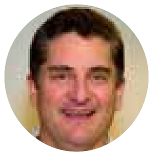
CAMBRIDGE SEMANTICS
THE SMART DATA COMPANY®

ANZO®

Using OLAP Knowledge Graphs for Data Integration and Harmonization



Sean Martin
Cambridge Semantics Inc.

Traditional Enterprise Data Warehouse (EDW) technologies have been used successfully in many settings to integrate and harmonize data so that analysts and users can reliably extract meaning from their large enterprise data-sets. But it is well-established that this approach comes with significant up-front and ongoing costs, enormous risks of failure, and leaves large areas of enterprise data simply unaddressable (due to their complexity). They also lack flexibility, making them too slow to meet the ever-changing demands for data by modern businesses. In this paper, we argue that OLAP Knowledge Graph Technologies (GOLAP) are poised to address these concerns. To explain, we'll briefly review the way EDW technologies are deployed, their limitations, costs, and risks, in order to later contrast how emerging OLAP Graph Technologies have evolved to provide solutions to these issues and are positioned to leapfrog the EDW for modern enterprise OLAP.

On the way, we will also briefly address an inadequate attempt at solving these problems, Hadoop-based Data Lakes and explain how neither of these approaches is fit for the current purpose. Finally, we'll describe GOLAP technology in more detail, showing how it resolves these concerns, and opens up a vista of new data-integration possibilities. OLAP graph technologies and the implementation of broad data integrations, known as knowledge graphs, can link a vast variety of entity types based on many sources of data. We will describe two methods for data integration using GOLAP and an example of a new methodology for data harmonization that is evolving to take great advantage of it.

Introducing Graph Technology

People already know that graph technology is extremely useful because it makes it easy to create, store and query the edges (connections or relationships) that connect vertices (graph nodes or entities – essentially what are presented in RDBMS as records) in a very intuitive manner. The most regularly used examples being the vast graphs representing entities and their interconnections in social networks like Facebook, Twitter, and LinkedIn where the benefit of the connections is self-evident. A key difference between graph and RDBMS databases is that with a graph system it is a simple query that will report all the ways in which two entities are connected (even through multiple vertex hops) whereas with RDBMS it is only possible to formulate SQL queries that test if a specific relationship exists and to repeat that process for each potential relationship to enumerate them.

Another graph technology usage that has entered the mainstream is the "knowledge graph", popularized for example as an improvement to Google's and similar search services, in which often loosely structured but connected information related to the results of a text search are instantly retrieved via a graph database query and displayed in an adjacent supplementary info-box. All of the Wikipedia data has also been turned into a knowledge graph called DBpedia which it is possible to query, and is finding its way into question-and-answer style products. Until recently knowledge graphs have not been widely used for the purposes of OLAP analytics style queries which tend to be complex and usually require reading and acting on far more data than the fast retrieval of a handful of related graph nodes and edges for search result augmentation.

It is also widely, if often vaguely, understood that it is possible for more sophisticated data practitioners to apply relatively "exotic" graph algorithms to extract or calculate all manner of information from data stored in a graph form. Popular examples of these are PageRank (centrality) or calculating how vertices are clustered, or various inference rule techniques where new data is created from existing data, or calculating the shortest path of connections between vertices, etc. Most recently, a vanguard of the wider data science community has begun to discover the predictive value in combining the results of these kinds of algorithms and graph queries as features in machine learning approaches and are reporting positive outcomes. A whole field is growing up around graphs and vertex embeddings, the transformation of graph data to a vector or a set of vectors from which predictive models can be formed.

Graph Technology for Data Integration

A far more fundamental and tactically valuable set of use cases has recently become available to enterprises, namely the extraordinary utility and accelerative nature of data integration and harmonization using graph. This technology has finally matured to the point where it now can perform capably on OLAP Data Warehouse scale data with similar and indeed far more complex JOIN/FILTER style aggregate queries. This new-found performance coupled with a number of attributes inherent to the graph approach finally makes it possible to tackle data integration problems that until now were considered impossibly expensive in time and resources to be viable. Using GOLAP technology to accelerate data integration provides the opportunity for an organization to afford to implement, maintain and extend far more integrations than was previously feasible. In addition, the actual integrations of data as a knowledge graph can be significantly richer or more detailed in their data models than was previously practical.

Enterprise Data Warehouses

It may help to understand both the advantages and the shortcomings of the last generation of technology that formed the basis for almost all sizable data integration projects. This technology is the RDBMS in the form of the hefty and expensive to operate Enterprise Data Warehouses (EDW). If this story including all the shortcomings is already familiar territory, skip ahead to the next section.

EDW systems are either the destination for ETL (Extract, Transform, Load) pipelines targeting data models or schema and are carefully designed to clean and integrate the incoming data set tables from across the enterprise or in many cases they may also be the venue for executing transformative ELT (Extract, Load, Transform) style SQL queries that supplement or supplant those upstream data pipelines' that clean and integrate incoming data. Once integrated by an EDW, data can be queried directly through SQL reporting tools or it may be used to create (and simplified through denormalization) extracts loaded into DataMarts supporting more focused reporting needs or into Business Intelligence (BI) analytics tools for end user-centric analytics.

Once successfully established, an EDW can efficiently and predictably process and query enormous volumes of moderately complex, but structured data. This is the reason they have been such a long-lived and widely deployed technology. In most cases, the data is stored through the application of elaborate data models that necessitate an extensive period of upfront schema design. Using the Relational Model requires that the domains of information across which data integration is desired, be represented in a multiplicity of interlinked tables in which columns represent attributes of the entity type that each table is designed to represent. Additional columns in the table are keys used to establish identifier values through which tables describing all the other entity types of the domain can be joined together in queries that often span the keys of multiple tables. When an entity instance needs to represent attributes that perhaps have multiple values or repeated values, this might also require sub-tables to describe those attribute value instances and are also linked and located by key values. Once the tables with their interlinking keys are in place, users may then design queries by understanding how the tables are arranged and what keys are available to tie them together. The more detailed the data model, usually the more complicated the SQL required to later query across it. Unfortunately, the underlying reality that a relational database is an attempt to model, rarely lines up to fit neatly into interlinked tables.

If this all seems complicated, that is because it very rapidly becomes so. Data modelers have to tackle the problem of data representation at three levels:

- The conceptual model in which the entities, attributes and their relationships (dependencies or associations) are captured at the business level along with an understanding of the reporting needs, the specific questions that the data integration is being constructed to be able to answer.

- The logical model begins the process of mapping a conceptual model to the lowest layer by determining the structure of the data elements (size & type) and sets the relationships between them. This includes representing the data in Third Normal Form (3NF) which ensures as little data is repeated as possible through the use of a cascading table structure that can be more efficiently queried at the cost of additional SQL query construction complexity.

- Finally, the physical model which is where the database software vendor and storage-specific details are determined including establishing the primary and secondary keys, constraints, triggers, indexes, views, etc.

While technically efficient at a time where all of CPU, RAM and disk storage were expensive and relatively slow compared to what is available today, it is not surprising that the trade-offs of model richness (the level of detail captured in the data model) versus what could practically be sustained in a system would usually trend towards simpler data representations as a practical matter.

The other thing to note is how brittle such a database system is in the face of either changing or additional sources of data (e.g. new or altered attributes and/or entity types) or changing reporting requirements (requiring different SQL queries). Even quite small changes to the data model's schema have a rippling effect requiring supporting changes in many other places across the overall system. At a minimum, any change to any table will usually also require changes to any reporting SQL that depends on that table and quite possibly changes in the reporting tools that issued that SQL query. A related set of problems of "slowly changing dimensions" or the strategies needed to manage the historical data as the data model's schema changes can add yet more modeling and maintenance complexity.

The major complaints about the EDW being inflexible and extremely slow to respond to new kinds of data or new business reporting needs are easily explained by the complexity of the upfront and ongoing data modeling required along with the significant effort needed to successfully modify a system in the face of such brittleness. Maintaining an EDW successfully requires discipline and strong governance processes. This is often all the more necessary because the business context and meaning of the data represented in the RDBMS is often stored in a separate system for metadata or even as a collection of documents or spreadsheets and can quickly become stale or out of sync. It also requires a great deal of forethought in the design of the data model as this must include a nearly complete understanding in advance of all the reporting requirements that the business might have for it. Unconsidered questions and data are often very hard to accommodate after the fact without significant rework effort.

Given how expensive and risky these systems are to initially assemble (reportedly a 50% failure rate), along with the high ongoing licensing and operational costs, including the cadre of skilled DBA's needed to maintain, modify and operate it, these systems are put in place mainly to solve the more essential or valuable integration and reporting problems. What is clear is that while the EDW does solve many of the problems of data integration at a technical level, the difficulty in putting that technology to work too often results in failure at the human and business level.

One final point to note about the Relational Model is that it only operates on data from sources that can be organized and coerced into neat tables. This means that the none of vast troves of complex unstructured data (e.g. web pages, emails, slides documents, audio, video, etc.) that reportedly makes up more than 80% of all enterprise data can be integrated using an EDW except in the most trivial ways. The modeling and remodeling that is required to represent the explosion of contained entity types and their interrelationships that today's unstructured data analytics and extraction techniques can uncover is simply too overwhelming.

Hadoop Big Data technologies and the Data Lake are not well suited to data integration

More recently, the practice of simply dumping raw files into cheap shared storage and leaving all the problems of integration to the downstream users of the data took hold in enterprises in the form of the Data Lake. The intention was to provide speedier access to data urgently needed to answer the business's questions through the simple expedient of skipping all the steps necessary for making clean, integrated data available via an EDW. The whole approach can be summed up by the popular dictum "schema on read" as opposed to the far heavier weight "schema on write", which is short-hand for the significant upfront effort required to properly understand, model and stand up data in a data warehouse.

Built on the Apache Hadoop family technologies like HDFS (file storage), Sqoop and Spark (ETL), Hive, Impala, Spark SQL (data access), etc. These technologies once seemed like they might offer organizations a short cut - particularly in more refined editions of the Data Lake that included a metadata catalog and the application of data governance policies making it possible to even find the right data as well as the addition of SQL to support access to it from existing analytics and reporting tools.

However, there are many problems with this methodology that have severely curtailed its successful application. Two particularly related to integrating data are as follows. Firstly data integration is both conceptually and practically difficult, so an approach that relies on skilled users to do all the heavy lifting at the time of arbitrary consumption of raw data, is both a lot less efficient than the EDW model which concentrates the effort into a structured, reusable asset and is even less likely to lead to broad success.

Secondly, although the Hadoop data access tools now all support SQL, the underlying technology was designed by engineers looking to cheaply and efficiently process, in parallel, pipelines of large amounts of relatively simple transactional or web/machine log and web crawl style data using the MapReduce algorithm and commodity scale-out hardware. They did this to both avoid the significant licensing cost of the EDW incumbents and to scale the amounts of data they were processing well beyond the reasonable limits of even the largest most expensive RDBMS systems using a much more practical and affordable approach that scaled up and scaled down as needed using clusters of often cloud-based virtual machines. SQL support was added much later to various components in the Hadoop stack as a means to provide data access to the Business Intelligence (BI) data consumption ecosystem tools (e.g. Excel, Spotfire, Qlik, and Tableau, etc.) used widely in enterprises as they attempted to repurpose their existing data analytics frontend technology while establishing their data lakes to supplement or in some cases attempt to replace their EDWs.

The underlying problem is that a system designed for processing simple but voluminous log file data is not solving the same problems needed to integrate complex data that is perhaps not as massive but is far more varied in types of things it must describe. In particular, SQL's JOIN operations perform badly on these systems. Benchmarks show they are orders of magnitude slower than what is possible in a well-tuned EDW. This in-turn forces radically simplifying compromises in the data models since the only operations that perform joining read operations well enough for practical use are those against a single or a small number of tables.

From a modeling point of view, the data usually stay, or end up, in an unnormalized form that crams the data from multiple entity types into the same or duplicated rows of sparse data in wide tables with many columns or in a scheme that places multiple values into the same column cell (e.g. a JSON string or using value delimiters). In these arrangements, rich data models that accurately represent a complex underlying reality consisting of a wide variety of related entity types are not viable. Data Lake and Hadoop Big Data technologies are poorly suited to replace most of the data integration tasks performed by the EDW as they do not succeed well on either the technical or human levels.

So why is GOLAP so useful for data integration?

Repeating the history of RDBMS, we are now seeing technical specialization and resulting segmentation in the graph database market. For over a decade the only graph systems available were essentially those databases suitable for Online Transactional Processing (OLTP), supporting increasingly fast transactional read and write of adjacent vertices with read performance scale-up through the maintenance of full replica instances of the database. Compared to RDBMS, these systems have proved far less suitable for integration of even modest data volumes, simply because the data load, transformation, and aggregation analytics query performance was generally inadequate to be of practical use for most solutions, and also being essentially single server copies (sometimes replicated to increase query throughput) they could only scale up the total volume of handled data vertically by using increasingly powerful server hardware.

However, in recent years the market has seen the introduction of true Online Analytical Processing (OLAP) graph-data or GOLAP technology and systems specialized to very rapidly load large amounts of data sharded across multiple compute nodes in a similar fashion to their massively parallel (MPP) RDBMS data warehouse antecedents. These new systems are capable of performing well on extremely complex JOIN/FILTER style analytics and transformational queries required for ELT and in-graph Feature Engineering (in support of Machine Learning) over large data volumes that form enormous knowledge graphs. They are now in production with what may prove to be a "killer app" for graph, namely accelerated data integration and harmonization. There are multiple technical and human reasons why this technology is so apt for these purposes:

- ## Intuitive to humans

  The graph model is intuitive to humans as it mirrors how we seem to make sense of the world around us. There are many kinds of things that we know about, these things all have descriptive attributes, some of those attributes are relationships or links to other things that may themselves have descriptive attributes (i.e. property graphs), etc. A graph can be used to far more easily model the world as it presents a natural model essentially combining the best of conceptual modeling and logical modeling, one in which the data remains somewhat like First Normal Form (1NF) sans primary keys, as an abstract Entity-Relationship (ER) model. Graphs can readily describe "ragged" data structures with non-uniform or incomplete data. This means that there is no need to perform physical modeling like one would in an RDBMS in which the real world has to be massaged into unnatural tabular structures that are either wide with many columns containing many record cells with no values or have a multiplicity of tables with significant key-based interlinking complexity that attempts to capture all the complexity of a reality. Modern implementations of graph data representations are therefore more directly understandable to people and entirely uncluttered by model artifacts related to any underlying storage mechanisms. The net effect is that it is far easier to create and manage very rich and complex data models that better represent a "digital twin" of some underlying reality.

- ## Labels, not tables

  Technology writer Dan Woods' phrase "labels not tables" neatly sums up one of the most fundamental differences between the Relational database approach and that of OLAP using graph. When using ETL to move data into a graph database, it is unnecessary to predetermine and then create the schema tables into which to fit the incoming data. Instead, the ETL process simply provides labels for incoming data values that describe the attributes of vertices and their connecting edges. Labels are data not schema, but they can be used like schema to describe the model, and thus they are far more flexible as they can also be created, deleted and manipulated flexibly through queries at any stage of a graph data systems life-cycle. Note that the GOLAP advantage described above applies particularly to RDF and similar triple stores that do not impose rigid schemas. Unfortunately, many modern graph technologies are still very much slaves to the relatively inflexible schema creation and maintenance constraints and naturally still suffer from the related issues written about earlier in the context of RDBMS. In fact, many of these graph stores essential require external ETL integration processes to essential pre-form the knowledge graph they will load by creating individual files containing the various vertices instances corresponding to their schema and separately files containing the edges that connect them. Graph database systems of this kind are not suitable for GOLAP.

- ## Instance data and metadata together at last!

  Another important attribute of graph databases is that every vertex (node - the equivalent of a single instance of an entity) has a unique identifier. Each identifier can be used as the anchor point on which to attach, annotate or connect additional associated data with arbitrarily complex structures and with little to no planning. This, in turn, means that the same graph database has a simple means to store connected together both "instance data" and all the "metadata" (both technical and business-related) that provides its context — in fact, the two become indistinguishable from the point of view of the graph, except through the labels that are chosen to describe them, which is not much of a difference. Any query can reference either or both since they are equally graph data. Additional data about any vertex or collection of vertices may be added at any time without disturbing any existing data or necessitating changes to queries on that data unless it is useful to update the query to take advantage of the new information.

- ## Labels have meaning.

  The choice of labels that have meaning to users can be drawn from business or domain-specific terms to make them easier to understand by the data consumers. In the case of W3C Semantic standards-based graphs, the labels are defined in OWL ontologies as URIs that in turn are associated in the graph with label strings supporting different languages and synonyms. OWL provides the ability to define classes, essentially a formal definition of the graph labels (both attributes and edge attributes) that you would expect to find associated with a particular entity type. In addition, OWL can define subclass or hierarchical style relationships where attributes from a base classes are inherited, making it easy to specialize entity types with their own attributes e.g. subclass entity "employee" can inherit all the attributes of base class "person" as well as adding some attributes or linkages that are specific to the nature of the employee type. Instances of employee can be returned by queries at the level of person, using person attributes, and also at the level of employee using both person and employee type-specific attributes.

- ## Labels themselves are queryable.

  For instance, OWL has a graph representation that allows it's conceptual definitions to be a query-able part of the same graph of data that it describes.

- ## It is simple to grow a graph, Or schema evolution that actually works.

  Unlike the Relational Model, it is simple to grow a graph model incrementally. A new attribute is just a new label, a new entity type is a collection of new labels including the label describing the edge that connects the new entity instances to existing entity instances in the graph. If instance data for a particular attribute on a particular vertex is not present, it is simply not there (there is no NULL). This allows graphs to easily store data at the level of completeness at which it exists. Linkages to existing data can be immediate if there is a shared vertex identifier between the existing and incoming data (usually because the vertex identifier was based on what was formerly a key-value) or using rules that construct new linking edge labels.

- ### Unstructured (or really arbitrarily complex) data no longer ignored.

  Given how easily its conceptual model can be grown dynamically, a graph is a good home for data extracted from textual and other unstructured data sources using NLP and other techniques - additional entity and relationship type label definitions can be grown in the graph organically to store the facts captured from incoming unstructured data. Naturally, this also makes it an ideal venue for integrating data from both unstructured and structured sources as both are treated identically after they have been transformed from their raw sources to a graph form. Analysis of text often results in an explosion of entity types which are very problematic to deal with in the tabular world of the RDBMS, but that a graph can easily handle in full fidelity (without loss).

- ### Metadata drives automated queries.

  The presence of the conceptually descriptive model as part of the graph is used to drive automatic query generation through model exploration. This can often remove the need for a user to learn the graph query language entirely since the complexity and size of the queries that can be instantly generated and responded to by such a system can be extraordinarily powerful. *More often than not, automatically generated queries are orders of magnitude larger than even a highly competent SQL programmer would choose to tackle due to time and difficulty of debugging each query variation. Without this capability, querying sophisticated entity-type rich knowledge-graphs would be extremely cumbersome and would practically limit exploitation of the potential richness provided by the graph model.* As it stands though, this facility alone can provide significant time and resource savings when it comes to creating both ELT and reporting queries, not to mention the vast numbers of feature-engineering transformation queries required for Machine Learning projects.

- ### Multi-graphs support non-linear workflows.

  Most graphs implement a "multi-graph" approach to data storage and query. What this means is that graph data can be segmented or isolated into multiple sub-graphs. An incoming query can be targeted at a combination of one, or more or indeed all of the loaded sub-graphs as the graph in the scope of the query. The isolation of individual sub-graphs is particularly useful during the process of GOLAP style data integration because it allows different data sources and any transformation related to them to be contained in one or more sub-graphs. Furthermore, additional sub-graphs might contain information that links two or more sub-graphs created through the use of ELT construct rule queries. Each sub-graph can be associated with the pipeline query logic needed to produce it as well as an understanding of the dependencies between sub-graphs. In an OLAP graph that performs sufficiently to allow interactivity, this has the effect of allowing a non-linear approach to data integration somewhat akin to multi-track audio or video editing in which all the components can be treated as a whole, in groups or discretely, and at different times.

- **Multi-graphs support access control.**
  Multi-graphs have an additional benefit in that they can be used to define flexible access control across the RDBMS equivalent of multiple linked tables. The scope of any query can easily be manipulated to allow different users' visibility on different integration combinations of the data instead. This approach can be used to avoid the time-consuming process of adding user or group access filter clauses to reporting queries to control data visibility. By delegating more sophisticated access control to the database, the reporting queries themselves can be simplified. A similar approach might also be applied to providing query access to different combinations of historical and current data.

- **Advanced analytics in support of integration.**
  Some GOLAP engine implementations support automated inference, advanced data analytics functions, graph algorithms and data science algorithms (e.g. correlations, estimators, profiling, distributions, etc.) all of which can be used to enhance the process of data integration and harmonization. This can either be in the form of data discovery and profiling and later in the form of rules (including automatically derived rules) that can transform wide-swaths of data quickly or create additional data and data linkages automatically. Multi-graphs are proving to be an efficient means of isolating and managing the life-cycle of derived data, and yet sometimes usefully combining them in the context of graph queries. Since Feature Selection and Feature Engineering in support of Machine Learning activities, require various forms of profiling analytics, understanding of correlations in the data and finally transformations that render data into the numerical and structural forms required for model training, the ability to keep all this processing activity in-graph provides a significant speed-up in the iteration times that Data Scientists need to their work.

Load all the data, ask questions later or data integration turned on its head

One significant difference between the GOLAP approaches to data integration and those of methods that preceded them is that because the graph data model is so malleable it simply follows the naturally understood conceptual model of the data it represents and consequently relatively little time needs to be spent in upfront model design. New users of a graph database are often surprised at how quickly all of the available source data is loaded and then shaped into conceptual models that described it in the way that they think about it. This often happens without much more than cursory thought about what questions they will ask of it. What they learn is that if all the available data is there connected up in the knowledge-graph, then there is a way to query it to answer not only their initial question but also follow on second and third-order, etc. ad hoc questions that they had not previously conceived. And if the data needed is not there to completely answer the follow-up, then it is usually a simple matter to expand the graph model and add it.

This "load all the data and ask questions later" approach offered by GOLAP contrasts starkly against what is needed to establish a successful EDW. There, quite literally, weeks and months can be spent defining the multi-layered relational schema in order to answer pre-defined business questions. Given the level of effort required to create the model for a data warehouse, it is rare that all of the sources of available data are loaded, and consequently, the schema model design and ETL programming efforts are concentrated around loading exactly the data needed to meet the agreed-upon upfront business reporting requirements and little else. New or follow-up questions often require a cycle of re-modeling the schema , new ETL or ELT programming and of course creation of new reporting SQL queries which all adds significantly to the perception of the EDW being extremely slow and expensive to respond to changing business requirements.

Bottom-up data integration and harmonization

The bottom-up method follows a more traditional ETL methodology in that sources are mapped to a target conceptual model and that the mappings may contain advanced transformation logic. A conceptual model essentially defines a template from which the graph labels are drawn. These conceptual models tend to be more formal and may originate from industry or domain interest groups or might be internally developed by enterprise staff tasked with standardization across the organization. This approach is as close to "schema on write" graph-based data integration (targeting triples) gets - technically it is more like creating "conceptual model on write". Using existing ETL runtime environments (e.g. Apache Spark), data can be extracted from individual source tables, each transformed as needed and then emitted with the appropriate graph labels describing each tabular record as an instance of an entity (graph vertex type), which when logically connected together with the output of additional jobs also extracting all the other available table data rows as different entity instance types, altogether they form the overall graph. The output of all of these jobs are usually either immediately loaded or are serialized together on disk, after which they can be loaded later to form the knowledge graph in the graph database. Additional data sources can provide either

instances of new entity types (new labels) or additional instances of an existing type (described using existing labels) depending on what sources must be integrated.

The GOLAP-based ELT approach to data integration

The other, often more popular, method is more akin to “schema on read” in that the business-friendly conceptual models are applied or “bubble-up” after the raw data has been loaded into the graph and usually close to the final consumption of the data. This form of ELT style integration often starts with the automatic load of all the available source data into the graph with little or no transformation or regard for source schema and then uses graph queries to profile, clean, link and transform it to user-friendly models that integrate all the data and makes it intelligible to downstream data consumers. This method of integration is usually extremely successful because it includes an element of data exploration and discovery against what is actually available data, often not too well understood in advance, and wastes no up-front time conceptually modeling and connecting data that does not exist.

RDBMS data can be automatically pulled wholesale into a graph using its catalog metadata to drive the ETL automation. Tables form the entity types, records form the individual vertices often using the relational key as their identifier label, column values are attributes, and foreign key relationships are the edges that connect multiple entity types. JOIN tables can be detected and factored out as they are unneeded in the graph representation. The use of data profiling can help to provide similar automated ingestion of data from collections of CSV files in which potential key attributes must be detected through either overlapping column names and/or data elements.

In practice, both of the ETL and ELT methods of data management may be applied in a single integration project where the bottom-up method is used to integrate better-curated and described, more often reused data sets provided by the central IT function, while the second method is used to add and massage less well-known data in a more ad hoc fashion as well as link together all the data sets ingested using either method.

Top-down data harmonization - an entirely new integration option available via GOLAP

Data harmonization is the effort needed to combine and conform conceptually similar data from multiple sources that have different formats, schemas, naming conventions, units of measurement, etc. in order to eventually compare all the data in an “apples to apples" manner or to use data combinations extracted from the combined data for new applications. It is possible to harmonize data using the bottom-up data integration method described earlier, using a process in which the tables in each data source are individually mapped to the target model, however, this approach becomes increasingly impractical as the number of sources to be harmonized rises.

A good example of this is historical clinical trial data. This is data that was sent to the FDA in order to prove that a prospective drug is efficacious and safe. In most cases, such a data set had this single purpose and is self-contained and complete in and of itself. It contains demographics of the trial patient population, medical test data, dosage information, adverse events, etc. A pharmaceutical company may have thousands of these studies available and might make the decision to combine them all for purposes of quickly being able to extract a cross-study patient population in order to discover new targets for an existing drug or perhaps to research certain patterns of recorded adverse events across trials of drug families. The target model might be the Study Data Tabulation Model (STDM) a fifteen-year-old standard for describing human clinical trial data. A general approach (simplified for clarity) for tackling this problem using the OLAP graph is as follows.

- Extract all available technical and content-related metadata from all the studies as data. This would include directory names, filenames, column names, SAS label names, and any study-specific metadata, etc., all of which must be loaded into the graph. For a few thousand studies this results in many tens of thousands of table names and millions of column names. Load additional domain-specific metadata, where available - for example, medical vocabularies that can be matched on SAS label names.

- Next use interactive analytics to determine rules (queries that construct additional metadata) by which to cluster the study tables into their respective conceptual SDTM domains (or conceptual classes e.g. demographics or dosing etc.) using simple heuristic techniques like matching similar column/label names or overlap with a medical terms known to be related to a particular STDM domain. The rules add graph annotation metadata persisting the discovered classifications.

- Use interactive analytics to similarly create rules that cluster column names within their SDTM domain and assign, via more metadata annotation, the appropriate STDM variable name to each cluster. The objective is to pair every column name with the appropriate STDM variable name. Since there is often a significant overlap in the naming conventions in families of studies, this can quickly be done for a large portion of the data.

- Analytics can be used in repeated iterations to continue to winnow down the list of columns awaiting pairing as well as establish annotations containing the provenance of which rule(s) was applied to any particular column to conform it.

- If any of the studies already have existing mappings or transformations these can be incorporated as rules (ELT queries) that create far more complex pairings that include transformation logic.

- Once this analysis and clustering or pairing is complete, the data in the graph is used to automatically generate instructions for ETL and/or ELT mappings which are used to load and transform all of the study data into the graph and map it to a conceptual representation of the STDM. Further rules are then used to perform transformations to conform differing measurement systems and correct outliers.

- Finally, once all the study data has been harmonized it can be exported from the graph as standard STDM tables suitable for loading into an RDBMS or used directly from the OLAP graph for interactive analytics.

CAMBRIDGE SEMANTICS
THE SMART DATA COMPANY®

ANZO®