data with data of the previous year, quarter or month are quite common to detect trends and necessary for planning and predictions. Additional requirements for storing historical data in the Core are traceability of data changes, repeatability of user queries, or point in time views of historical data.
The impact of the requirements above is that data in the Core must be stored in a way that supports change tracking and data versioning. Existing data must not be deleted or overwritten, but new rows have to be created for changes of existing source records. The behavior can be different for master data (dimensions) and transactional data (facts). Depending on the modeling approach, this can be implemented in different ways.

There are two commonly used approaches for DWH core modeling:

Relational Core Modeling

In this approach, data is consolidated and historized in a form as nearly derived from the original structure as possible. Most data usually is in the third normal form (3NF) when originating from a relational database at the source system.

When the relational Core mode is derived from the OLTP data models, structure changes of a source system require modifications of the Core model. The modeling strategy for a relational Core model is often data-driven or a combination of data-driven and reporting-driven. The Core data model is derived from the source systems, but known business requirements should be considered where possible. The model has to be subject-oriented, not a one-to-one copy of the source tables. Similar data is stored in one place of the Core.

For historization purposes, master data must be versioned in the Core data model. Data versioning could be realized with slowly changing dimensions type 2 (SCD2) like in a dimensional model, but for relationships between entities, this can lead to many different versions when existing data of a referred entity is changed. With the concept of Master Data Versioning, this ripple effect can be avoided. For each entity in the relational data model, two tables are defined:

- The head table contains the identification (business key or primary key of the source system) and all attributes that either cannot change during their lifecycle, or where only the current state is relevant (SCD1). These can also be relationships to other entities, e.g. the relationship from a contract to a client.
- The version table contains all attributes and relationships that can change during their lifecycle, and where the change history is required in at least one Data Mart (SCD2). Each version has a validity range (valid from / valid to) and refers to the entity in the corresponding head table.

A version table is defined when at least one attribute or relationship requires change history tracking.

The separation of head and version tables is required to store independent change history for each entity. Although a foreign key can be static (stored in the head table) or dynamic (stored in the version table), it always refers to the head table of the corresponding entity, not to a particular version.
When existing structures – especially relationships – are changed, the effort for data migration in the Core may be high. For example, it may be simple to add a new foreign key column to an existing table. But what contents must be filled in for the historical data records?

Data Vault Modeling

Data Vault modeling, invented by Dan Linstedt in 1990, is a modeling approach specialized to model the Enterprise Data Warehouse Core layer. This approach is suitable for multi-source environments needing a fast adaptation to changes. The model consists of a complete denormalization (“Unified Decomposition” mentioned by Hans Hultgren) of three base constructs into entities:

- A Hub is the business key defining a core business concept. This key is related to a business entity, not to a particular source system.
- A Link is the business natural relationship between business core entities.
- A Satellite is the contextual information around keys and relations. It contains the descriptive information of a business entity.

This split allows your Enterprise Data Warehouse to have a quicker adaptation to business need, source changes or business rules, though with the disadvantage of an increasing number of structures in the core Data Warehouse.

The core of the model has to be independent from the sources to guarantee the longevity of the enterprise core Data Warehouse and minimize dependencies with the source systems.

The Hub structure contains the business core concepts of a functional area.

Links are the Hub surrogate key associations. There is no limitation for the number of referenced Hubs as long as the association is a business relationship. The physical entity contains the different surrogate keys of the linked Hubs, each in a separate column, and a surrogate key for the association.

The Satellites contain all the information describing the business keys or the associations. There can be many Satellites for one Hub or one Link, the idea is to regroup attributes by frequency of changes or/and by source. Each Satellite entity contains a surrogate key linking to a business key in the Hub or to a Link. The particularity of the Satellite is that the load date is part of Satellite key. Each time an attribute change is loaded, a new row is inserted with the load date of the Satellites insertion. In our example, there are two Satellites for the Hub customer because we regroup attributes by frequency of changes.

The implementation strategy for Data Vault modeling will be to find the core business concepts and the natural business keys in the source systems and then verify that they confirm to the business point of view. The same approach is used for the relations. The flexibility of the Data Vault model is due to the fact that it allows adding objects without a complete redesign of the existing structure and the possibility to have different Satellites for the source systems . This strategy allows tracking of the differences and quality job into the core DWH but postpone the sourcing choice of the attributes to the Data Mart layer. For example, if a new source system, also containing customer entities, has to be loaded into the Data Warehouse, a new Satellite with different attributes can be added to the model. A simple report comparing the two Satellites will show the quality issues to the business.

The ETL jobs to load a Data Vault typically run in two steps: In a first step, all Hubs are loaded in parallel. In a second step, all Links and Satellites are loaded in parallel. The individual ETL operations for each type of objects are:

- Hubs: The business key must appear only once in the Hub; insert with a lookup on the business key
- Links: The association must appear only one time in the Link; insert with a lookup on the association of surrogate key
- Satellites: The loading of the object is the most complex, it depends on the historization mode. Either the changed attributes are just updated, or a new version must be inserted.

In the Data Vault model, historization is located in the Satellites. It looks like SCD1 or SCD2 dimensions, except that the primary key of the Satellite is the surrogate key of your Hub and the load date of the new record in the Satellite. This historization approach is used including the facts (facts are Hubs and have Satellites).

It is possible to have multiple Satellites per Hub/per Link grouping them by frequency of change. The multiple Satellites strategy can become complicated when multiple Satellites are historized, the complexity is to regenerate time slices within multiple historizations. In this case, there is a possible solution called “point-in-time” table, the structure placed between the Hub/Links and the historized Satellites stores the time slice reconciliation and the keys. This reconciliation is heavy: every time there is a change in one of the Satellites, a new record in the point-in-time table linking each Satellite (surrogate key of each Satellites + load date of each Satellites) with the current time slice record is created.

The “Unified Decomposition” offers the possibility to adapt/change/create tables without touching to the core structure of your Data Warehouse.

Data delivery from a Core Data Vault model can generate performance issues, because of the multiple “joins” that are sometimes outer in case all the records are not in the Satellite of a Hub. If the model contains point-in-time tables, the queries to reconsolidate the historized dimensions or facts through multiple SCD2 Satellites is very complex and, again, increase the number of “joins” in the queries.

Recommendations for Core Modeling

Relational Modeling is useful for the Core layer if the analytical requirements are diffuse or not yet defined or if the purpose of the Data Warehouse is the delivery of information to different kind of BI applications and other target systems. In combination with Master Data Versioning, a relational Core gives flexible support of data historization and allows delivering current and historical data to the Data Marts. The ETL processes to load a relational Core are usually easier than for a dimensional Core, but the data delivery to Data Marts can be more complex – especially if the Core model is derived from the data models of the source systems. A relational Core is recommended if the source system structures are relatively stable and if a highly integrated Core data model is required.

Data Vault Modeling is a powerful approach for Data Warehouses with many source systems and regular structure changes on these systems. Adding new data to a Data Vault model is quite simple and fast, but the price for this is a complex data model which is harder to understand than a relational data model. Instead of a full integration, data from different sources is assigned to common business keys, but stored in its original form. This allows simpler ETL processes to load data into the Core, but postpones the integration effort to downstream processes. Hence data delivery to the dimensional Data Marts is highly complex and costly. Data Vault modeling is a recommended method especially for *agile* project environments with short incremental release cycles.

Fundamental for all modeling approaches is that the method is accepted by the DWH development team, and all architects and developers understand the concepts, benefits and issues of the chosen modeling approach.

Warehouse or Data Store?

The effort to model data, using the architure concepts mentioned above, is growing in the pace we consolidate source data in the warehouse. To keep the modeling effort minimal, we can decide to keep data transformations at a minimum and use source system data in it's current form whenever possible. This is the concept of Oracle Transactional Business Intelligence (OTBI) and the BI Apps. Data is taken from the source system and directly delivered to the BI framework, or just treated minimally in a dimensional data mart. In this way, most of the benefits of an enterprise data warehouse (EDWH) cannot be realized, though (like enterprise data governance, commonmetadata directories and enterprise data integration and historization).

MART layer

The Data Mart layer is used for BI instruments like Oracle Business Intelligence (OBIEE) to deliver data for reports, dahboards and analysis. Data is modeled in dimension form as a star schema.

Dimensional modeling is a set of techniques and concepts used in Data Warehouse design proposed by Ralph Kimball, which is oriented around understandability and performance and which uses the concepts of facts (measures), and dimensions (context). Most reporting tools and frameworks require a dimensional model, business users intuitively understand a dimensional model and are able to formulate queries. A dimensional model is a conceptual model rather than a logical/physical data model, since it can be used for any physical form, e.g. as relational or multidimensional database. If a dimensional model is implemented in a relational database, the most commonly used schema types are star and snowflake schema. The latter one normalizes dimension tables, in a star schema the dimension tables are denormalized.

Tools and methodology

Database (cloud approach)

Stage, Core and Mart layers need a relational database (RDBMS) as underlying data storage. Wheras as technically possible with PostgreSQL or other relational databases, practical and corporate considerations suggest to use Oracle RDBMS as data sink.

Business Intelligence and Data Warehouse archtectures within Data Lake concepts are out of scope of this document and considered as not mature enough for efficient BI usage at this time.

Although most DWH databases in the industry are on premise in 2017, Oracles cloud offerings are interesting and DNVGL does have a cloud strategy. Since database versions are current, and even Exadata technology is available, Oracle Database Exadata Cloud Service is the preferred storage technology.

Data Warehousing options needed are:

- Partitioning
- Compression
- Multitenant (if possible)
- Data Guard (for availability)

All options are licensed in Exadata Cloud Service.

If availability of the DWH platform is critical, a standby database can be created using Data Guard. In case of any failure of the primary instance, the standby instance can be brought online in short time.

Data Modeling

Data Modeling should be done using standard tools, e.g. Oracle SQL Developer Data Modeler. Data Modeler can be configured to use a version control system for best collaboration results. Recommended are Subversion or Git (available in the newer Data Modeler Versions).

Extract, Transform an Load data

To stay in the Oracle world, Oracle Data Integrator ist the ETL (or ELT) tool delivering best results with the other technologies used. ODI has the capability to push data to the Oracle Cloud Database Service, and to obtain data from various sources, including Oracle and Non-Oracle databases and files as CSV or XML.

BI (in the cloud)

Although there are several alternatives available on the market (like Pentaho or Tableau), Oracle Business Intelligence still seems to be the market leader and is the standard tool for DNVGL.

Oracle Business Intelligence can be installed on-premise (OBIEE server) or used as Oracle cloud offerings. The BI cloud solution can be ingetrated with ODI and Database in the cloud.

OBI requires a repository to be created with the OBI Admin tool. In this repository, or universe, the data structures to be used in the BI frontend are defined in three layers:

- Physical Layer
- Logical Layer
- Busines Layer

Those layers of abstraction form the user view to the data, as required by the business model which lays beneath.

In addition to drill-down analysis using the OBI frontend, there are dashboards and traditions reports then can be delivered.

Recommended for further reading

- Oracle Database Cloud Offerings
- Oracle BI Cloud Offerings
- Trivadis White Paper Comparison of Data Modeling Methods for a Core Data Warehouse