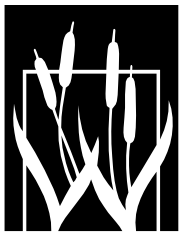

Whitemarsh
Information Systems Corporation

Database Schema Management

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: Whitemarsh@wiscorp.com
Web: www.wiscorp.com

Table of Contents

1. Objective

The objective of this ***Whitemarsh Short Paper*** is to present an approach to managing existing database schemas in the eventual hope of creating enterprise-wide semantics. ***Whitemarsh Short Papers***, all available from the Whitemarsh website at:

http://www.wiscorp.com/short_paper_series.html

These papers all implicitly start with the premise of a clean slate. While those papers are all important and contain necessary concepts and techniques for developing and managing data-based semantics in the enterprise, this paper is for the 99.99% of us in organizations that have an existing quantity of databases. This paper presumes that readers have downloaded, read, understood, and generally agreed with the concepts and techniques contained in these short papers.

Databases were first created to bring together data from the vast sea of standalone files. Eventually, many if not most of the stand alone files disappeared and data processing through database management systems (DBMS) became the norm.

Now we have vast seas of standalone databases. In response, data warehousing was born. These are not necessarily bigger databases. Rather, they are cross functional databases. Bill Inmon began this effort in a formal way in the late 1980s or early 1990s. Born somewhat later was the Operational Data Store. Recently, collections of reference data and other multi-use data have been collected into a new concept, Master Databases. Across the years this new class of databases was commonly known as the "authoritative data sources." One large international corporation called this data its "Strategic Data." New name for the same goal or objective.

To manage the extraction, transformation and loading of data from all these disparate sources, another whole IT "industry," ETL (extract-transform-load) was born in the middle to late 1990s. ETL environments, however, do nothing for discordant semantics. They remain as they are, discordant. Each ETL process represents a point-to-point interface in which data is extracted, transformed to the receiving database's semantics, and then loaded. While ETL environments store all the transformation maps, there is no attempt to build higher levels of common semantics. As new databases are created, they are likely created with their own semantics appropriate to their unique collection users of the database. Additional ETL maps are then required to be created to integrate this new stovepipe of semantics into the environment. As these environments grow the cost of their maintenance grows as well. A study by the United States Air Force in the middle 1990s shows that the DoD Agency was spending more than $167 million per year for just interface development and maintenance. Expensive? Yes. Real value to the enterprise? None.

Most recently, another "silver bullet" has been created, XML, which has largely been accorded the honor of "curing all manners of diseases, and even world hunger." Some have even chided, "it doesn't matter the question, the answer is XML." XML is really just another form of ETL. Systems that are needed to interface with other systems have special modules created that

transform data extractions into a technology independent form that exists in two parts: XML schema, and XML data streams. This process is called "Composition." Systems that are to receive this XML formatted data need to have specially created inverse processes that read and decompose the XML data streams given the semantics of the XML schema and transform the data most commonly into DBMS table formats. This process is called "Shredding." Again, like ETL, the semantics of the sending and receiving systems is not changed, improved, and the like. All the costs and troubles associated with ETL are merely transformed into XML environments. Again, expensive? Yes. Real value to the enterprise? None.

What none of these activities, architectures, or "silver bullets" do is address the problem of discordant semantics that are rampant throughout our databases. Nor do any of these activities or techniques create inventories, transformation strategies, or address the elimination of these semantic problems that have slowly and surely been created in ever larger quantities over the past 50 years.

If this paper took the position that "*here's* the silver bullet" then that would be a lie. Rather, this paper takes the position that "here's an approach" that coupled with clear thinking, hard work and semantic consensus building that can lead to reduced semantic discord. What a good approach, backed up by a robust sophisticated infrastructure, coupled with clear thinking and hard work will do is make an impossible job possible, and make an infinite workload finite. What is clearly recommended is that you start now. Waiting until later only makes the semantic abyss deeper and wider; more costly to fix, and finally, increases your organization's existence risk.

2. Topics Covered

The topics in this paper include:

- What is a database Schema?
- Where are all the variants
- Reverse Engineering
- Forward Engineering

3. What Is a Database Schema?

A schema, the creation of database management systems (DBMS), acts as a container for defined tables, relationships among tables and other schema objects. There is a large quantity of different classes of objects (that is not to imply "object oriented") that can be represented in a schema. Included are data types (simple and complex), assertions, triggers, stored procedures, views and the like.

Schemas are completely encapsulated. Simply, that implies that there's no mechanism for schemas objects from one schema to interoperate with schema objects of another schema. That's just the nature of things. The ISO/ANSI SQL standard for schemas has no capability for interoperability among schemas. Nor is any such capability contemplated. Encapsulated as well is all the semantics of the data represented by the tables and any of the schema objects. That means that if there are exactly same-named and defined schema objects that mean something different there's no ISO/ANSI SQL facility that can either "know" about this nor do anything about it. There's no "intersection" mechanism nor any thing that can store, process, or interrelate schema commonness or differences.

There is another ISO/ANSI Standard for data element metadata. It is commonly called by its ISO standard's "number," 11179. This is a standard for defining metadata about data elements. There is nothing in that standard, however, that binds it to the ISO/ANSI SQL standard. Nor is there anything in the ISO/ANSI SQL standard that binds it to the ISO 11179 standard. There is, though, a common understanding among data management professionals that the semantics of a data element "should" be applicable to all columns in all tables of all schemas that are intended to mean the same as what is defined in the ISO 11179 data elements. Because there is no binding mechanism, only good engineering and best practices ensure that the specifications represented by one standard are appropriately represented in the other.

Another problem within the ISO/ANSI SQL standard is that there is no concept of data element. In SQL:1992 there was the concept of "Domain," which could be considered the equivalent of a data element. Domain was not only deprecated in SQL 2003, but it was also not universally implemented by all DBMS vendors.

Because of the lack of integration between the ISO 11179 and SQL standards there must exist mechanisms outside the scope of both standards that can hold metadata such that:

- It is compliant to each standard.
- Can be interrelated within multi-standard compliant repositories.
- Which can be exported from the repository and imported into the SQL standard compliant DBMSs.

That's the best there can be, at this time given the current state of IT technology capabilities and IT standards.

4. Where Are All the Variants?

Schema variants come in a variety of forms. Included are:

- Same schema different vertical slices of data
- Summarized data schemas
- Common table schemas

- Compatible but slightly different design schemas

The first type of variant is easy to perceive. The schemas are exactly alike but the data contained in one of the schema's databases consists of "East Coast" data while another database only contains "West Coast" data. In this case the databases are semantically congruent. Vertical data divisions could also related to "years of data." One database have this year's data and another database have last year's data.

The second variant class of database, summarized data schemas, might be equivalent schemas but they would only have summarized values. In this case a series of data summarization queries is run against the source databases and the resultant summary values are stored in the summary database. A good example could be a Sales Summary database that would have Total sales for every week by Customer, or Division, etc. The unsummarized data would be in one database and the summarized data in another. In this situation the schemas are generally equivalent. The differences would mainly exist in a modifier assigned to columns names such as Sales vs. Weekly Sales, and the granularity of the data. Other forms of granularity could exist like a database that contains only the rolling-month (last 30 days) of data.

A third variant class of database, common table schemas, could contain tables of data that are intended to not only be semantically equivalent but also exactly the same as to data value sets. Such database tables might contain the data for products that would be needed for customer sales records, ordering, inventory, product development, and marketing. In this case the common tables could be a well-engineered collection that must be exactly the same in all databases within which this data is used. Another common table variant might just be the corporate calendar that would contain the definitions of weekends, company holidays, and local country or community holidays.

The fourth variant class of database, compatible but slightly different design schemas, often occurs in the creation of acceptable performance databases. Sometimes there's only so far a database designer (or database administrator) can go with hardware expansion, data clustering, alternative index creation, buffer management and other performance tuning measures before a database redesign becomes imperative. During the database redesign, summary data may be created, or denormalization may occur.

Denormalization generally refers to the process of re-inserting factored data, or of making data that commonly exists across multiple rows of one table exist in one row. In the first case, it may be bringing back into an address the City, State, and Zip values rather than having a foreign key to the factored City-State-Zip row of data. In the single versus multiple rows, there may be three addresses for a given customer for headquarters, buyer, and delivery. In a "normalized" database design there would be an address table and there would be three rows of data in a Customer-Address table that would intersect address and customer with an Address Type (e.g., headquarters, buyer, and delivery values) column in the Customer-Address table. Database processing would require joining and accessing those two tables onto the Customer table to then get all the addresses. In a denormalized design, there might be three sets of address columns in the Customer table, one set for Headquarters, another for Buyer, and a third for

Delivery. This would have the performance effect of eliminating the joins of the two additional tables. While such a design would be more efficient in terms of processing, it would significantly more expensive if there was an additional address type, say one for Returns, or Invoicing, or Payments. Once however a database design becomes stable, the value of having all the design change flexibility definitely wanes.

The key point to all these variants is simply this. Are these really different databases or are they just different variants. Clearly they are the latter. What is necessary then to manage the semantics across all these variants is a common layer "above" them. The remaining sections of this paper set down and describe the process necessary for managing a large collection of schemas with the ultimate goal of having an integrated set of semantics across the enterprise.

The value of building these "upper" layers is to then have a foundation for forward engineering new database schemas which automatically have the enterprise semantics that promotes data interoperability. The processes that follow presume the existence of a metadata repository environment that fully supports these processes such that work not only has improved quality but also is accomplished faster with the side benefits of lowering risk and increasing productivity.

5. Reverse Engineering

The fundamental process of reverse engineering involves:

- Importing schemas from databases
- Discerning common data model semantic layers and creating them
- Creating abstract data model templates
- Creating ISO 11179 data elements

There are two fundamental goals for reverse engineering:

- Create a metadata infrastructure than enables commonness across collections of schemas or across collections of schema objects.

- Crate a common semantic infrastructure that can then be used to manufacture new database schemas based on common understanding and semantics.

Figure 1 presents an over depiction of the reverse engineering process. Fundamentally, the first step of the process involves importing the collection of database schemas that are involved in the re-engineering process. These schemas are all imported into what is called the Operational Data Model area. This term is used to merely indicate that these are all schemas of operating databases.

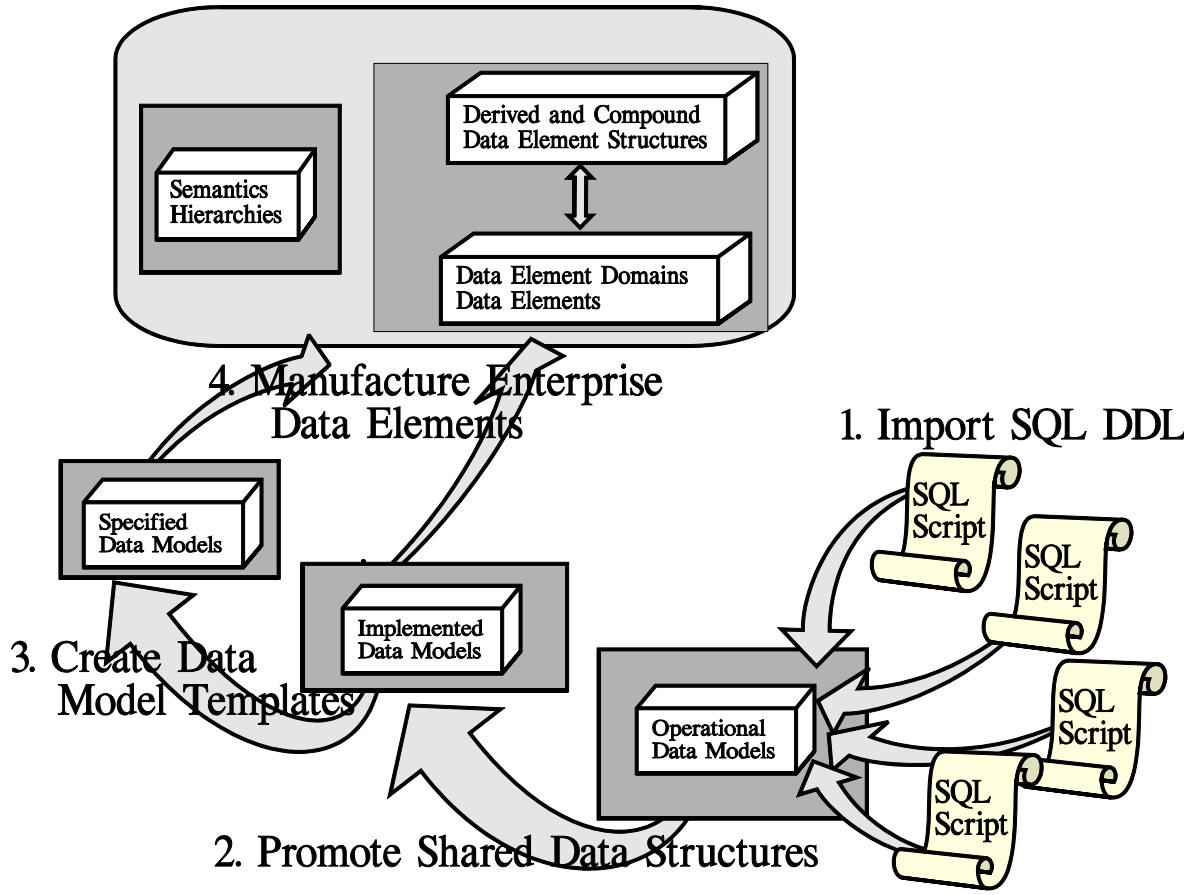

Figure 1. Reverse engineering process.

These schemas need to be imported because in ISO/ANSI SQL there is no mechanism for "seeing" across collections of schemas or even collections of schema objects within different schemas. It would only be within the scope of one special type of database, a metadata repository database that common semantics could be discovered among the differently-named schema objects, and discordant semantics could be discovered among the same-named schema objects.

A common approach to discovering common semantics is to first isolate the schemas by functional area. For those schemas that fit into one functional area, create a schema at the implemented data model level and then promote up to that level the largest and most comprehensive of the functional schemas. If, at this implemented level, the names of the tables and the columns are in "IT speak" then these names should all be changed to "Business Language" speak. Additionally, definitions at the implemented level should be reviewed and if in "IT speak" they should also be changed to "Business Language" speak. Each model at the

implemented data model is a schema based model as is the operational data model. That means that implemented data models are containers for tables, columns, and other schema objects.

If there is a collection of tables within a specific operational schema that are cross functional, as would happen with data warehouse databases, then tables from these databases can be placed in different implemented data models. Just create the different implemented data models schemas and then promote the individual tables from the operational data model to the appropriate implemented data model. Of course, if names and/or definitions are in "IT speak" then they should be changed to "Business Language" speak. What is critical to know is that while you now have "Business speak" names, you still will have all the "IT speak" names in the operational data models.

Eventually there will exist a collection of functional data models that are all defined with business-based semantics. That's the objective. In addition to the creation of business-speak implemented data models, what is automatically maintained are relationships between the implemented data models and the operational data models. All the implemented data models should be in third normal form.

As each operational data model is reverse engineered, there will be an increase in the inventory of common semantics across the operational data models.

Once all the operational data models are reverse engineered into implemented data models the next step can begin, that is, the creation of data model templates. A data model template is a collection of entities and attributes within a specifically named subject area. Note, subjects are not schemas. Thus, subjects are not containers. Data models at the specified data model level are very different from models at the implemented and operational levels. All tables at the implemented and operational level must reside within their respective schema. There cannot be relationships across schemas because tables must be completely encapsulated within their schemas. In contrast, there can be relationships between entities across different subjects within the specified data model level.

To build the specified data models, the individual tables within implemented data models should be examined to discover the columns that are of the same basic subject area. An example might be address data. And, there could be multiple types of addresses, business, residential, and the like. The address subject area or nested subject areas should be created. Then, either entire tables, collections of columns, or individual columns should be promoted to an entity within a subject area. Other examples of subject-based data are persons, locations, assets both real and abstract, information technology, events, financial instruments and the like. The ultimate goal is to allocate all columns of all tables to subjects and entities within subjects. The result will of this effort will be to maximize the commonality of the various types of data that are ultimately employed to define columns and tables within implemented schemas.

Specified data models enable a "where used" approach to the subjects, entities, and attributes across the implemented and operational data models. Reports can be generated that show how the same semantics are deployed across same-named and different-named schema objects. Similarly, reports can show how different semantics are deployed across same-named schema objects.

The final step of reverse engineering is building the ISO 11179 data element metadata. Once complete all the necessary layers to then understand common semantics across the enterprise will be in place. A data element represents a context independent business fact semantic template that can be used as the semantic source for attributes of entities or columns of tables. Data elements are also supported by additional levels of concepts, conceptual value domains, value domains and associated values, and finally data element concepts. Another final component in the Whitemarsh data element model are semantic hierarchies that enable the words, phrases, definitions, and abbreviations that are critical in the automation and maintenance of data element, attribute, and column names, definitions and abbreviations.

When all this data model reverse engineering work is accomplished, the metadata repository with respect to data model metadata, will contain metadata across the models depicted in Figure 2. Also shown in Figure 2 are the functional responsibilities. In general, the organization responsible for data administration and enterprise data would be charged with developing and maintaining the top model of semantic hierarchies, derived data and data elements. Functional data administrators are responsible for the development of the specified

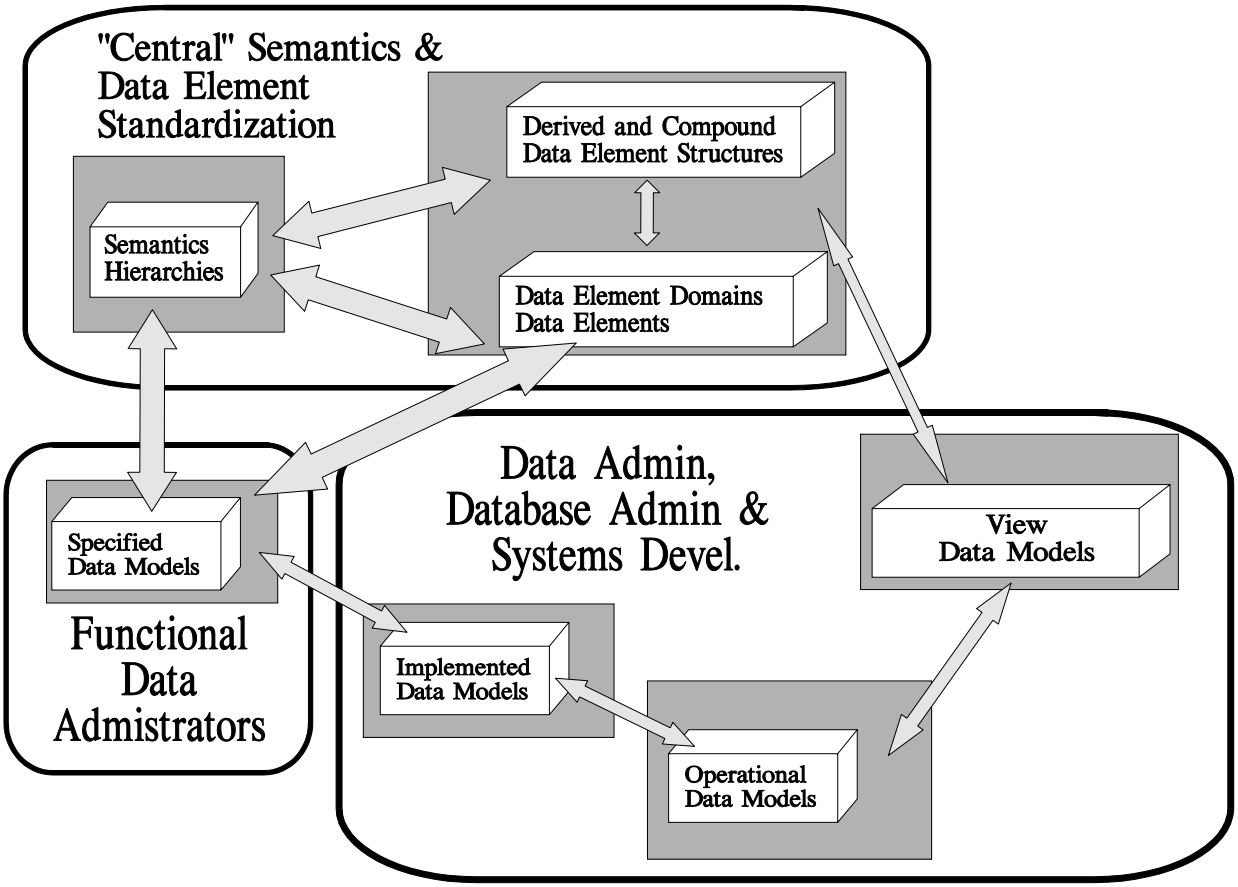

Figure 2. Metadata across data models and functional responsibility.

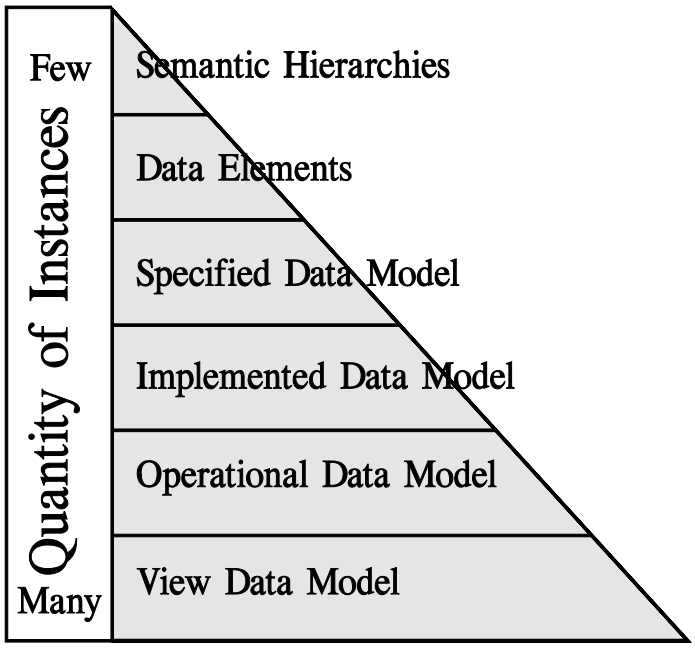

Figure 3. Relative quantity of metadata across metadata models.

data models which would then be used to define the schemas, tables and columns in implemented data models and operational data models. The implemented and operational data models are the proper provinces of data administration, database administration, and systems development. Each organizational group is responsible for the maintenance of their metadata. Additionally, the division of labor is such that the metadata can be created independently and associated in a federated manner. When all the metadata is finally collected, the volume of metadata across the models is illustrated in Figure 3. The largest quantity of metadata is contained in the operational data models, which largely already exist. Metadata at the implemented data model level is less in quantity and is largely built through metadata repository automation assists and renaming efforts. Smaller quantities exist at the specified data model and data element levels. The only real original metadata creation efforts are within the specified and data element, and semantics hierarchy models. The quantity of analysis and creation time is similar to the quantity proportions but is greatly mitigated as set out in the Forward Engineering section that follows.

6. Forward Engineering

Figure 4 presents an over depiction of the forward engineering process. This process is generally similar to that which exists for reverse engineering but with one great distinction. Virtually all the metadata needed for an implemented and/or operational data model will already exist. That

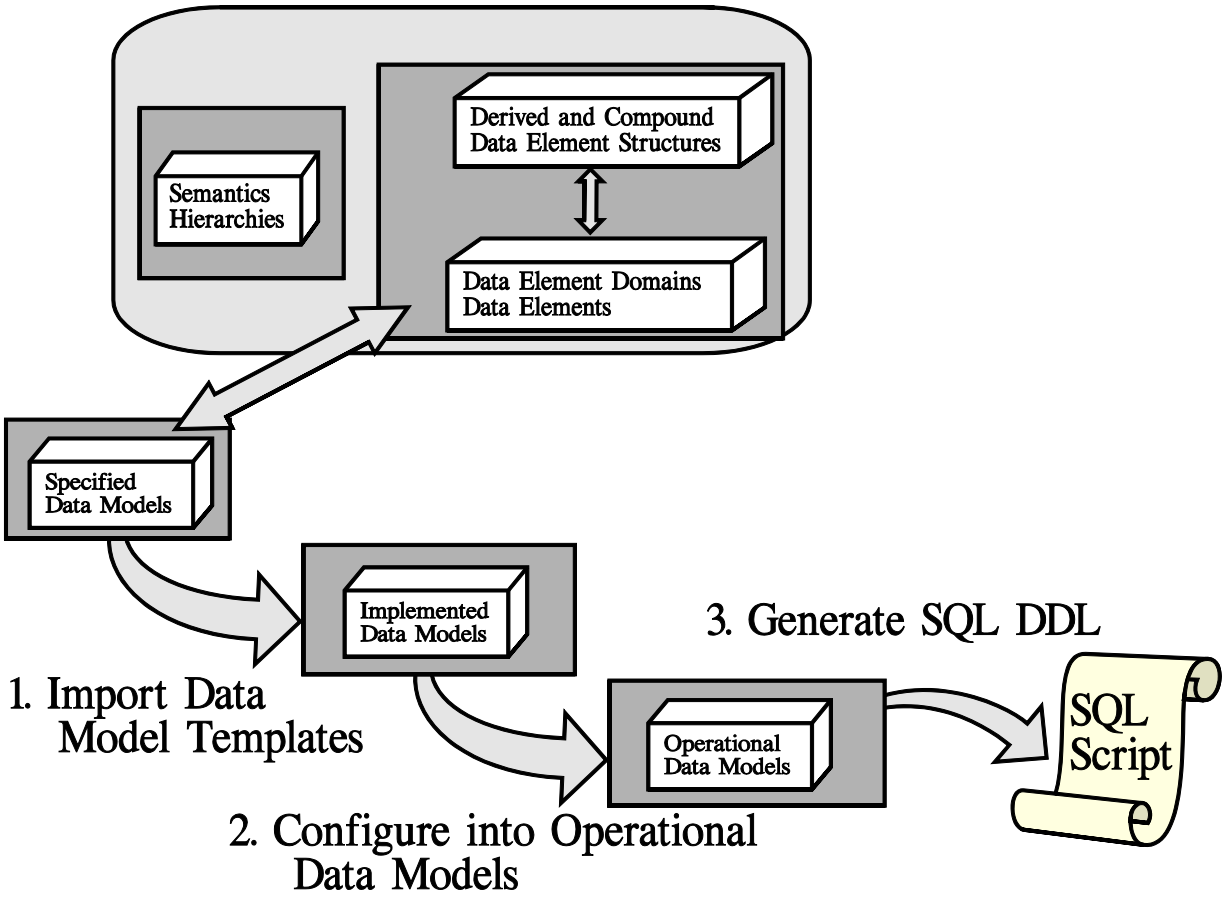

Figure 4. Forward engineering process.

means that when it took months to create an implemented or operational data model will likely take weeks or less if the required metadata foundation already exists.

When an organization desires to create a new database schema, it names the schema and then imports data model templates from the specified data model to the implemented data model. What can be imported are whole entities, entity collections, and individual attributes. Automatically created are all the relationships between the specified data model attributes and the newly created implemented data model table columns. The only thing left for a data modeler to do is to create the relationships among the tables. Automatically created are all the names, abbreviations and definitions. If value domains are associated with data elements or attributes then these are automatically associated with the implemented data model table columns. If more restricted value domains are desired, they merely have to be created within the context of their "parent" value domains and then associated with the columns.

The whole process is simple and straight forward. Creating the operational data model is similar to the creation process of the implemented data model. If a data warehouse database is

needed to be created then the process is the creation of the operational data model schema, and the importation of the various tables from the different implemented data model schema tables that are to be the source of data for the operational data model. Again the process is simple and straight forward. What used to take months can now take weeks.

7. Conclusions

The practical points made in this paper include:

- Data interoperability starts with legacy systems and their associated databases.
- There are many databases that are really just variants of other databases.
- Common semantics can only be discovered and managed within metadata repository databases and supporting metadata management systems.
- Reverse and forward engineering are both straight forward, common sense approaches.
- Over time, semantic maps across discordant semantic databases will be replaced with forward-engineered common semantics if this approach is followed.
- And finally, there are no silver bullets. Stop looking and get to work. In the end it will be faster and cheaper.

8. References

The following references to Whitemarsh materials provide a more detailed exposition practical application of the significant content of this paper.

The following documents are available free from the Whitemarsh website:

Paper	URL
Comprehensive Metadata Management	http://www.wiscorp.com/ComprehensiveMetadataManagement.pdf
Metabase Overview	http://www.wiscorp.com/Metabase.zip

Whitemarsh Data Modeler, Architecture and Concept of Operations	http://www.wiscorp.com/MetabaseDataModelerArchitectueandConceptofOperations.zip
Metabase User Guides	http://www.wiscorp.com/MetabaseUserGuides.zip
Reverse and Forward Engineering Guide	http://www.wiscorp.com/MetabaseReverseAndForwardEngineeringUsersGuide.zip

The following documents are available for Whitemarsh Website Members. The URLs that follow provide descriptions of the pages. Members should log in and proceed to the appropriate page, e.g., Enterprise Database, find the book, paper, or course and perform the download.

Paper	URL
Data Management Program - Metadata Architecture For Data Sharing Data Management Program - Database Interface Architectures Data Management Program - Projects And Data-Asset Product Specifications	http://www.wiscorp.com/wwmembr/mbr_products_edb.html