



Whitemarsh  
Information Systems Corporation

Whitemarsh Metabase:  
Data Modeler Architecture  
Et  
Concept of Operations

February 2006

Whitemarsh Information Systems Corporation  
2008 Althea Lane  
Bowie, Maryland 20716  
Tele: 301-249-1142  
Email: [Whitemarsh@wiscorp.com](mailto:Whitemarsh@wiscorp.com)  
Web: [www.wiscorp.com](http://www.wiscorp.com)

## Table of Contents

List of Figures .....	v
Forward .....	vii
1.0 Why Data Standardization Is Important .....	1
1.1 Areas of Data Standardization .....	4
1.2 Data Architecture Classes .....	6
1.3 ISO and ANSI Data Management Activities .....	8
1.4 Bulk Data Loading or Importing .....	8
1.5 Business Fact Standardization Problems and Solution Approach .....	9
1.6 Data Modeling Problems and Solution Approach .....	10
1.7 View Data Model Problems and Solution Approach .....	12
1.8 Integration with Database Objects .....	13
1.9 Whitemarsh Data Modeler Graphics .....	13
1.10 Whitemarsh Data Modeler, a Difference in Kind .....	13
2.0 Whitemarsh Data Modeler Architecture & Concept of Operations .....	15
3.0 Data Semantics .....	17
3.1 Semantic Modifier Hierarchies .....	17
3.2 Data Use Modifier Hierarchies .....	21
3.3 Meta Category Value Types and Meta Category Value Hierarchies Summary .....	21
4.0 Data Element Model .....	24
4.1 Concept .....	24
4.2 Conceptual Value Domains .....	25
4.3 Data Element Concepts .....	26
4.4 Value Domains .....	27
4.5 Data Elements .....	29
4.6 Data Element Classifications .....	31
4.7 Compound Data Elements .....	31
4.8 Derived Data Elements .....	33
4.9 Data Element Model Process Model .....	34
4.10 Data Element Model Summary .....	35
5.0 Specified Data Model .....	37
5.1 Subject Area, Entities and Attributes .....	39
5.1.1 Subjects .....	39
5.1.2 Entities .....	40
5.1.3 Attributes .....	42
5.2 Key Support .....	45



5.2.1	Primary Keys .....	46
5.2.2	Foreign Keys .....	48
5.2.3	Referential Integrity .....	50
5.2.4	Referential Integrity .....	51
5.3	Attribute Value Domains .....	51
5.4	Reverse Engineering .....	52
5.5	Specified Data Model Maintenance .....	53
5.6	Specified Data Model DDL and Graphics .....	54
5.7	Specified Data Model Summary .....	54
6.0	Implemented Data Model .....	56
6.1	Forward Engineering .....	59
6.1.1	Schema Creation .....	59
6.1.2	Table and Column Creation .....	60
6.1.3	Table Maintenance .....	61
6.1.4	Column Maintenance .....	62
6.1.5	Relationship Maintenance .....	65
6.2	Original Creation .....	65
6.2.1	Schema Creation .....	66
6.2.2	Table Creation .....	66
6.2.3	Column Creation .....	66
6.2.4	Mapping Columns of Tables to Attributes of Entities .....	70
6.2.5	Key Support .....	72
6.2.5.1	Primary Keys .....	73
6.2.5.2	Foreign Keys .....	75
6.2.5.3	Referential Integrity .....	77
6.3	Reverse Engineering .....	78
6.4	Column Value Domains .....	79
6.5	Implemented Data Model DDL and Graphics .....	79
6.6	Implemented Data Model Summary .....	80
7.0	Operational Data Model .....	82
7.1	Forward Engineering .....	85
7.1.1	DBMS Schema Creation .....	85
7.1.2	DBMS Table Creation .....	85
7.1.3	DBMS Table Maintenance .....	87
7.1.4	DBMS Column Maintenance .....	88
7.1.5	Key Support .....	88
7.2	Original Creation .....	89
7.2.1	DBMS Table Creation .....	89
7.2.2	DBMS Column Creation .....	89
7.2.3	Key Support .....	90
7.2.5.1	Primary Keys .....	91
7.2.5.2	Foreign Keys .....	91



7.2.5.3	Referential Integrity .....	91
7.3	Reverse Engineering .....	91
7.3.1	DBMS identification .....	92
7.3.2	Database Identification .....	92
7.3.3	External File Identification .....	92
7.3.3	DBMS Schema Creation through Relationship Creation .....	93
7.3.4	Mapping DBMS Columns of DBMS tables to Columns of Tables .....	93
7.4	DBMS Column Value Domains .....	94
7.5	Operational Data Model DDL and Graphics .....	95
7.5	Operational Data Model Summary .....	95
7.6	View Data Model .....	96
8.0	Summary and Conclusions .....	98
8.1	Semantic Hierarchies and Data Element Model .....	100
8.2	Specified Data Models .....	100
8.3	Implemented Data Models .....	101
8.4	Operational Data Models .....	102
8.5	View Data Models .....	103
8.6	Approach Summary .....	103
Appendix 1 Referential Integrity Actions .....		105
Index .....		109



## List of Figures

<b>Figure 1.</b> Typical metadata statistics for databases. ....	1
<b>Figure 2.</b> Costs and Benefits from Data Standardization Alternatives .....	2
<b>Figure 3.</b> Enumerated code values for a collection of healthcare tables for Yes No columns ...	3
<b>Figure 4.</b> Areas of the Knowledge Worker Framework Affected by Data Standardization and Data Modeling .....	5
<b>Figure 5.</b> Data Architecture Classes .....	7
<b>Figure 6.</b> Meta Attributes for Context Independent and Dependent Business Facts .....	10
<b>Figure 7.</b> Interrelationship among the six models that comprise data modeling. ....	15
<b>Figure 8.</b> Meta Category Value Type and Meta Category Value Hierarchies .....	17
<b>Figure 9.</b> Meta Category Value Types and Meta Category Value Hierarchies .....	19
<b>Figure 10.</b> Meta Category Value Types and Meta Category Value Hierarchies .....	20
<b>Figure 11.</b> Two Methods of Binding Semantics into Tables and Columns of Operational Databases .....	20
<b>Figure 12.</b> Meta Category Value Type and Meta Category Values .....	22
<b>Figure 13.</b> Meta Category Value Type and Meta Category Value Hierarchies .....	22
<b>Figure 14.</b> Data element meta model. ....	25
<b>Figure 15.</b> Data element concept, personal compensation. ....	26
<b>Figure 16.</b> Value domains, associated values and assignment to data element. ....	27
<b>Figure 17.</b> Context Independent Data Element Salary with Associated Meta Category Value Types, Meta Category Values, Data Element Domain, Business Domain, and Common Business name. ....	30
<b>Figure 18.</b> Meta data records required to represent the compound data element, part number .....	32
<b>Figure 19.</b> Data elements involved in derived data element, Age at Death .....	33
<b>Figure 20.</b> Specified data model meta model. ....	39
<b>Figure 21.</b> Subject Area and Associated Entities .....	40
<b>Figure 22.</b> Fully Selected Semantics for Hourly Wage Attribute of Employee Entity .....	43
<b>Figure 23.</b> Full Associated Semantics for Hourly Wage Semantics with Employee Entity ...	44
<b>Figure 24.</b> Meta data records that define primary key for Project Assignment Entity .....	47
<b>Figure 25.</b> Data Model for Employee, Project, Role, and Project Assignments .....	49
<b>Figure 26.</b> Meta data records required to support primary and foreign key relationships among Employee, Project, Project Assignment, and Role. ....	50
<b>Figure 27.</b> Value Domain for Hourly Wage Attribute of the Salary Entity .....	52
<b>Figure 28.</b> Implemented data model meta mode. ....	58
<b>Figure 29.</b> “Pruning” cases for table maintenance .....	61
<b>Figure 30.</b> Entities that are transformed into tables of an implemented data model .....	61
<b>Figure 31.</b> Stand alone, Primary, and Foreign Key Columns. ....	63
<b>Figure 32.</b> Primary and Foreign Keys in Work. ....	64
<b>Figure 33.</b> Associating a column to a data element. ....	67
. Additional semantics associated with the hourly wage column. ....	68
<b>Figure 35.</b> Project Assignment table with surrogate primary key .....	73



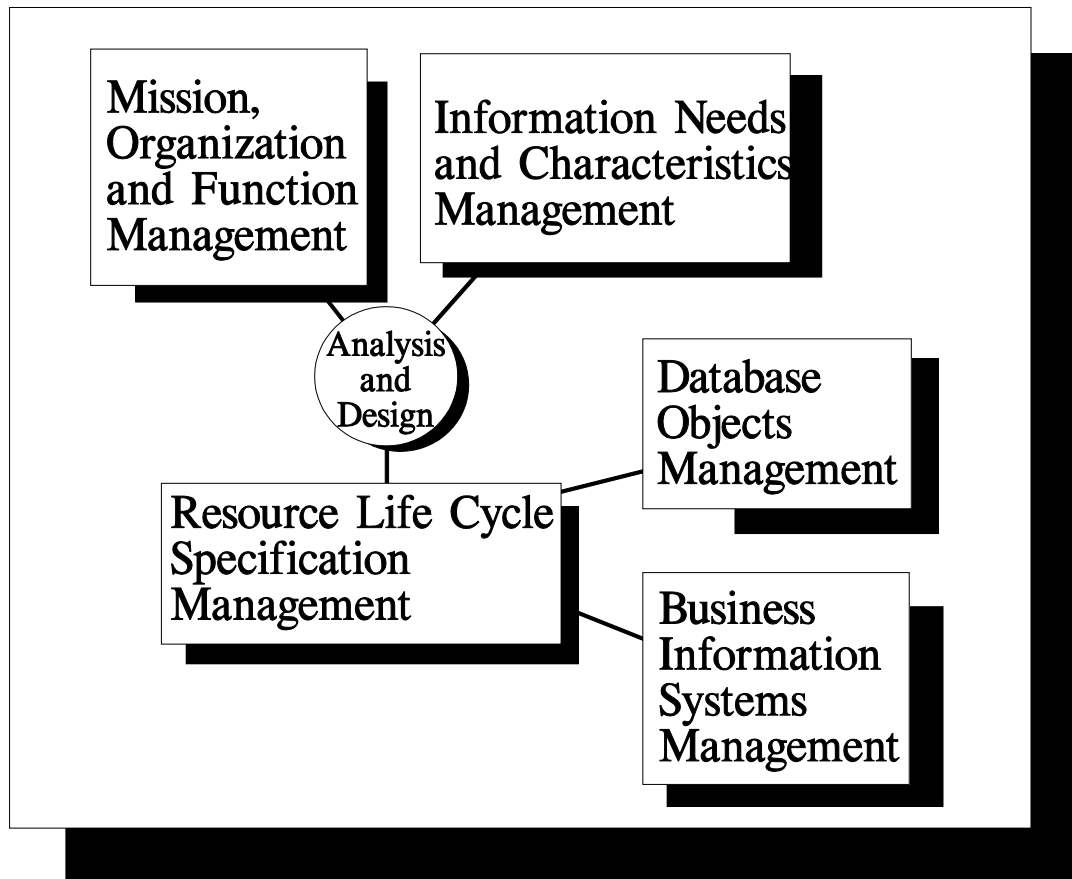
<b>Figure 36.</b> Meta category value type and meta category values allocated to Identifier data element. ....	75
<b>Figure 37.</b> Metadata required to support surrogate primary and foreign key relationships among Employee, Project, Project Assignment, and Role. ....	76
<b>Figure 38.</b> Hierarchy of value domains .....	80
<b>Figure 39.</b> Meta model for the operational data model. ....	82
<b>Figure 40.</b> Relationship among Specified, Implemented and Operational Data Models .....	84
<b>Figure 41.</b> Multiple versions for a particular operational data model DBMS schema and supporting metadata within the domain of a particular database and DBMS. ....	86
<b>Figure 42.</b> SQL:1999 DBMS Column transformations that may be required to accommodate SQL:1992 DBMSs. ....	88
<b>Figure 43.</b> Hierarchy of value domains from data element value domain through DBMS column value domain. ....	94
<b>Figure 44.</b> View meta model data model. ....	97
<b>Figure 45.</b> Integration of semantics and data element model with other essential to enterprise-wide data standardization models. ....	99



## **Forward**

This book introduces the architecture and concept of operations for the Whitemarsh Metabase module: Data Modeler. This book coupled with the various Data Modeler user guides enable users to effectively design and evolve enterprise wide data semantics and data models.

The data modeler module “lives” within the Database and Database Objects component of the Metabase Domain diagram below. As seen through this diagram, persons through their role within an organization perform functions in the accomplishment of enterprise missions. They both employ and need information. These information needs reflect the state of certain enterprise resources such as finance, people, and products that are known to the enterprises. The states are created through business information systems and databases. Databases in turn are known to the enterprise through database objects and data models. This book is all about the architecture of the data models within and among databases and database objects.



## 1.0 Why Data Standardization Is Important

From the information technology perspective, it is commonly felt that data standardization affects only information technology. That cannot be farther from the truth. While certainly data standardization affects data element definitions and their specifications that are employed as semantic templates for business facts contained in a myriad of data files, data models, databases<sup>1</sup>, and information systems, the domain of data standardization is much pervasive. Because data standardization is so pervasive, the lack of data standardization is very expensive. Four examples serve to illustrate the “value” in achieving enterprise-wide data standardization.

First, there are generally common statistics about database environments with a typical state government, or multiple business line enterprise. These are provided in the Figure 1.

Unit	Components
1 database	100 tables
1 table	15 columns
Each agency	100 databases
Each state	40 agencies

**Figure 1.** Typical metadata statistics for databases.

The total count of columns for each database is thus 1500. The total columns for an agency is 15,000 columns, and the total for the state, for example, like Washington, New York, or Maryland would be 6,000,000. It is Michael Brackett’s assertion that, for example, the State of Washington only has about 20,000 data elements. Given that this is approximately true, which is likely because of Brackett’s long association with the State Government of Washington, then each of the asserted 20,000 data elements is reused about 300 times. For sure many are used hundreds of times and some are used only once. Clearly then, a key benefit is that there is significantly reduced effort in specification, implementation and maintenance.

In a second example of the benefits of achieving enterprise wide data standardization, an agency within the United States Department of Defense, tasked a contractor (bordering on a sentence of eternal damnation, some would suggest) to synthesize the unique set of data elements from among the complete set of columns. The result is presented in Figure 2.

---

<sup>1</sup>

In this book and in all Whitemarsh materials, database means database. That is, a well organization collection of data that may be defined through and controlled by a database management system (DBMS). When the term database is used it means database. When DBMS is used it means database management system.





<b>Data Standardization Alternative</b>	<b>Final Quantity of defined components</b>	<b>Cost</b>
Traditional (prime + modifier + classword) across all systems	19,000 (at the columns, cells, fields, and attributes level)	\$6.75 million
Accomplished by standardizing closely named columns and fields	3,000 (still at the columns, cells, fields, and attributes)	\$1.06 million
Accomplished through standardization techniques in this paper.	560 (at the data element level and then generation of automatically defined columns, fields and cells)	\$450,000

**Figure 2.** Costs and Benefits from Data Standardization Alternatives

In this particular example of determining data elements from columns, the ratio was 34 to 1. While this ratio is smaller than the first example, it was for only one database application. The third column in this table clearly shows the effect on cost incurred to define the original 19,000 columns. The effect is dramatic, simply dramatic.

A third example comes from another United States Department of Defense agency. A study was undertaken to determine the cost of Extract-Transform-Load (ETL) efforts. Each effort is characterized by a requirement, design, software implementation and maintenance. Each such ETL represents a failure in data standardization. Columns that were supposed to be the same were represented through different names, semantics, data types, levels of granularity, time-sequencing, and the like. While an enterprise-wide data element standardization approach would not solve all these problems, it would clearly affect different names, semantics, data types. The agency spends about \$175,000,000 each and every year on such ETL activities. In case that doesn't seem like much money, it represents about six hundred \$300,000 houses every year. If the data element approach resolved 50% then that would represent savings of about \$90 million per year. Extended to the U.S. DoD as a whole the savings would be about \$450 million, and to the U.S. Government as a whole, about \$1.5 Billion. Given that the U.S. Government spending represents about 10% of the total economy, then the savings to the economy as a whole is about \$15 Billion. These savings are not small.

A fourth and final example was provided by a reviewer of the data element meta model that is part of the data modeler component of the Whitemarsh metabase. He reports that an effort was mounted to bring together sets of rows from a large number of different SQL tables within a health care environment. The columns, whose actual data values were "mined" were all expected to have values of Yes or No. Figure 3 speaks for itself.



Yes No Column Frequency	Code values
1296	1 = yes.....0 = no.....
899	Y = yes.....N = no.....
740	1 = yes.....
75	Y = yes.....
17	0 = yes.....1 = no.....
2	1 = yes.....0 = no.....2= error
104	Y = yes.....N = no U = unknown
1	Y = yes.....N = no.....D = does not apply
3	0 = false.....1 = true.....
5	Y = yes.....N = no.....N/A = not applicable
90	1 = yes.....2 = no.....
6	1 = yes.....2 = no.....3 = not applicable
1	1 = yes.....2 = no.....N = none
2	1 = yes.....2 = no.....0 = do not use default
88	1 = yes.....0 = no.....99 = unknown
1	1 = yes.....0 = no/negative.....99 = unknown
1	1 = yes.....0 = no.....2 = unknown
24	Y = yes.....N = no.....NA = not applicable
17	Y/N items to be deciphered.....
11	1 = yes.....0 = no.....2 = not applicable
1	1 = yes.....0 = no.....U = undetermined
1	1 = yes.....2 = no.....3 = unknown
2	Y = yes.....N = no.....blank = no
1	0 = yes.....1 = no.....2 = not applicable
1	2 = yes.....1 = no.....
1	1 = yes.....0 = no.....U = unknown
1	Y = yes N = no A = ask
1	Y = yes N = no R = restricted
1	yes = yes no = no
1	1 = yes 0 = no 2 = ask
3394	Total Columns

**Figure 3.** Enumerated code values for a collection of healthcare tables for Yes|No columns



All of these examples and savings that result from an enterprise-wide approach to data elements are *just* for database applications. What are the savings for other types of access methods like spread-sheets, non-database file access such as COBOL and ISAM, and non-automated “systems” such as form, person-to-person communications? What about savings in documentation, training, systems analysis, and design? What about the probably uncountable calls to technical support to ask, “What does that field mean?” Those costs are probably incalculable by any reasonably defensible means.

Situated within the context of the Knowledge Worker Framework<sup>2</sup>, attention to data standardization starts on a high row and its effects spread across all columns. Figure 4 presents the Knowledge Worker Framework and the cells directly affected by data standardization are shaded.

## 1.1 Areas of Data Standardization

Data exists within various databases in a variety of formats and under a variety of names. Because there are seldom enterprise-wide standards that control these formats and names, the same business data is often represented differently: hence conflicting semantics.

The six distinct functions within the Whitemarsh metabase data modeler broadly accomplish:

- Data Semantics
- Data elements
- Specified data models
- Implemented data models
- Operational data models
- View data models

Data semantics, created within the data element module are employed in the data standardization effort to regularize the rules and meanings of the data represented by the data values associated with either a context dependent or context independent business fact. These are then applicable to business fact representations, that is, data elements, attributes of entities, columns of tables, DBMS columns of DBMS tables, and view columns of views.

Data elements are employed as semantic templates for the creation of context dependent collections of business facts. Each collection is commonly referred to as a data model. In the most general terms, four of the functions support the creation of four data models:

- Specified
- Implemented
- Operational
- View

---

<sup>2</sup>

The book, Knowledge Worker Framework (significantly different from the Zachman Framework), can be downloaded from the website, [www.wiscorp.com](http://www.wiscorp.com).



Whitemarsh Knowledge Worker Framework						
Deliver-ables	Mission	Man-Machine Interface				
		Machine		Interface	Man	
		Database Object	Business Information System	Business Event	Business Function	Organization
Scope	List of business missions	List of major business resources	List of business information Systems	List of interface events	List of major business scenarios	List of organizations
Business	Mission hier-archies	Resource life cycles	Information sequencing and hierarchies	Event sequencing and hierarchies	Business scenario sequencing and hierarchies	Organization charts, jobs and descriptions
System	Policy hiera-archies	Semantics, Data Elements, and Specified Data Model	Information system designs	Invocation protocols, input and output data, and messages	Best practices, quality measures and accomplish-ment assessments	Job roles, responsibilities, and activity schedules
Tech-nology	Policy execution enforce-ment	Implemented Data Model and Detailed Database Objects	Information systems application designs	Presentation layer information system instigators	Activity sequences to accomplish business scenarios	Procedure manuals, task lists, quality measures and assessments
Deploy-ment	Installed business policy and pro-cedures	Operational Data Model	Implemented information systems	Client & server windows and/or batch execution mechanisms	Office policies and procedures to accomplish activities	Daily schedules, shift and personnel assignments
Oper-ations	Operating business	Application view Data Models	Operating information systems	Start, stop, and messages	Detailed procedure based instructions	Daily activity executions, and assessments

**Figure 4.** Areas of the Knowledge Worker Framework Affected by Data Standardization and Data Modeling



All four of these models are critical to achieve enterprise-wide data standardization. If one or more models are missing enterprise-wide data standardization failure is assured.

A specified data model is one that is technology independent and thus not bound by any particular database management system or computing implementation, for example, a data structure modeling the needs of human resources (HR). The specified data model triple is: subject, entity and attribute.

An implemented data model is one that is technology dependent and is bound by a particular linguistic expression of a data model's semantics, but has not been implemented through a particular DBMS, nor particular computing environment. For example, a HR database expressed in the ANSI SQL:1999 language. The implemented data model triple is: schema, table, column.

An operational data model is one that is technology dependent, is bound by a particular linguistic expression through a specific DBMS and is implemented on a specific computing environment to serve the needs of one or more data processing applications. For example, an HR database under Oracle/10g on a Compaq Proliant Unix server in Cleveland, Ohio. The operational data model triple is: DBMS schema, DBMS table, and DBMS column.

Finally, the view model is the application-side data model expressed as SQL views. Thus, the application view model is technology dependent, is bound by a particular linguistic expression through a specific DBMS, and is implemented on a specific computing environment to serve the needs of one or more data processing applications.

Section 1.5 of this book identifies a summary of the problems associated with managing enterprise-wide data models. A more detailed presentation of these data element problems is contained in the paper, *An Olde Saw that Just Won't Cut*, and in other Whitemarsh materials that address data standardization. The Whitemarsh paper, *Data Architectures*, presents the characteristics of the five data architecture classes. Finally, the paper, *A Column by Any Other Name is Not a Data Element*, are all described on the Whitemarsh website, [www.wiscorp.com](http://www.wiscorp.com). Full versions of these papers are available for Whitemarsh website members.

Collectively, if the six functions (data semantics through view data models) are accomplished in roughly that order, enterprises will achieve a significant boost in data standardization. Enterprises will additionally notice that:

- Data quality will rise because there will be fewer unnecessary semantic conflicts for the same data
- Inherent project risks will decrease because fewer abstract concepts will be unknown as projects proceed
- Computing efficiency will increase because computers will not be spending wasted time extracting, transforming and then loading data into downstream databases and file structures
- Staff productivity will increase because projects will require less work for the same scope

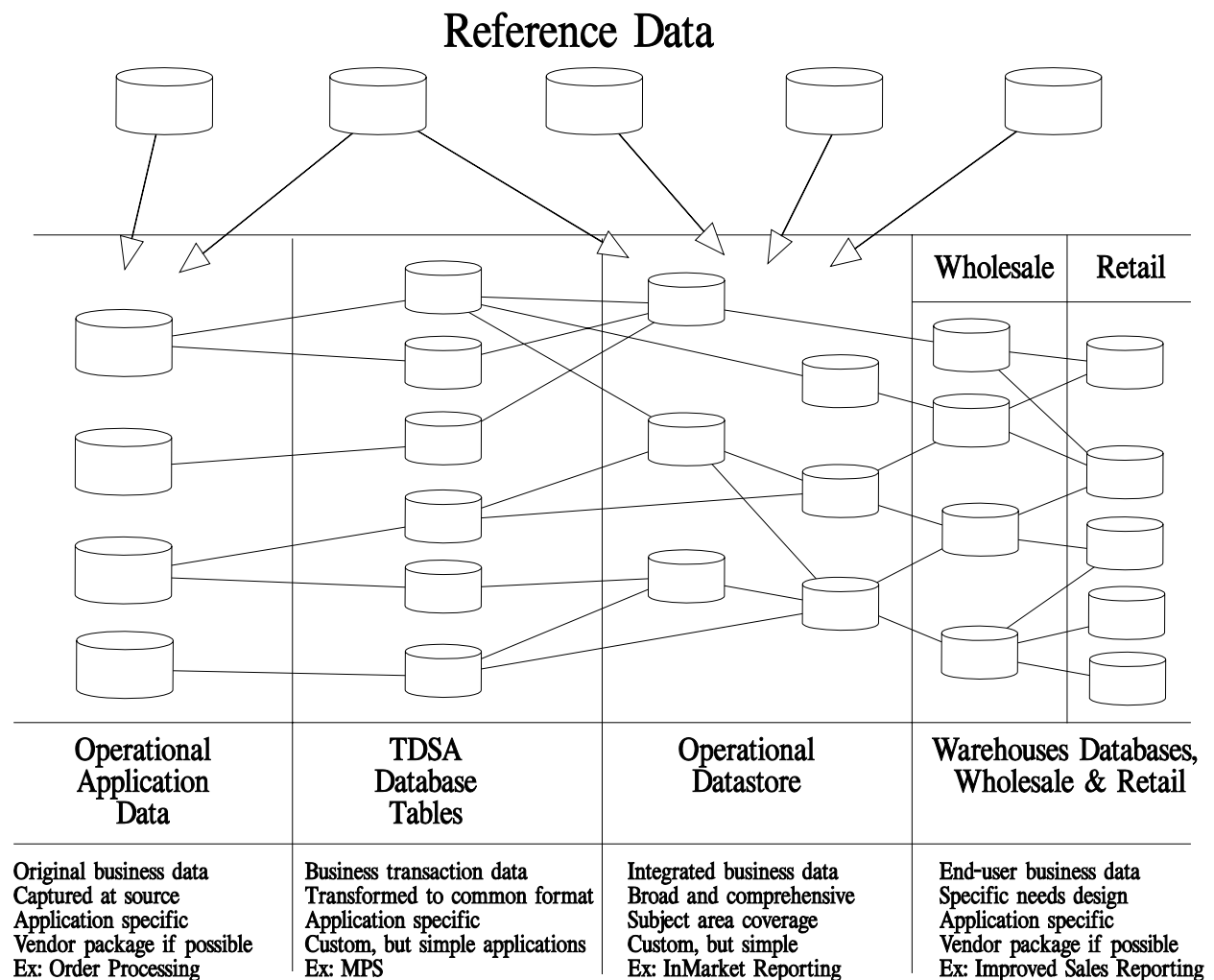
## 1.2 Data Architecture Classes



In addition to the four classes of data models there are also data architecture classes. A data architecture class is a particular style of data model that serves a particular class of database applications. The five most widely recognized data architecture classes are:

- Original data capture
- Transaction data staging area
- Subject area database
- Data warehouse
- Reference data

Figure 5 illustrates these five data architecture classes and shows the flow of data from the original data capture to data warehouses. Reference data is employed by which ever data architecture class is appropriate. These data architecture classes are fully described in other Whitemarsh materials.



**Figure 5.** Data Architecture Classes



### 1.3 ISO and ANSI Data Management Activities

The ANSI (American National Standards Institute) and ISO (International Standards Organization) data management standards efforts are roughly divided along the same lines as the major components of the Whitemarsh data modeler. The ISO SC32 (Standing Committee) Data Management and Interchange committee has a number of working groups. Working Group 2 (WG2) addresses data semantic classification hierarchies, data element model, and the specified data model, and metadata repositories on the whole. The Whitemarsh data element submodel claims conformance to the requirements of the ISO standard, 11179, created by WG2, Data Element Metadata.

WG3, the “SQL” committee addresses the implemented data model effort and all the metadata appropriate for “SQL databases” within the SQL:1999 standard’s schema information tables. Finally, all the SQL vendors such as Oracle, Sybase, IBM, CA, Informix and Microsoft address the operational data model efforts. WG4 is a committee that specified standard uses of SQL through the creation of SQL methods libraries for spatial and full text.

In the ANSI INCITS (International Committee for Information Technology Standards) technical committees, L8 (the ANSI committee that is associated with ISO’s WG2) addresses data semantics classification hierarchies and data elements, T2 addresses conceptual schemas as is appropriate for specified data models, and H2 (the ANSI committee associated with ISO’s WG3 and WG4) addresses implemented data models and the metadata for implemented data models through the schema information tables. Finally, the SQL vendors such as Oracle, Sybase, IBM, CA, Informix and Microsoft address the operational data model efforts.

### 1.4 Bulk Data Loading or Importing

Because most—if not all—organizations have already expended significant funds on data models or have large collections of data file definitions, the Whitemarsh data modeler supports the importing of:

- Data element metadata
- Data models

Whenever data is imported, the required foreign key linkages to the appropriate “upstream” semantics (DBMS column must be related to table column (which in turn is related to data element)) cannot, of course, already be present. To overcome this lack of mandatory foreign key values, the Whitemarsh bulk data loading or import functions will default the foreign key values to a special value that means “unknown semantics.”

This special “unknown semantics” value is created through a special administrative function when the data modeler is first installed. Then, as these imported data are integrated into the overall set of enterprise wide semantics the “unknown semantics” can be updated to the appropriate “known semantics” by choosing the appropriate foreign key value.

For example, suppose there is a data file to be loaded that contains 50 new DBMS tables and 15-40 DBMS columns for each of the new DBMS tables. Given that one of the new DBMS



columns for a new DBMS table, *Salesman*, is *FN*, then, if there isn't an already loaded default "unknown semantics" foreign key value default, then the DBMS column, *FN*, would not be successfully loaded because of a referential integrity error between the DBMS column and the table column. To resolve this "problem" the implemented data model is preloaded an "unknown" schema that has a table called Unknown Table with a column called Unknown Column. But because of the "unknown semantics" default foreign key value, *FN*, is automatically related to the "unknown column" within the "unknown table," that represents "unknown semantics." Once properly analyzed *FN* can then be related to the appropriate table column, Employee First Name, within an appropriate table, Employee. The table column should already be related to the data element, *First Name* within the Business Domain, *Person*. This then means that while *FN* is a DBMS column name within the DBMS table *Salesman*, its real semantics are that it is really the first name of a person.

## 1.5 Business Fact Standardization Problems and Solution Approach

The traditional approach to business fact standardization, that is, the three-part paradigm of *prime word*, *modifier[s]*, and *class word* was introduced in the middle 1970s to satisfy the needs of naming fields within the contained data structures of COBOL programs. While quite suitable for COBOL, once this three-part paradigm was brought into the database world, five distinct problems immediately arose. These were:

- That data elements were mistakenly seen as synonyms for table columns, screen cells, entity attributes, or report fields.
- That data elements don't have names
- That prime word, modifier[s] and class word *are* part of the data element's name
- That modifiers are from a single homogeneous set
- That there is only a choice of one class word

These problems are presented and analyzed at length in data standardization materials from the Whitemarsh website ([www.wiscorp.com](http://www.wiscorp.com)). Collectively, these errors have cost hundreds of millions of dollars in wasted data standardization efforts. The Whitemarsh approach to a solution is a business fact standardization strategy that is based on a meta attribute classification scheme that both supports a business domain, common business name, modifier classes and subclasses, multiple class word classes and subclasses, and is also preserved within the metadata repository so the relevant semantics are immediately retrievable.

Data elements, are thus context independent business fact semantic templates. Context dependent business facts are cells in a screen, fields in a file, columns in a table, or variables in a program. While most of the meta attributes necessary to describe the context independent and





Meta Attributes for Context Independent and Dependent Business Facts		
Associated Meta Attribute or Semantic Part	Context Independent Data Element	Context Dependent: Table Column, Screen Cell, Entity Attribute, Report Field, etc
Business Domain	Yes	No (but yes by inheritance)
Common Business Name	Yes	Yes
Entity or Table	No	Yes
Semantic Modifiers	No	Yes
Data Use Modifiers	Yes	Yes
Generated Policy Basis Description	Yes	Yes
Computer Data Type	No	Yes
Data Structure	Yes	Yes
Required Uniqueness	No	Yes
Relationship Function	No	Yes

**Figure 6.** Meta Attributes for Context Independent and Dependent Business Facts

dependent business facts are the same, there are some differences. Figure 6 identifies these differences.

## 1.6 Data Modeling Problems and Solution Approach

Many data modeling software tools are founded on graphics. That is, entity-relationship diagrams of one style or the other. While pictures are nice, they do not compile and they are seldom sufficient or complete. There are so many different types of metadata necessary for a complete data model that a “picture” seldom represents more than a small percentage of what is necessary. Finally, if it was all graphically represented then these graphics would just be a semantic blizzard.

In addition to having a pretty picture of a single data model, there are actually **three** different kinds of models that must remain in synchronization. These are: *specified*, *implemented*, and *operational*. The specified model is the province of the functional data modeler and/or data administration (DA). The implemented model is the province of the database administrator (DBA). The operational model is commonly the responsibility of a



DBMS administrator which may also be someone in the Database Administration group. A comprehensive solution to data modeling must therefore provide:

- Full control over context independent data elements that are then employed, for example, as attributes of entities or columns of tables.
- A clear demarcation between data models that serve different purposes and stages in the database design and implementation process with the ability to evolve these different models and to maintain the mapping between the models, possibly through SQL view-like language commands.
- Fully definable meta-attributes for all components of the data model (e.g., entities, data elements, columns, primary and foreign keys) including the ability to have meta-attribute value set definitions that can in turn be used for automatic definitions.
- The ability to express sophisticated integrity constraints for individual columns, groups of columns, inter-table constraints, rows, sets of rows within and across tables, etc.
- Fully definable reporting capability against the data modeling metadata database.
- The ability to have data model meta components fully integrated with other IT components such as views, reports, screens, systems, "programs," user guides, and other forms of documentation.
- Graphical support with the ability to re-size the icons that represent "entities," the ability to have multiple levels to any diagram to then expose the underlying substructures within represented "entities."

The approach provided by Whitemarsh addresses all these needs except for the last, graphical support. Graphical support is already provided within by diagraming packages (e.g., Dezine by Datanamics ([www.datanamic.com](http://www.datanamic.com))) that can create, read and write SQL DDL streams and create diagrams. The Whitemarsh data modeler produces SQL DDL streams as ASCII output files or reports. To have diagrams that support more semantics than ANSI SQL can implement is a waste of analysis and design efforts because it gives a false impression of what can be accomplished through an ANSI SQL DBMS. The Whitemarsh data modeler, in contrast, has one and only one target: The ANSI SQL DBMS. If it can be created in the Whitemarsh Data Modeler then it can be implemented in an ANSI SQL DBMS. While this strategy may be an affront to data model diagram purists, it must then be stated that the Whitemarsh Data Modeler was not created for them. Rather, it was created for the data administrators and database administrators who actually have to implement, operate and maintain what they "devine."

The terms, logical and physical are purposefully not used by Whitemarsh because they are ambiguous. For example, the physical model to a data modeler is SQL schema data definition



language (DDL). To a database designer (i.e., DBA) the schema DDL is the logical model, especially if it's in third normal form. The DBA then transforms the schema DDL to one or more different schema DDL implemented designs to conform to different DBMSs, operating systems, database sizes, query characteristics and volumes. Some DBAs consider each one of these physical schemas while other DBAs consider them still to be logical schemas. Some also hold that the logical schema is the input side of a transformation and the physical schema is the output side of the transformation.

Whitemarsh therefore uses the terms *specified*, *implemented*, and *operational* rather than *conceptual*, *logical* and *physical* as there may be many transformations that occur for different reasons within the specified database design activities and within the implemented database design activities. As to where the metadata are stored, the answer is quite simple: the metadata repository. The key characteristic is that they both be stored in the same metadata repository. The specified model MUST always be in at least third normal form as that is the most precise.

The Whitemarsh data modeler allows [human] data modelers and database designers to create models by entering data through a panels interface. Reports from data modeler can be exported to SQL and read into an entity relationship graphics package that can import SQL DDL. Whitemarsh already uses such a package.

## 1.7 View Data Model Problems and Solution Approach

The third area addressed by the Whitemarsh data modeler is the interface between applications and operational data models. With the advent of the ANSI SQL standard for Call Level Interface (CLI), which Microsoft has implemented as its ODBC, application programs are able to interact with multiple DBMSs. This means that shrink-wrapped software can be purchased that employs the ANSI CLI to access data without having to have the application software specially created for the particular DBMS.

Because of these two layers of independence between application programs and databases through vendor specific DBMSs, enterprises cannot safely depend on the DBMS's internal metadata administration facilities to manage all SQL views. These views, if they are to be centrally managed must be defined within the metabase and then issued as data definition language command files to the DBMSs that are in turn accessed by the application programs for accessing data.

An additional feature of ANSI SQL views is that significant procedure logic can be installed into the view. This additionally allows more and more of the application independent data integrity rules to be defined within the purview of the DBMS and to be managed by the enterprises's data and database administration groups. This decreases risk, increases quality, and generally increases the overall effectiveness of critical staff.

The Whitemarsh metabase data modeler module enables the central definition and administration of ANSI SQL views. These views can be printed as command files that can be sent to DBMSs for compiling and storage within the DBMS's schema information tables.



## 1.8 Integration with Database Objects

With the standardization of SQL:1999, two dimensional data structures are an anachronism. SQL:1999 tables can contain many layers of substructures and each defined data unit can be supported by sophisticated processes. This enables the creation of database object classes with the scope the ANSI SQL language. The Whitemarsh metabase module, data modeler directly supports these SQL database objects.

## 1.9 Whitemarsh Data Modeler Graphics

The Whitemarsh data modeler creates graphics in two ways: dynamically and indirectly. First, it dynamically generate graphically oriented data model trees for the specified, implemented, and operational data models. In these graphic trees, the targeted root entity upon which the dynamically created tree is based, is colored black, descendent entities are colored blue, ancestor entities are red, and subtyped entities are colored cyan. As each entity is highlighted in the data model tree, all the attributes and keys are presented in “surrounding” list windows.

The Whitemarsh data modeler generates an output file that can be fed to an optionally purchased product that creates both entity-relationship diagrams for the specified, implemented, and operational data models, and also full SQL streams for the operational data model. This output file may also be fed to the Clarion development environment in the support of completely operational client-side applications.

Notwithstanding, these “pretty pictures” serve only as graphical representation reports of some aspects of the underlying metadata that must be stored and interrelated in a comprehensive enterprise-wide metabase of semantics.

From Whitemarsh’s point of view, graphics are like a few deck chairs on the Titanic. Having all the deck chairs but missing the completely water tight hull is like trying to say that these lashed-down deck chairs will keep the Titanic afloat after it hit iceberg. Clearly it sank, deck chairs and all. So too has sank many enterprise-wide data standardization efforts that were based almost exclusively on pretty pictures.

## 1.10 Whitemarsh Data Modeler, a Difference in Kind

The Whitemarsh data modeler is different from many data modeling packages because it approaches data modeling from the vantage point of the entire enterprise rather than from the viewpoint of a single database. Furthermore, a key goal of the Whitemarsh data modeler is to generate ANSI SQL DDL command files for ANSI SQL-based databases. Enterprise database is real only if it exists as enterprise-wide operational databases across all the data architecture classes. Otherwise, enterprise database is just a set of pretty pictures representing the nirvana that will never be.

Whitemarsh’s design presumes that there will be many different databases that employ the same set of semantics over and over throughout different data architecture classes. This fundamental design requirement means that the Whitemarsh metabase must first and foremost



support the definition of the semantics of the enterprise's business facts, that is, data elements, and then the deployment of those data element semantic templates within the specified, implemented, and operational data models across the five distinct classes of data architecture.

Figure 2, above, highlights the financial rationale that makes this approach absolutely critical to data standardization. In a reflective study of a very large data standardization effort, it was shown that even though the organization had in excess of 19,000 application data elements, the actual quantity of completely context independent business facts was far less than 1000. If the enterprise had followed the approach presented in this book and in the other Whitemarsh materials then their expenditure of \$6.75 million for application column development could have been less than \$500 thousand.

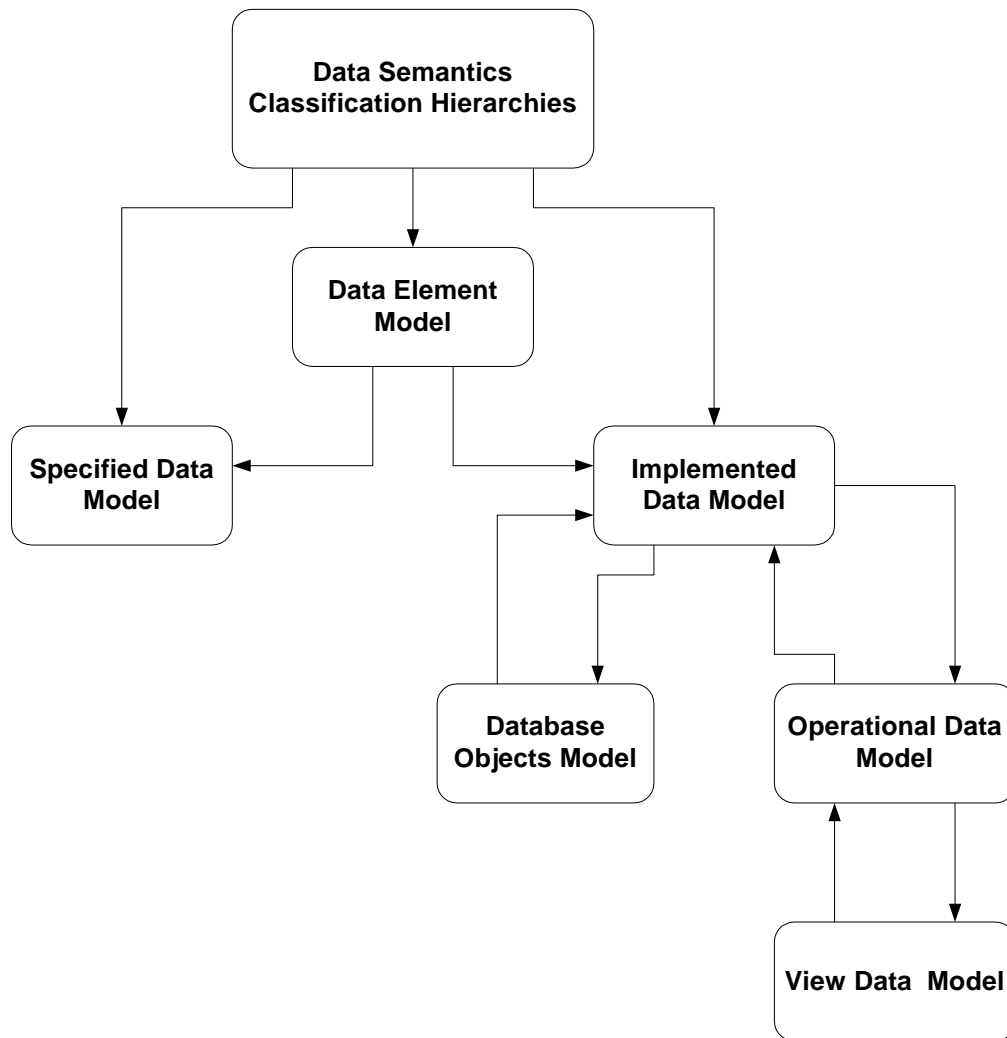
The Whitemarsh metabase module, data modeler, produces ANSI standard SQL data definition language. Many development environments such as Oracle, Powersoft, and Clarion for Windows all support the direct development of entire application systems from ANSI standard SQL. In the case of Clarion for Windows, once the SQL compiled by the DBMS, Clarion can synchronize to that database schematic and then through its own facilities, create a complete client side application system through code generation. The metabase also supports the creation of Clarion's DDL, and once the Clarion DDL is compiled, a complete client side application system through code generation



## 2.0 Whitemarsh Data Modeler Architecture & Concept of Operations

Figure 7 presents the overall design of the data modeler. The six distinct functions of data modeler are:

- Data Semantics
- Data elements
- Specified data model
- Implemented data model
- Operational data model
- View data models



**Figure 7.** Interrelationship among the six models that comprise data modeling.



Figure 7 also shows that the database objects model interfaces with the implemented data model. It interfaces there because the data structures within database objects are tables, and when exported are linguistically expressed in the ANSI standard SQL.

Because each model corresponds to a class of work that is generally accomplished by a single group of data designers or data modelers, each is presented in its own section. The diagram for each section contains shaded and unshaded meta entities. The data represented by shaded meta entities is created and updated either in other data modeler submodels or in other metabase meta entity models.



### 3.0 Data Semantics

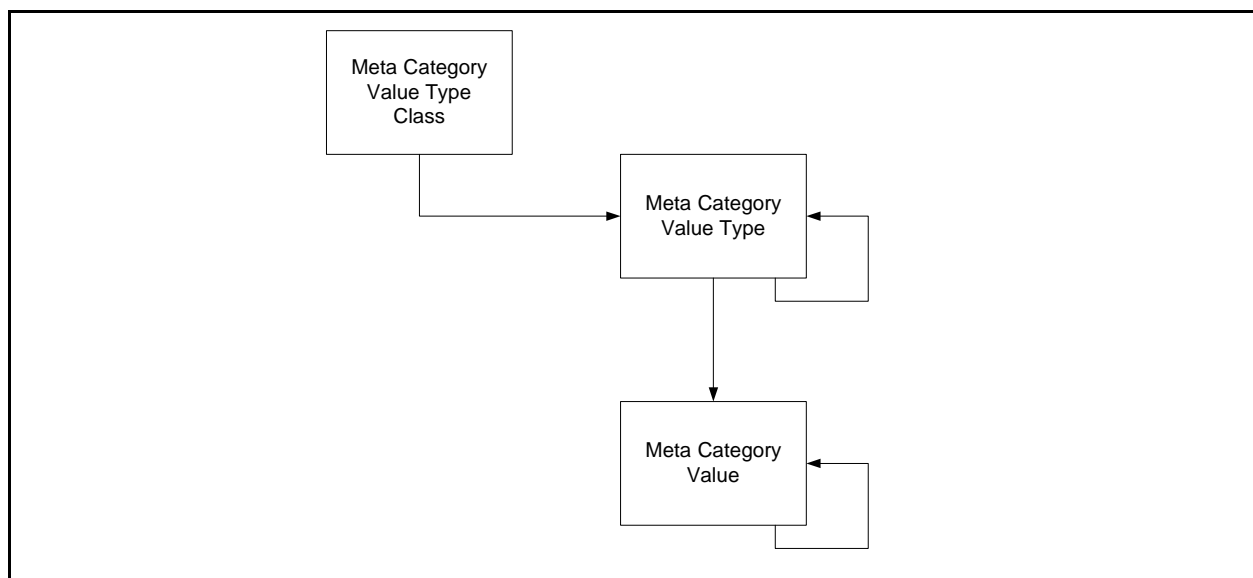
Business facts, whether context independent or dependent represent a set of semantics established by the enterprise. Semantics are the rules that govern meaning and usage. The semantics are represented through a series of classification hierarchies. Each classification hierarchy represents one complete class of semantics. Each class of semantics relates to a single meta category value type, which itself is organized into a hierarchy. While there is no limit to either the quantity, breadth, or depth of a meta category classification hierarchy, the most common ones are:

- Semantic modifiers that provide special contexts for the common business fact meaning. These are the prefix meta category values
- Data use modifiers that point to the data types, roles performed, and value units. These are the suffix meta category values.

The data model for Meta Category Value Type Classes, Meta Category Value Types, and Meta Category Values is depicted in Figure 8. The Meta Category Value Type Class broadly divides the meta category value type and meta category value hierarchies into their class types, prefix and suffix. This diagram shows only three entities. Two are recursive. That means that each can contain hierarchically related rows of data.

### 3.1 Semantic Modifier Hierarchies

Semantic modifiers represent semantic restrictions on both the meaning and the implied value set



**Figure 8.** Meta Category Value Type and Meta Category Value Hierarchies





of a business fact. Semantic modifiers always precede the common business name. The general meaning of the value set is provided by the common business name of the business fact. For example, the common business name, Sales, does not in and of itself contain contextual restrictions on which Sales are implied. Semantic modifiers provide the semantic contexts. For example, it may be <North American> Sales. Or, it might be <Estimated> <annual> <North American> Sales. Each semantic modifier conveys a restricted meaning to the value set represented by the business fact, Sales. Each angle-bracketed term is an instance of some meta category value type. A common set of semantic modifier types include:

- Temporal
- Accuracy
- Geographic
- Organizational

Each type instance may have subtypes and once the type hierarchy is complete, the full meta category value set is provided. There can be at most one semantic modifier from each semantic modifier type hierarchy. For example, from above, there can be at most one temporal modifier, one accuracy modifier, one geographic modifier, and one organizational modifier. There can be fewer modifiers than the quantity of first level types, but not more. The data modeler automatically creates names for data elements, attributes of entities, columns of tables, and DBMS columns of DBMS tables. User-names may, of course, be additionally be created by the user. Notwithstanding any such changes, the set of semantics inherited through the meta category value types and hierarchies are maintained.

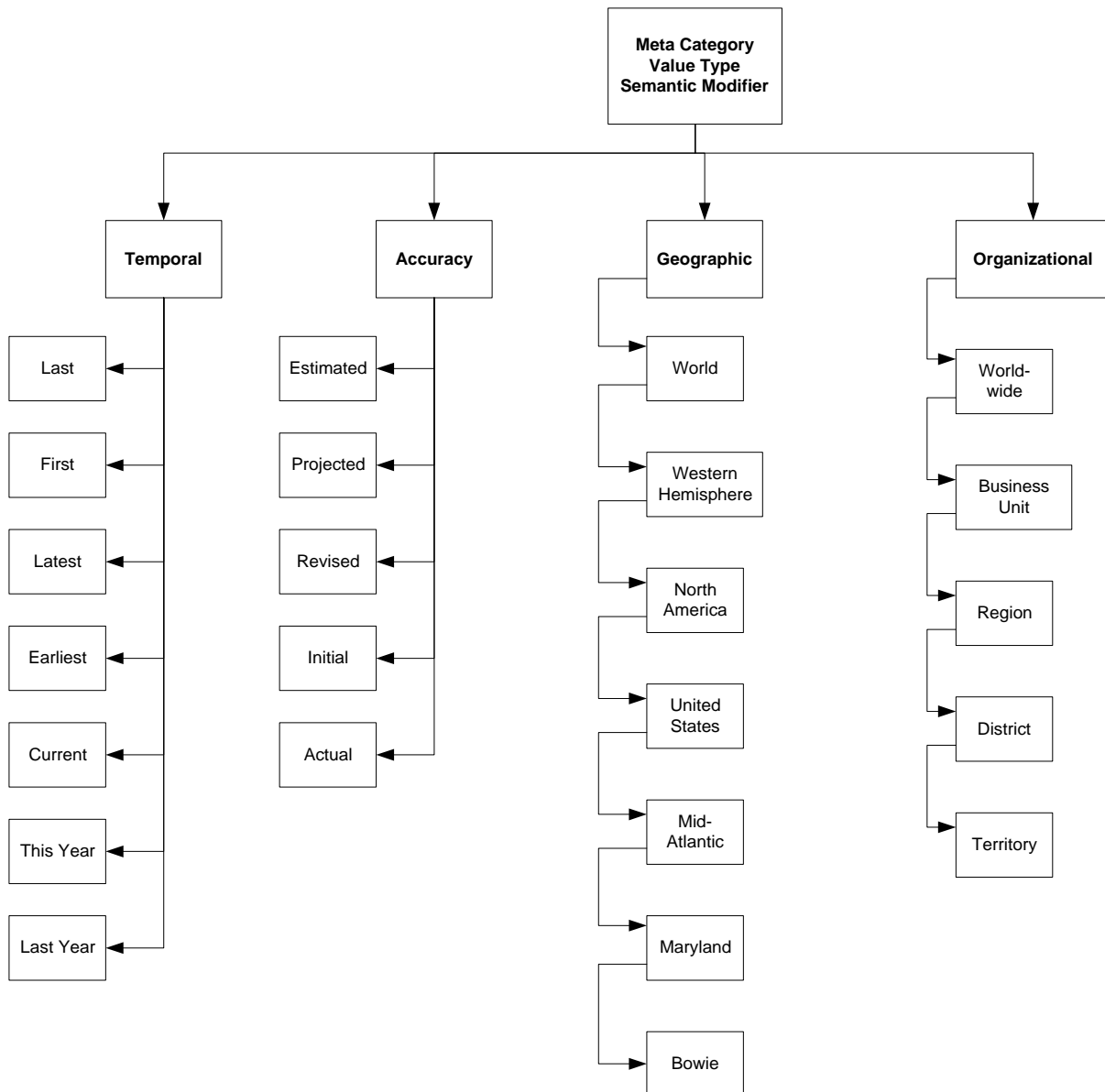
The primary purpose for breaking the meta category value classifications into two distinct hierarchies, that is, one for the meta category value types and another for the meta category values is to ensure that only one value from each meta category value hierarchy becomes a part of any context independent or dependent business fact. In Figure 9, which contains a set of semantic modifiers, the first major row is divided into two rows represents the meta category value type hierarchy. The second major row is divided into seven subordinate rows represents the lower level hierarchy. In the case of the Temporal and Accuracy semantic modifier columns, all the entries are siblings. In the case of the Geographic and Organizational columns the entries are hierarchically related.

Each meta category value and meta category value type contain a number of meta attributes beyond just the value (e.g., North American). Figure 10 presents an instance diagram of the entries of the data contained in the Meta Category Value Type and Meta Category Value Hierarchy table for Semantic Modifiers.

In operational databases, semantic modifiers are represented in two different ways depending most generally on the type of data architecture class, and most specifically on the type of data structure in which semantics are implemented. In most original data capture, transaction-data staging area, and subject area data architecture classes, semantic modifiers and even some data use modifiers are represented in columns as column name parts so that the columns can be readily recognized. In data architecture classes, data warehouse and reference data, semantic modifiers often become actual values through which data rows are selected, analyzed, summarized, and reported. In the first way, the data structure of the tables and columns is highly



specialized as the semantics are bound into the column's name. In the second way, the data structure of the tables is very generalized as the semantics become actual data values. Figure 11



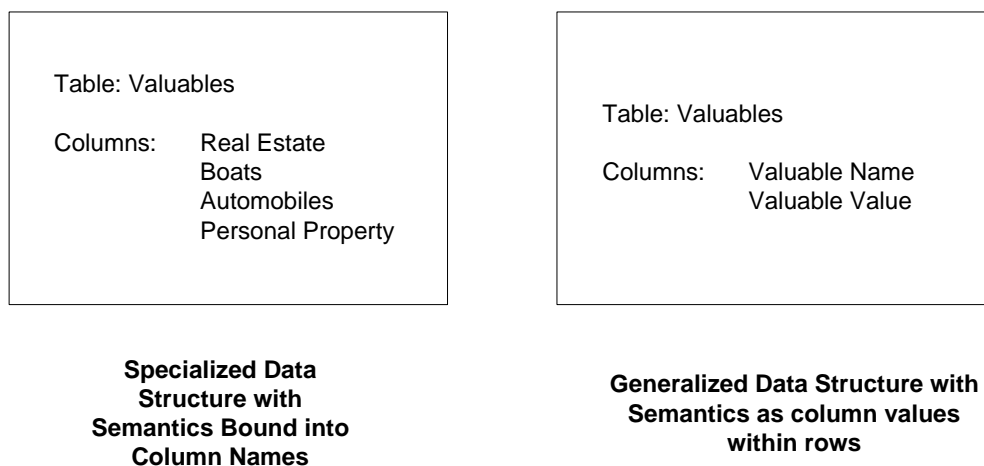
**Figure 9.** Meta Category Value Types and Meta Category Value Hierarchies

illustrates these two approaches. Because the Whitemarsh metabase's module, data modeler, approaches data standardization from a database perspective, all data semantics are first and foremost data values that can then be used either way. That is, as column names for the specialized data structures, or as column values in rows for generalized data structures.



Meta Category Value Type and Meta Category Value Hierarchies				
Meta Category Value Type Hierarchy	Examples of Semantic Modifiers			
	Temporal	Accuracy	Geographic	Organizational
Examples of Meta Category Value Hierarchies for the Meta Category Value Types	Last	Estimated	World	World-wide
	First	Projected	Hemisphere	Business unit
	Latest	Revised	North America	Region
	Earliest	Initial	United States	District
	Current	Actual	Mid-Atlantic	Territory
	This year		Maryland	
	Last year		Bowie	

**Figure 10.** Meta Category Value Types and Meta Category Value Hierarchies



**Figure 11.** Two Methods of Binding Semantics into Tables and Columns of Operational Databases



### 3.2 Data Use Modifier Hierarchies

Data elements have data use modifiers that point to the data types, roles performed, and value units that serve to describe the use of the data element. Data user modifiers are appended to the end of the common business name. These are properly quite separate from the data element's name as the common business name might misrepresent a data element's allowed use. For example, Social Security Number is not really a number in the mathematical sense of the word. What is the meaning of the average of a set of 10 Social Security Numbers, or their sum, difference, etc.

The same problem exists with Telephone Number. It's really a code consisting of a country code, an area code, an exchange and finally a four digit number. And, to dial a phone number you might enter the numeric string, 1-800-USA-RAIL. That's a number?

The most common three data use modifiers are:

- Data type
- Role
- Unit

Data types represent the fundamental business-use nature of the data values themselves. These are not however computer data types such as CHAR, VARCHAR, INTEGER, and LONG. Rather, they represent the business use of the data value. For example, does the number represent a weight, a code, dimension, volume, money, decimal, or an integer.

The second common type of data use modifier identifies the business role the data element value serves. For example, in the case of a data element called, *Skill Level*, the business data type of a number might be decimal, and the role it plays is that of a *multiplier*. The role represents the "job" the data element performs within which context it is contained. An example of the types of data use modifiers is provided in the Figure 12. Figure 13 presents that same information as an instance diagram.

There can be at most one data use modifier from each data use modifier type hierarchy. For example, from above, there can be at most one data type (e.g., decimal, money, integer), one role (e.g., identifier, factor, flag, or indicator), and one unit (e.g., inches, time, and US Dollars). There can be fewer data use modifiers than the quantity of first level types, but not more. As stated above, the data modeler automatically creates names for data elements, attributes of entities, columns of tables, and DBMS columns of DBMS tables. User-names may, of course, be additionally be created by the user. Notwithstanding any such changes, the set of semantics inherited through the meta category value types and hierarchies are maintained.

### 3.3 Meta Category Value Types and Meta Category Value Hierarchies Summary

Meta category value types and meta category values, along with other meta attributes such as definition fragments and a defined set of abbreviations greatly enhance and facilitate the construction process of the semantics of data element and the semantics for technology



Meta Category Value Type and Meta Category Value Hierarchies			
Meta Category Value Type Hierarchy	Data User Modifiers		
	Data Type	Role	Unit
Examples of Meta Category Value Hierarchies for the Meta Category Value Types	Date or date component	Identifier component	Day
	Code	Factor	Case
	Text	Flag	Aisle
	Weight	Indicator	Pallet
	Dimension	Identifier component	Transaction
	Money	Rank	Percent
	Integer	Business fact	Inches

**Figure 12.** Meta Category Value Type and Meta Category Values

dependent contexts such as fields in files, cells in screens, columns in tables, and variables in programs. A most commonly produced meta attribute is name. That is, data element name, column name, field name, cell name, or variable name. To assist in repeatable name construction, the Whitemarsh data modeler requires that the meta category value types and meta category values be sequenced in an enterprise-defined order. This way, when a name is constructed the name parts are strung together, one with another in a repeatable order.

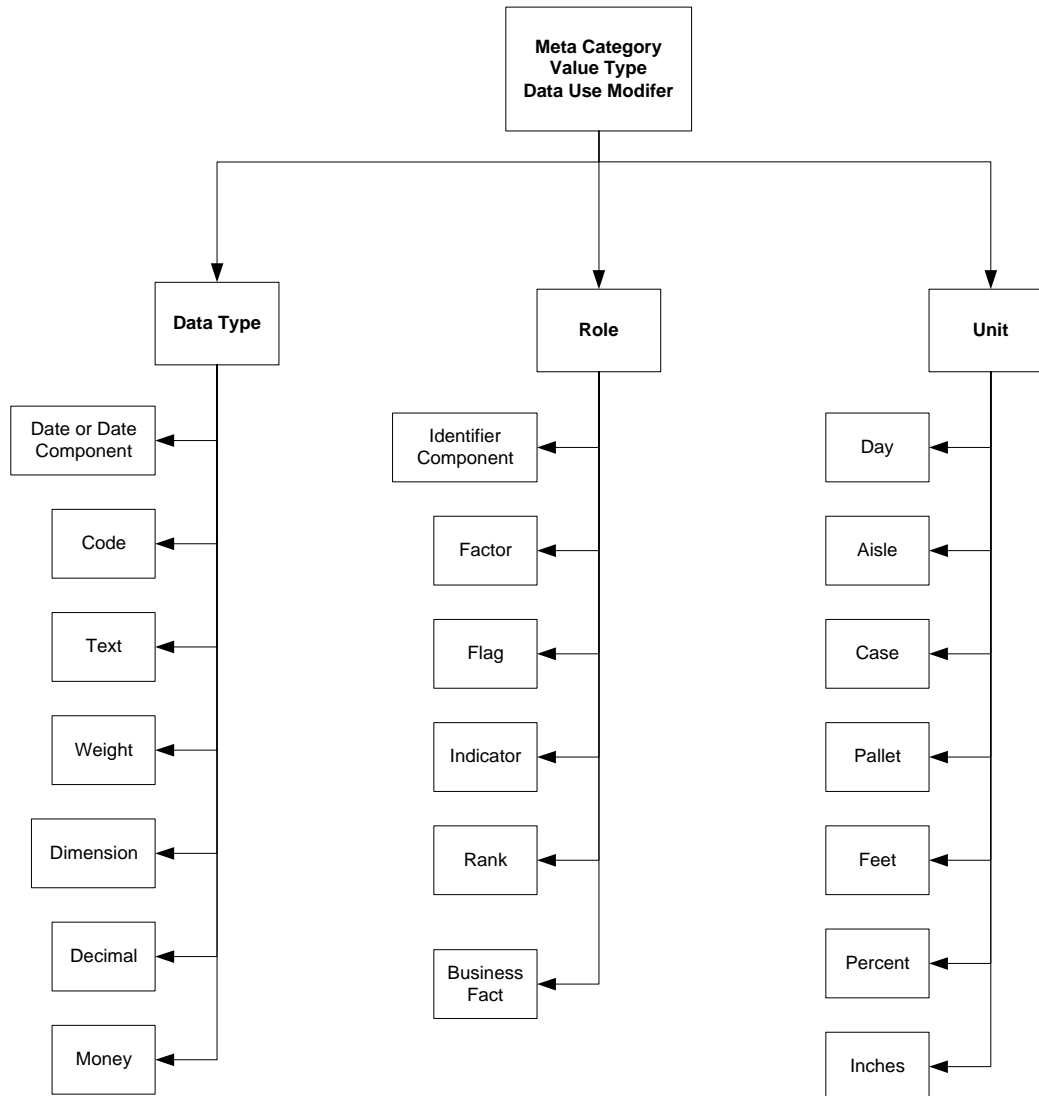
For example, the definition of North America would be the Continental United States and Canada. The acceptable alternatives might be: N\_America<sup>3</sup>, and N\_A. An easy extension to this architecture is the addition of the values, definitions and abbreviations in different languages.

*Note well, however, the name is not the semantics. Rather name is just one of the meta attributes associated with the data element, field, cell, column, or variable.*

The definitions and value alternatives are especially valuable in avoiding the task of creating business fact definitions altogether, which should be no more nor any less than the stylistic

<sup>3</sup> Note: computer languages usually “frown” on the use of spaces and periods within names. So, when ever they are encountered they are replaced with the underscore (\_).





**Figure 13.** Meta Category Value Type and Meta Category Value Hierarchies

concatenation of the definitions fragments of its semantic parts. The Whitemarsh data modeler provides that facility. The definition fragments are brought together in the same sequence as are the name parts.

A standard report from the Whitemarsh metabase data modeler prints these meta category value types and meta category value hierarchies so they can be reviewed and analyzed against the actual data semantics desired by the enterprise. Since these meta category value types and hierarchies form the essential basis for the most significant of the enterprise's data semantics, great care should be expended in their determination, review and finalization.



## 4.0 Data Element Model

The fundamental premise of the data element model is that a data element is a context independent semantic template for contextually deployed business facts. Because data elements are context independent they can be used as the semantic models for various contained business facts such as fields in a file, columns in a table, cells on a report, and variables in a program. While it may be tempting to think of data elements as fields, columns, cells, or variables, they are not. Rather, data elements are semantic templates. The semantics of data elements are represented through:

- The common name that represents the essential meaning of the business fact
- The data use modifiers that point to the data types, roles performed, and value units.
- Their allowed value sets

When fully specified, the complete semantics of a data element are represented through its associated set of meta attributes. The meta model for data elements is provided in Figure 14.

Figure 14 shows that the meta category value type and meta category values are shaded entities. That is because they should be valued prior to the entering meta data associated with data element domains or data elements. Once these data are entered, it is used to create the metabase intersection records for data element domains and data elements. The diagram is broadly divided into the following sections:

- Compound data elements
- Concepts
- Conceptual value domain
- Data element classifications
- Data element concepts
- Data elements
- Derived data elements
- Value Domains

## 4.1 Concept

Concepts represent the sets of ideas, abstractions, or things in the real world that are identified with explicit boundaries and meaning and whose properties and behavior follow the same rules. Concepts are used as a basis for specifying the concepts of data elements.







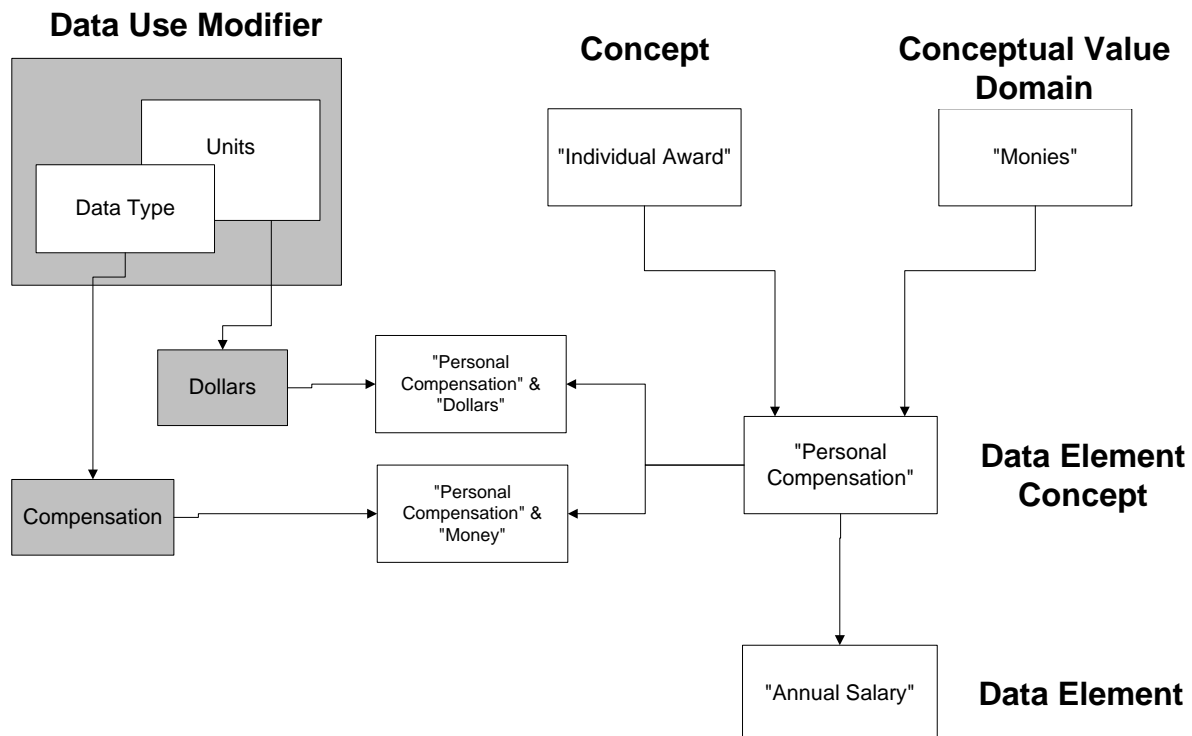
### 4.3 Data Element Concepts

Data element concepts maintain the information on the concepts and conceptual value domains upon which data elements are developed. The components of these meta entities concentrate on semantics. The concepts are independent of any internal or external physical representation. From Figure 14 it can be seen that the data element concept represents the association of a given concept with a conceptual value domain.

Data element concepts also allows for a full bill of materials structure supporting both network and hierarchical data element concepts in support of data element specifications.

For example, in Figure 15 there is a data element, Annual Salary. It clearly represents some form of personal compensation that is received by an employee within the context of the enterprise. The data element concept, Personal Compensation, is the association of the concept, individual award with the value domain, monies.

The allocation of the two meta category values, dollars and money are then appropriately added to personal compensation. While it is true that personal compensation could have been in the form of land, days off with pay, free travel, and the like, the conceptual value domain, monies constraints the potential set of allowed values. From the point of view of a data element concept, there is first Personal Compensation and then the data element Annual Salary. Each has its associated meta category value type and values as appropriate. Figure 15 presents the data element concept employed as the contextual representation for the data element, Annual Salary.



**Figure 15.** Data element concept, personal compensation.



## 4.4 Value Domains

The four meta entities in Figure 14 for value domains are:

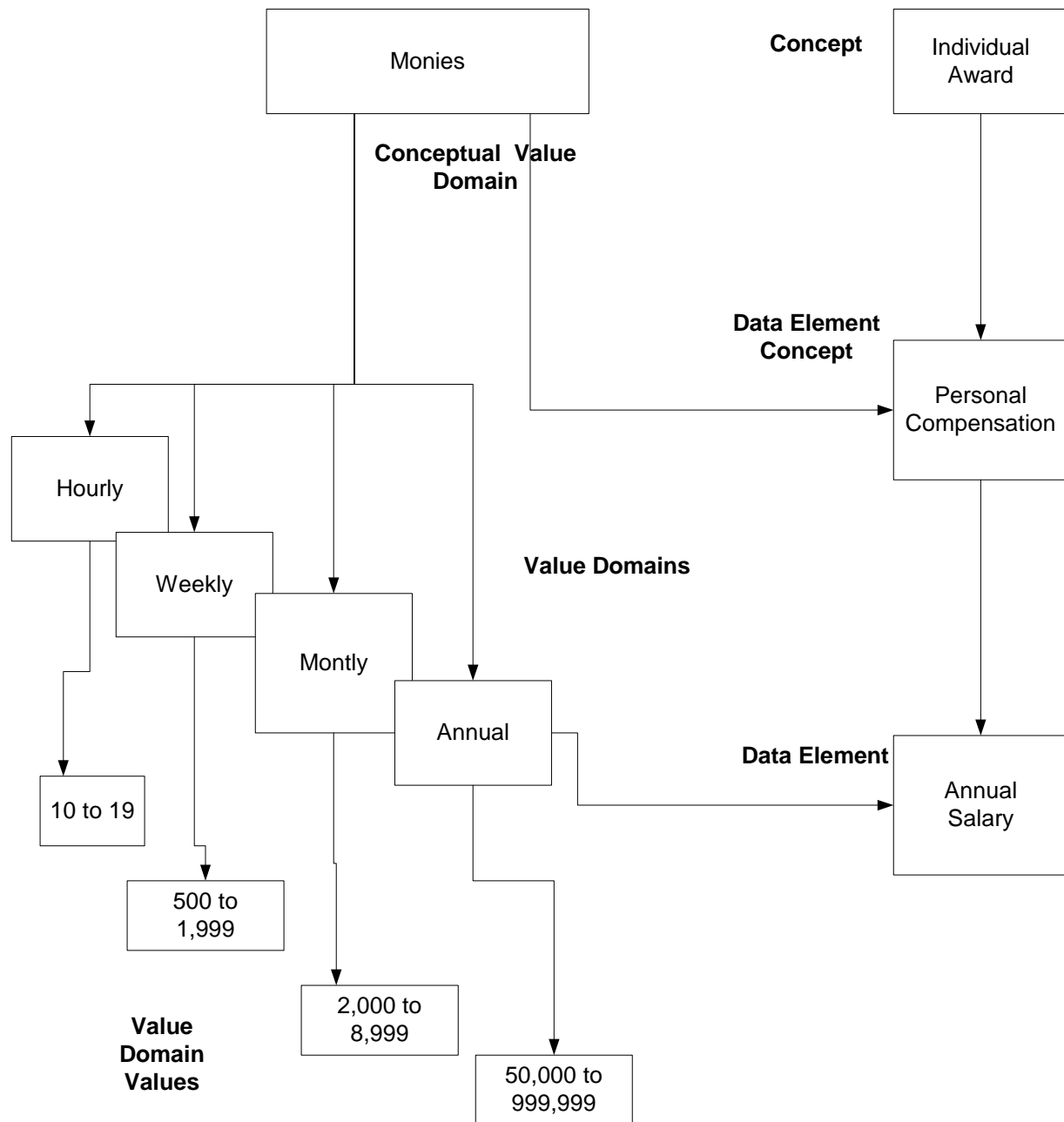
- Value domains
- Value domain values
- Value domain structures
- Value domain structure types

The purpose of these meta entities is to provide specific descriptions about the allowed values associated with one or more value domains. Value domains that are assignable to a data element must be in the same conceptual domain family as is the data element's data element concept. Value domains assigned to an attribute of an entity within the specified data model, or column within a table of the implemented data model or a DBMS column within a DBMS table of the operational data model must all be hierarchically related

Figure 16 illustrates the salary value domain which is related to the data element, annual salary. In this value domain there are hourly, weekly, monthly, and annual salary value domains. The annual value domain then has an allowed set of values. In this example, the allowed set of values are ranges up to \$999,999, appropriately able to be set for each different value domain.

If one of these value domain value restrictions is applied to a data element then it can be assigned. In this case, the annual range is assigned to the data element, annual salary. It then binds the value domain value ranges of all attributes or columns (and by inference to DBMS columns) associated with the data element. If it is not appropriate to assign a value domain value to a data element then the actual implementation strategies for these value restrictions becomes part of the specified, implemented, and operational data models. As the specification proceeds towards implementation, the sophistication of the implementing DBMS software comes into play. The least sophisticated DBMS implementation technique would require all the process and validation rules to be part of each application program that accomplishes data entry and maintenance. The most sophisticated DBMS has these rules implemented as assertions and/or triggers that determine the types of allowed values, and then validates the specific allowed value either to an enumerated list or to an allowed range for such values.





**Figure 16.** Value domains, associated values and assignment to data element.



## 4.5 Data Elements

A data element is a semantic template for use in constructing context dependent columns in tables, fields in files, cells in screens, and variables in programs. Figure 17 shows an instance diagram that represents the intersection records between a set of meta category values and a data element. From Figure 14 it should be quite clear that a data element is associative type meta entity between data element concept and value domain that is set within the context of a particular business domain.

The meta category value types are contained within the “grey” box at the top-left of the figure. The particular meta category values that are “pointed” to by the meta category value types are themselves in “grey” boxes. The meta category values, dollars and compensation, are associated with the data element domain, Compensation. Compensation is in turn, the business value domain for the data element salary. Because of inheritance, the semantics associated with the data element domain are assumed by the associated data elements. Thus, Salary is a business fact in the form of compensation of money in dollars.

All data elements exist within a specific business domain. In the case of compensation, human resources is appropriate. The reason for the business domain is that many data elements of the same name mean different concepts. For example, the data element, Tract, can mean 1) and area of land, 2) a space of time, 3) a system of body parts, or 4) a bundle of nerves. Another example, Trade, can mean 1) a course of conduct, 2) a path traversed, 3) a business one practices, and 4) to give in exchange for another commodity. Without business domains, there would be no way to differentiate the varied word uses. Either there can be no data elements with different meanings across all the business domains, or there must be a way to separate out these semantic homonyms.

Also associated with the data element is its common business name, which represents the name most commonly used name to convey the nature and purpose of the represented context independent business fact. For example, First Name, Telephone Number, Gender, Social Security Number, Invoice Date, Shipping Date, and Zip Code. Even if the data elements values are a coded set, as would the case for Gender (1 for male and 2 for female), the common name is just that: the common name. A particular data element, salary, which has as its common business name, Salary, is connected to the other remaining meta category value of interest, that is, that the salary’s role is that of a business fact. Collectively then, the data element:

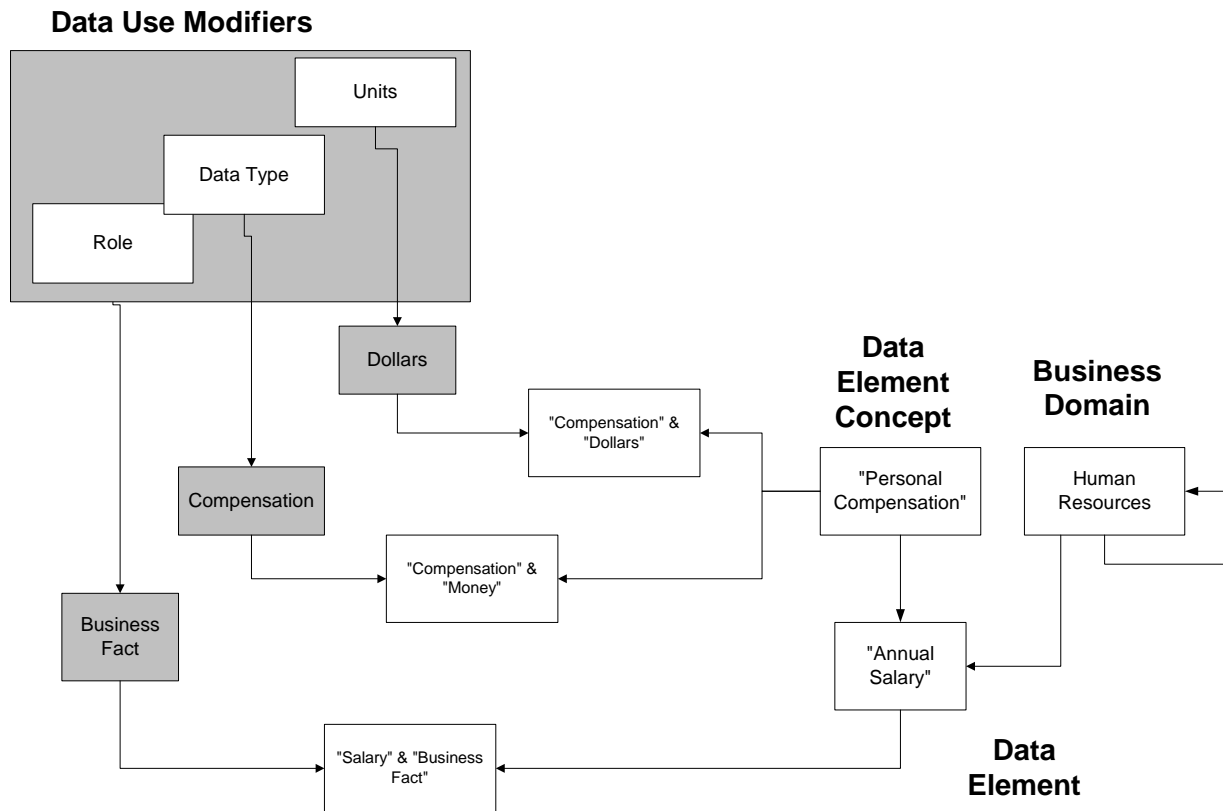
*Annual Salary is a business fact that represents compensation of money in dollars.*

The actual process of creating such a context independent business fact, that is, a data element consists solely of tagging the relevant:

- Business domain
- Data element concept, compensation

Then allocating the relevant meta category values and assigning the appropriate value domain. The result of this effort is that regardless of which context Annual Salary is employed, it’s semantics are automatically part of its nature. Every entity attribute, every table column (and by inference every DBMS column) is a use of the data element, Annual Salary.





**Figure 17.** Context Independent Data Element Salary with Associated Meta Category Value Types, Meta Category Values, Data Element Domain, Business Domain, and Common Business name.

But in the situation where a more general notion of salary is to be employed to govern attributes, columns (and by inference, DBMS columns), rather than name the data element Annual Salary, it should be named Salary. Further, no value domain should be assigned to the data element. When the data element Salary is employed to govern an attribute, then it could be more locally named Annual Salary and could be allocated the annual amount value domain range. Additionally, there could also then be attributes for monthly, or weekly, or hourly salaries. These too could then be allocated their appropriate value domains.

Finally, in the case of bottom-up development where the specified data models are yet to be defined, individual columns could be created that are annual, monthly, weekly, or hourly salary and the value domains allocated at the level of granularity.

Even in cases where a contained column is for example, Hourly Wage, its semantic data element template, Salary, produces the overarching set of semantics. In the case of hourly wage, it can have a more localized name and a more localized definition that denotes that this column, while semantically modeled after Salary, actually represents a semantic subset. That is, hour wages, versus monthly or annual salaries.

A final benefit from this approach is that all the semantic parts are actual “records” in the metadata repository that can be queried, selected, updated, and reported. This means that all



business facts that have to do with compensation can be found, and, in which ever context they occur, can be retrieved and printed. This represents a significant improvement over searching through metric tons of semantics to find names that contain compensation, only to find some that relate to some work environment that is constructed to adequately address a worker's special physical condition such as a special chair for a bad back

## 4.6 Data Element Classifications

Data element classifications are used to manage classification schemes and the classification scheme items that are in the classification schemes. A classification scheme may be a taxonomy, a network, an ontology, or any other system for systematizing where the categories are mutually exclusive. The classification may also be just a list of controlled vocabulary of property words (or terms). The list might be taken from the "leaf level" of taxonomy.

The classification scheme allows for a full bill of materials structure supporting both network and hierarchical classification schemes in support of an administered item. Administered items relate to one or more data elements.

## 4.7 Compound Data Elements

Compound data elements are data elements that are perceived as a unit but have subordinate contained units that are generally "invisible" to the uninitiated. For example, while it's well recognized that a telephone number in the United States consists of a country code, area code, exchange, and then a number, people rarely refer to those specific parts. Rather they say, that their phone number is "1.717.648.5913" as if it were all one continuous string.

Compound data elements are different from arrays, groups, or repeating groups because arrays, groups, and repeating groups all exist within a defined context while a data element is context independent. Thus, arrays, groups, or repeating groups would be defined, for example, within an SQL:1999 table, a screen, or a program.

An array is a multi-celled business fact in which all the values are of the same type and have the same meaning. Nicknames is an example of an array. The values might be "Shorty," "Junior," and "Slim."

A group is a set of individually defined business facts that collectively have a name and each contained business fact has its own name, data type, and definition. Each fact has only one value within the context of the group. Address, for example, in the United States is a group and it contains a number of contained business facts including street number, street name, floor, room or suite, city, state, zip, and country. Other components might include company name and subordinate company unit name.

A repeating group is a collection of individually defined business facts like a group. The main difference is that a repeating group may have more than one value instance. Employee\_Dependents might include for example, Social Security Number, Birth Date, First Name, Middle Initial, Last Name, and Gender, and for each employee there could be more than one dependent.

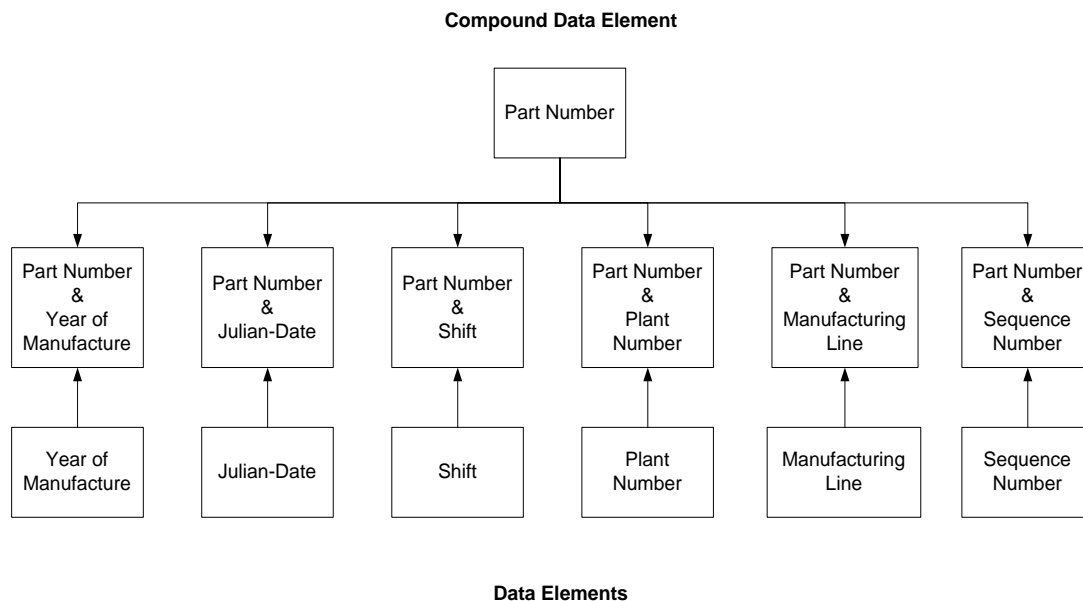


More complex data structures could include arrays within groups or repeating groups, and nested repeating groups.

None of these, however, are compound data elements. A telephone number is an example of a compound data element because it contains multiple subordinate units, but it's commonly referred to as a single data element. Part numbers, serial numbers, product-case codes, and the like are common examples of compound data elements.

In the context of data modeler, compound data elements do not have assigned meta category value types or meta category values. Rather, compound data elements are defined in terms of their subordinate contained data elements. Thus, in the case of phone number, it would be a compound data element and would be defined into its most widely recognized subordinate parts. That is, Country Code, Area Code, and Phone Number. The telephone number's exchange, while important to the telephone company is generally unimportant as a distinguishable sub-component, despite being separated by a period or a hyphen from the remaining part of the phone number.

The meta data model in Figure 14 shows that a compound data element is a Bill of Materials data structure that consists of nested sets of subordinate compound data elements. The figure also shows that a given data element may participate in more than one compound data element. This only makes sense as the "real" data is actually defined within a data element, and if there is one compound data element that makes use of some "real" data element there are probably many other compound data elements employing the same real data element. With this scheme, access to all the different compound data elements is practical and reasonable.



**Figure 18.** Meta data records required to represent the compound data element, part number



Figure 18 illustrates the meta data records that would be stored to represent the compound data element, Part Number<sup>4</sup>, which contains, Year of Manufacture, Julian-Date of Manufacture, Shift, Plant Number, Manufacturing Line, and Sequence Number. In this example the data elements are stored, the compound data element is stored, and then the intersection records between the data elements and the compound data elements are stored. While this certainly appears to be a lot of work for such a simple concept, the value proceeds from two sources. First there is now firmly recorded semantics about each of the component parts of the compound data element, and also for any other compound data element that involves one or more of the contained data elements, the only effort is to store the compound data element and then the intersection record to the appropriate data element. For a one-off example, it is more work. But for larger work units starting with a project it is less work. Across the enterprise, the benefits in terms of risk, cost, quality, and productivity approximate those presented in Figure 2.

## 4.8 Derived Data Elements

A derived data element is a data element whose value has to be calculated from other data elements. An example could be the *age-at-death* of a person. This value is clearly computed from the difference in terms of years between the *date of birth* and the *date of death*. The reason this might be of interest is that if a person died on February 29<sup>th</sup> of a leap year, then in all other years the person might have would have a year older when they died if they had been born on February 28<sup>th</sup> of a non leap year. Derived data is best represented in terms of the actual data elements that comprise the calculated result along with a formula or process that would determine the derived data result.

Figure 14 shows how a derived data element is mapped to the set of data elements from which it is calculated. Additionally, Figure 14 shows that a derived data element may be a component of one or more compound data elements, and vice versa.

Figure 19 illustrates the meta data records that would be stored to represent the derived data element, Age at Death, which relates to the two data elements, Birth Date and Death Date. The implementation of process logic that computes the difference between these two data elements would be placed in the columns of the relevant tables. The process logic specification is a type of data integrity rule that becomes a condition on the successful valuation of the data element, Age of Death, in whatever environment it is implemented. That is, no value would be allowed until there was a Date of Death, and then the value would be automatically calculated and stored.

---

4

Readers of this paper should please refrain from sending Whitemarsh email regarding the quality or lack thereof relating to the definition of this compound data element. It is provided for illustration purposes ONLY.

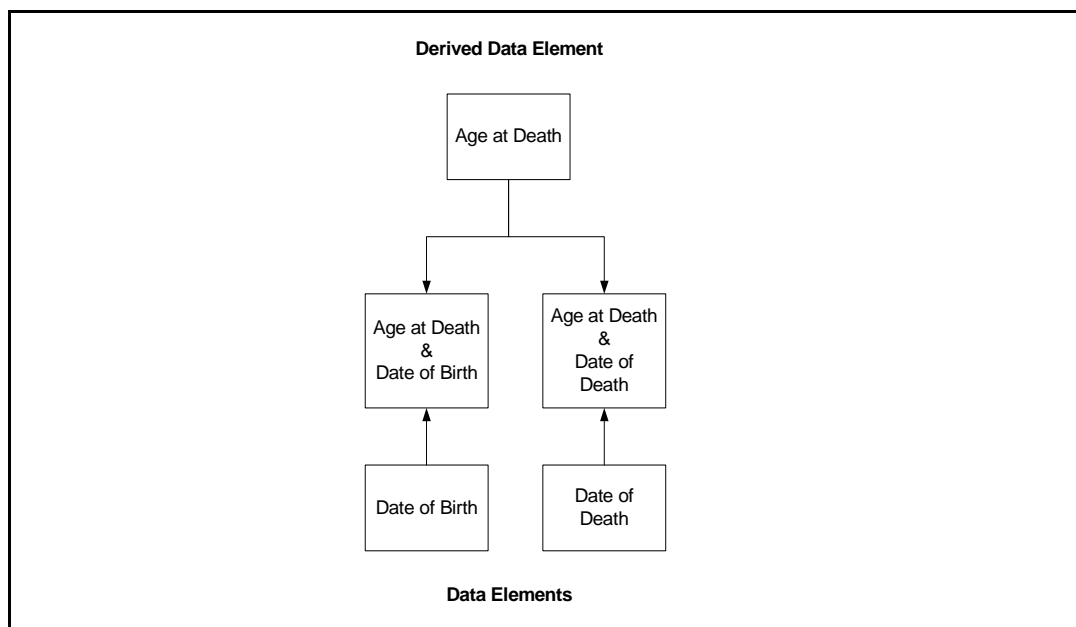




## 4.9 Data Element Model Process Model

The overall processes supporting the creation of data elements are:

- Creating data semantics
  - ◆ Create meta category value type classes
  - ◆ Create meta category value types
  - ◆ Create meta category values
- Creating elements upper level metadata
  - ◆ Create concepts
  - ◆ Create conceptual value domains
  - ◆ Create data element concepts
    - Assign data element concept meta category values
  - Create value domains
  - Create data element classifications
- Creating data elements
  - ◆ Assign data element value domains
  - ◆ Assign data element meta category values
- Create compound data elements



**Figure 19.** Data elements involved in derived data element, Age at Death

- ◆ Create compound data element structure types
- ◆ Create compound data element structures



- - ◆ Assign data elements to compound data elements
- Create derived data elements
  - ◆ Assign data elements to derived data elements
  - ◆ Assign data elements to compound data elements
- Perform reverse engineering
  - ◆ Reassign data elements to business domains
  - ◆ Reassign data elements to data element concepts
  - ◆ Reassign data elements to value domains
  - ◆ Reassign data element concepts to conceptual value domains
  - ◆ Reassign value domain to conceptual value domains
  - ◆ Promote data elements to data element concepts
  - ◆ Promoted data element concept to concept
- Perform exporting and importing

There are two strategies for creating data elements: Forward engineering and reverse engineering. Under a forward engineering scenario data element metadata would be created in a top-down fashion. This generally follows with the processes listed above in generally that same order. It is unlikely however that once a cache of data elements was created that it would definitively serve the enterprise. This process would likely have to be supplemented by a reverse engineering process.

Under a reverse engineering strategy, a key legacy schemas would be obtained and then their SQL data definition language schemas would be imported into the operational data model then promoted into the implemented data model. Thereafter the columns would be reviewed and as a common one is discovered the reverse engineering process of Promote Column to Data Element would be executed. Then the upper levels for that data element would be created. That is, creating

## 4.10 Data Element Model Summary

Data elements are context independent business facts that are used as semantic templates for context dependent business facts like column in tables, fields in files, cells in screens, and variables in programs.

Properly defined, data elements can be of great value to the enterprise's mission of achieving data standardization. The following example proves this assertion. If estimates are properly done, information technology project managers should allow about two staff hours to fully define each context dependent business fact, that is, a column in a table, a field in a file, a cell in a screen, or a variable in a program.

If there are 100 tables of 15 columns per table, and 200 screens with 15 screen cells on each, and 300 programs with about 30 variables per program, the quantity of these context dependent business facts is 13,500. Thus, the quantity of staff hours that should be expended on context dependent business fact definition should be about 27,000 staff hours. That's about 14 staff years with no time off for good behavior.



Given that no project manager will ever presume that such an estimate will be accepted, either an unrealistic estimate will be made, such as 15 minutes per context dependent business fact, which brings the estimate down to about 4,000 staff hours, or the context dependent business facts just won't be defined. Since the 4,000 staff hours would also never be accepted, then, in virtually 100% of the cases the context dependent business facts will not be defined. Clearly, not a proper solution to a very real problem. Not documenting the various business facts in all their contextual uses is professionally irresponsible. Y2K was the perfect example of the consequences of ignoring data standardization.

Given that there are only about 20% "real" data elements across a population of columns in tables, then the 1500 columns is immediately reduced to just 300 data elements. If two hours is expended for each, then that's just 15 staff weeks. But given that a database is largely a semantic clone of a number of other databases, then the population of 300 data elements probably already has 75% defined. That reduces the number to just 75 data elements, or about 4 staff weeks.

Since virtually all the other context dependent business facts, that is, fields in files, cells in screens, and variables in programs are uses of already defined columns in tables, which are now almost automatically defined through the 4 staff weeks for defining the few data elements not yet defined, then the data definition problem is almost largely non-existent. Now, that's a proper solution to the very real and serious problem.

Finally, through this approach, the actual work is so completely integrated into the natural work of the analysis and design members, that the actual time to accomplish all this data standardization effort will actually be **NEGATIVE** because of the normal acceleration in productivity due to reductions in:

- Research
- Rework
- Presentations
- Documentation preparation

Simply stated, there are no down-sides to this approach. It reduces risk, increases productivity, and increases quality. The approach is all benefit at virtually no cost. It is work accomplished at the rate management expected it to be done in the first place.



## 5.0 Specified Data Model

The specified data model represents the technology independent representation of a set of data structures that are considered important to the enterprise. These can be used by enterprise data modelers to standardized data structures across all implemented and operational data models. Specified data models thus represent templates in the same way that a data element represents a business fact template for attributes of entities or columns of tables. In addition to all the appropriate requirements for quality data models, the key requirement for the specified data model is that it can be implemented through a variety of technologies such as DBMS, spread sheet tables, traditional access file structures, computer program embedded temporary files and memory arrays, and the like.

As might be expected, Whitemarsh only emphasizes DBMS as an appropriate implementation technique for data models, whether the DBMS is present on a personal computer, server, or mainframe. Computing advances have brought about the gradual disappearance of the sophistication demarcations among these three DBMS platform types. In the early 1970s, mainframe DBMSs were processing four to six transactions per wall clock second. Today, personal computer based DBMSs are processing hundreds of similar transactions per wall clock second. Additionally, personal computers now have disk space, memory, and processor speeds that were far distant dreams within mainframe environments of just the mid 1980s.

As a consequence of the disappearing technology differences, the main differences that are present among the three data model classes (specified, implemented, and operational) are those imposed by the different natures of the databases represented. For example, there may be a larger quantity of small-in-scope data models within the sphere of specified data models because they are mainly representational of models of data for certain well defined subject areas. For example, there could well be a data model for finance, and another for human resources.

Implemented data models on the other hand conform to the specific requirements of a database that has to be actually implemented. Thus, the database might contain data structures from many different specified data models. Inclusion of many different models can be justified on the basis that a implemented database collectively embraces the four specified data model areas for Sales, Market Management, Customers and Customer Product Distribution. Because of this real difference between specified and implemented data models, it should be clear that the differences between them is not just related to transformation. That is, from “conceptual” to “logical” to “physical.” While that may be the case in some trivial situations, it is not the most important or useful distinction between these models.

The development and maintenance of specified data models is a measure of the data management sophistication of the enterprise. Enterprises that recognize the value of enterprise-wide database are willing to invest in the development of enterprise-wide data semantics in order to receive the benefits of lowered risk, increased productivity, and enhanced quality.

Implemented data models can certainly be created without first developing specified data models. So also can operational data models be created without first developing the implemented data models. In these cases, however, the mapping between the columns of tables within the implemented data model or DBMS columns of DBMS tables within the operational data models to the respective specified and implemented models is to “unknown semantics.” This strategy of



mapping to “unknown semantics” may well be necessary and useful when first incorporating a commercial-off-the-shelf (COTS) package data model. The Whitemarsh data modeler can read SQL/DDDL streams and will impute the operational data model from that. Most likely, the COTS data model will be a melding of two of the data architecture classes: original data capture data model and subject area data model.

Enterprises that continuously develop databases through data models at the operational level without either first creating the specified and implemented data models or mapping to these models will ultimately pay a very steep price for this ad hoc data modeling behavior. As stated in Section 1, a United States Department of Defense agency now pays \$175 million per year for its ad hoc data definition behavior, and, that’s just the cost of extra information technology resources, not the cost of lost opportunity or the expense incurred by not having quality or up-to-date information for decision making.

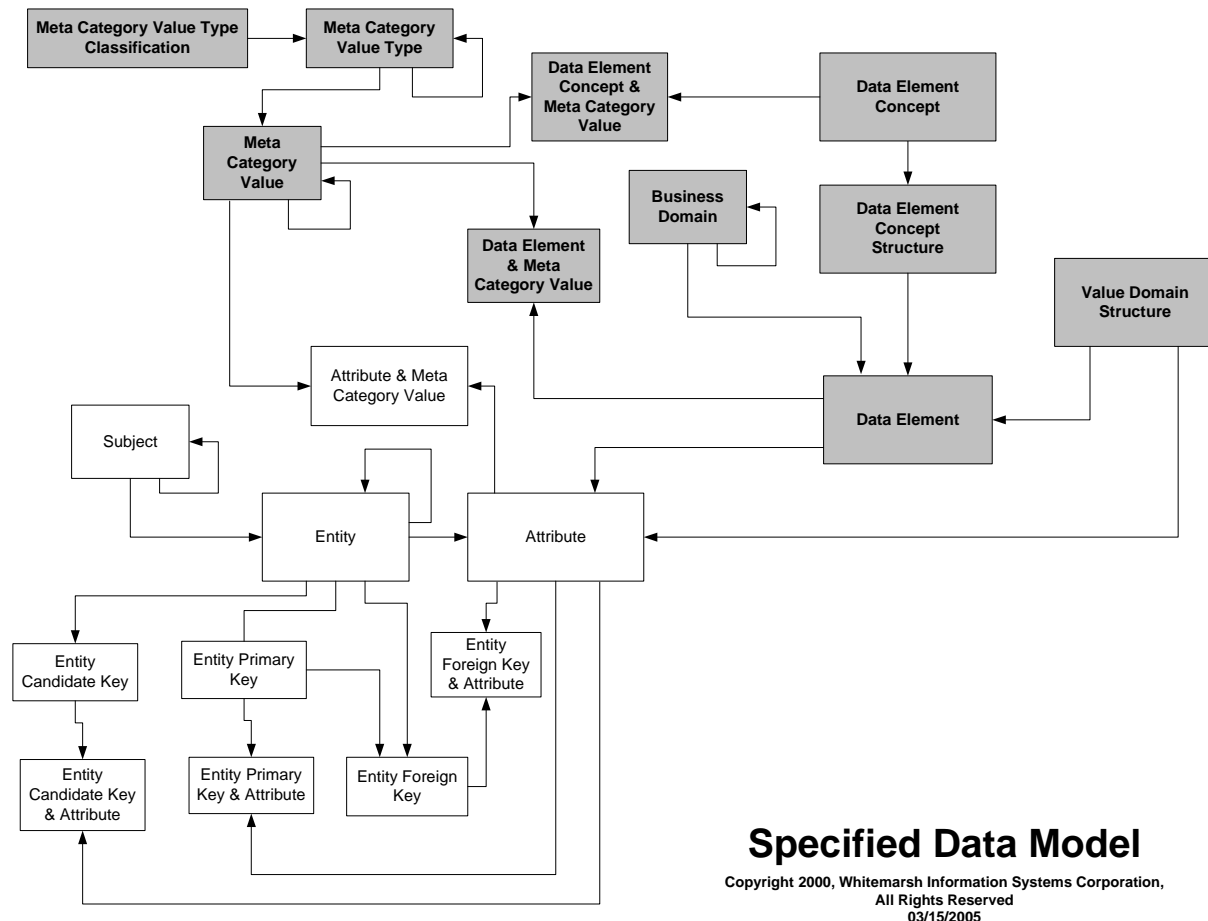
The meta model design for the specified data model is presented in Figure 20. This diagram contains the following meta entity groups:

- Meta Category Value Types and Meta Category Values (upper left)
- Data Element Domains and Value Domains, Data Elements and Value Domains (middle and right)
- Subject, Entity, and Attribute (middle left)
- Primary and Foreign Key Support for Entities (lower left)

The first two meta entity groups are shaded on the diagram. This means that they should already be valued in support of the data stored in the specified data model. In case there is a need to map an attribute to a data element that’s not already defined, the Whitemarsh data modeler supports two options: The process of automatically creating a data element from an attribute, and the explicit definition of new data elements through the data element module. This module may be under special security to prevent the ad hoc definition of semantics. In such a case, the attributes can still be added, but when they are without the appropriate mapping, reports containing these unmapped attributes indicate that they are unmapped to standard data semantics.

Shaded meta entities are not allowed to be updated within the specified data model module. Rather they are accomplished in either the meta category value hierarchy classes, or in data element modules. The reason these functions are segregated is to prevent ad hoc data modeling behavior by allowing separate security over each distinct data modeler module.





**Figure 20.** Specified data model meta model.

## 5.1 Subject Area, Entities and Attributes

The process of creating the specified data model consists of creating subjects, entities within those subjects, and then attributes within entities.

### 5.1.1 Subjects

Subject areas, within the context of the specified data model is a method of classifying entities. For example, the subject area, Finance, contains two subordinate subject areas, Accounts Payable and Asset Management. Subjects can be hierarchical. Entities are tied to the leaves of the subjects. Entities are thus directly tied to accounts payables or to asset management, but not to finance.



## **5.1.2 Entities**

While most all the data modeler material to this point concentrates on data elements and the semantics attached thereto, the most valuable component of a data model is the entity. An entity should represent a well defined and obviously recognizable component of business policy that is instantiated, modified, selected, and reported. Entities are of course recognized through the data values present in its contained attributes, hence the need for business policy rooted and easily recognizable data element semantic template from which to model the attributes.

The methodology for developing quality entities is not the subject of this book. There is a good amount of Whitemarsh material that addresses that process. As a summary of the entity discovery process within the Whitemarsh methodology, candidate entities are initially derived from database domains. Database domains are noun-intensive descriptions of the leaves of mission descriptions. Mission descriptions are the essential end-result characterizations of the essence of the enterprise.

While it appears that entities are thus a third level hierarchy product from missions to database domains and then entities, in fact, the candidate entities are collected and a non-redundant set is produced. Entities are thus related to database domains through a many-to-many relationship. The non-redundant collections of policy-related entities are subject areas.

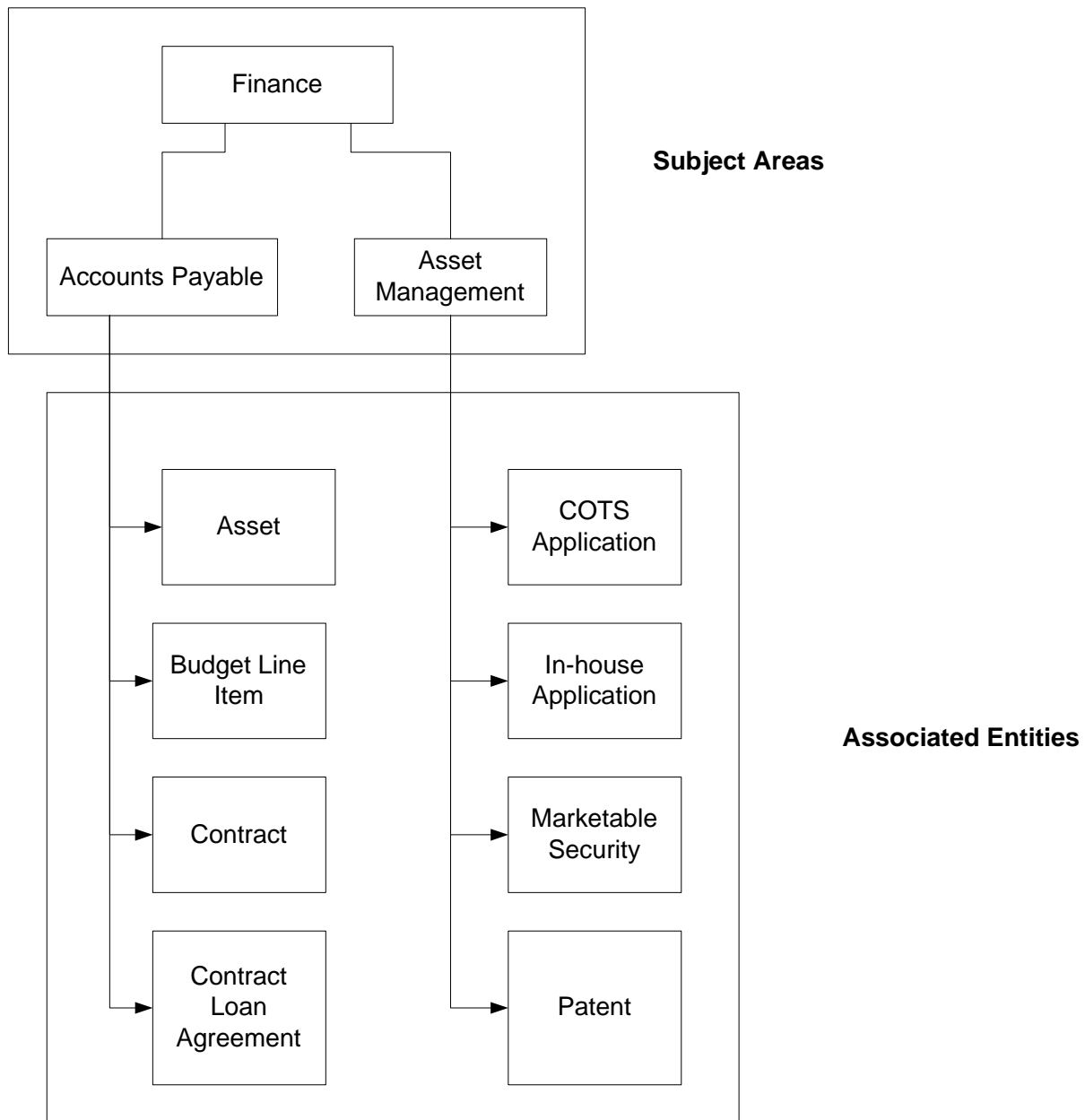
When that process is accomplished, other parts of the metabase are filled in with very critical metadata. The metabase module user guides that readers should consult are:

- Resource Life Cycle Analysis
- Business Information Systems
- Mission, Organization and Function
- Information Needs Analysis
- Database Objects

Entities are identified within a specific subject area. Figure 21 illustrates the allocation of two groups of entities to the two different subject areas. Subject areas are merely a mechanism for classifying the entities. In this case, the subject area, Finance, contains two subordinate subject areas, Accounts Payable and Asset Management. Associated with these two subordinate subject areas are a set of entities. This list is not a data model is merely a grouping of entities by subject area.



While entities are properly defined within one subject area, for example, finance, entities from different subject areas such as employees are able to be interrelated through primary and foreign keys. Thus, all entities that are directly and indirectly associated with a particular subject area can be reported. In this case, the directly associated entities for the subject area, accounts payable within finance, would include asset, budget line item, and contract, and the indirectly



**Figure 21.** Subject Area and Associated Entities

associated entity, employee, would be from the human resources subject area.





Entities can also be subtyped. For example, there may be a real-property entity that is to represent a property in general and also the specialized attributes for an apartment or for a house. The root entity would contain all the attributes common to all types of properties, and the individual entity subtypes, one for apartment and the other for house would contain the attributes special to each. Subtyped entities are not related to each other through keys.

### 5.1.3 Attributes

Once the subject areas and entities are identified, the task of creating attributes starts. At this point, the basic definition of data element comes into play. That is, a semantic template for the business fact attributes within an entity. Attribute definition is accomplished in two stages:

- Employment of the data element semantic template along with the entity to create an attribute.
- Refinement of the attribute's meta attributes<sup>5</sup> through the creation of its remaining meta attribute values.

This first stage is accomplished through the Whitemarsh metabase technique of tagging. A three list window is displayed. In the upper left list is the list of entities. In the upper right window is the list of data elements. An entity is tagged and then one or more data elements are tagged. Once the "build" button is pressed the attributes are built and displayed in the bottom list that runs across the full screen. That's all there is to it. Because of inheritance, all the semantics of the data element are automatically assumed by the attribute. Figure 22 displays the result of associating the data element, Salary, with the entity, Employee. The semantics that are automatically inherited are:

- Data element domain and value restrictions
- Data element and value restrictions
- Data element's business domain
- Data use meta category value types and values associated with the data element
- Entity subject areas
- Entity

All these attribute semantics are present solely because of tagging and selecting already defined semantic instances from other meta entities. There's been no keying of any additional data. In addition to all these inherited semantics, all the definition fragments associated with these meta entity types are employed to provide a full definition of the attribute.

---

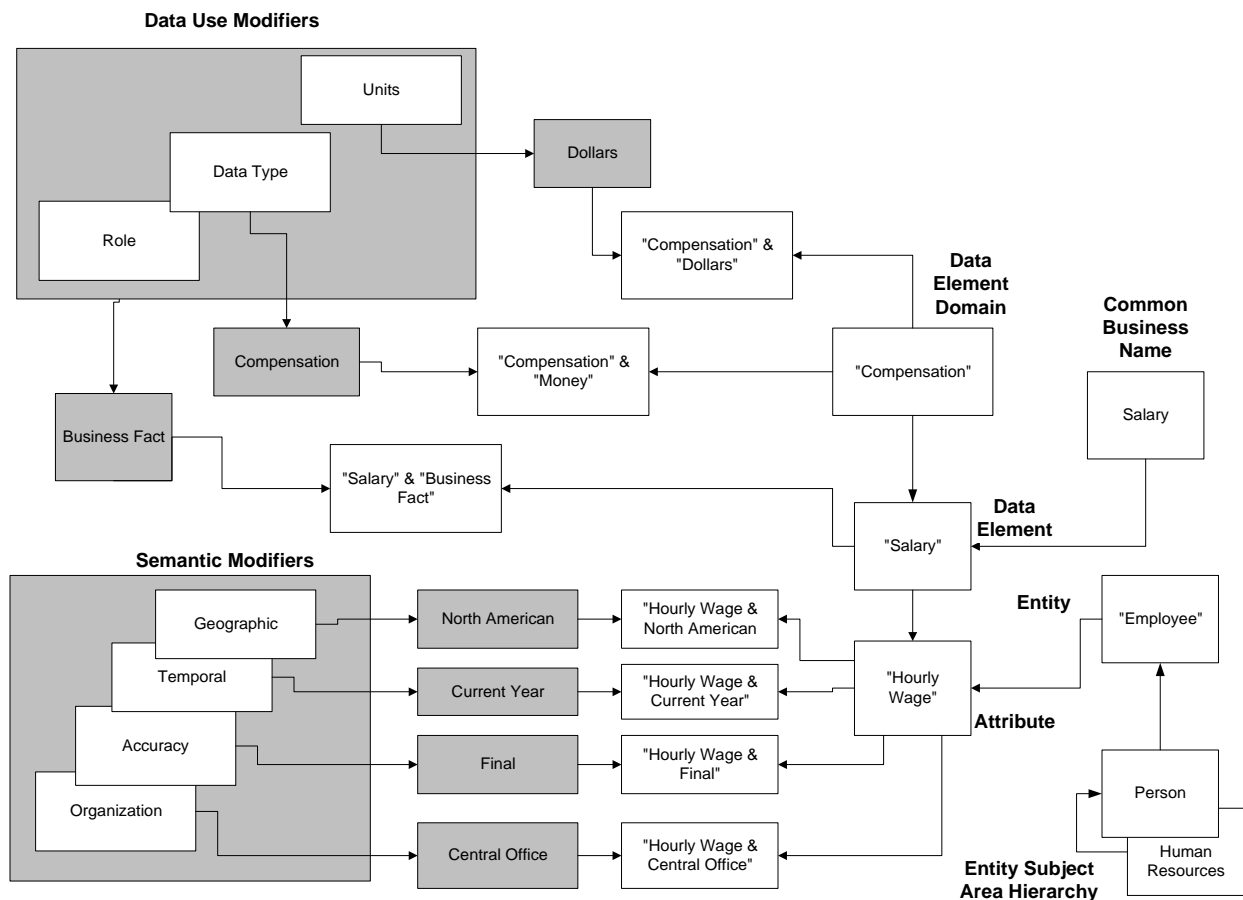
<sup>5</sup>

An unfortunate circumstance of a metabase (or any other form of meta modeling) is the use and reuse of the same terms. For example, the attributes of an entity contain meta attributes.



The second stage involves allocating the semantic modifiers meta category value types and meta category values. Semantic modifiers provide additional restrictions on the contextual use of the attribute. In this example, the intended attribute, Hourly Wage, applies to a particular class of person, that is, employee, who works within other semantic constraints such as organization, temporal, accuracy, and geography. For these semantic modifiers, the salary is for the Central Office from Organization, the Current Year from Temporal, Final from Accuracy, and North America from Geography. Collectively then the complete set of semantics for the hourly wage attribute is graphically depicted in Figure 22, and presented in a tabular manner in Figure 23.

From the point of view of a single attribute within an entity, Figures 22 and 23 represent a significant amount of semantics that becomes available solely on the basis of inheritance from the other meta entities to which the attribute is associated. When all the definition fragments



**Figure 22.** Fully Selected Semantics for Hourly Wage Attribute of Employee Entity



Attribute: Hourly Wage		
Inherited Meta Attribute Class		Inherited Hierarchy of Values as Appropriate
Data element domain		Compensation
Business domain		Human Resources
Common Business Name		Salary
Data Element		Salary
Data use modifiers	Role	Dollars
	Data Type	Compensation
	Units	Business Fact
Semantic modifiers	Geography	North American
	Temporal	Current Year
	Accuracy	Final
	Organization	Central Office
Entity Subject area		Human Resources
Entity		Employee

**Figure 23.** Full Associated Semantics for Hourly Wage Semantics with Employee Entity

from all these inherited semantics are collected together into a loose definition, nothing more should be required. In short, the only thing that may have to be originally entered is the local name of the attribute in the column. And, even in most cases, the common business name that is inherited from the data element may be sufficient.

In the case, however, when a data element is associated several times within an entity, then while there may not be the need for additional or different data use or semantic modifiers there still needs to be some additional information to distinguish the different intent from what would appear to be “identical twin” attributes.

For example, suppose that the data element, telephone number is allocated to the employee entity four different times as a way to represent the telephone number for the employee’s home, office, cell, and fax. In this situation, it may well be that only a slightly different local attribute name is necessary to represent the difference. So, data modeler provides



the ability to change the local name for the attribute from its default name, that is, the data element name (which in turn has as its default name the common business name) to some other name. In this case, the local names would most likely be, home telephone number, office telephone number, cell telephone number, and fax telephone number.

The reason these four different phone numbers are not four different data elements is both simple and practical. Simple because the essence of each is really just a telephone number. Practical because if every attribute is a data element then so too would likely be every column and DBMS column. This results in an impossible to accomplish and ever expanding task. Finally, given that there are bare enough resources in most organizations to accomplish the essential and important activities, then these certainly are no resources to devote to what is arguably inessential and unimportant activities.

## 5.2 Key Support

The specified data model supports three types of keys:

- Primary
- Foreign, and
- Candidate

There are two classes of primary keys. The collective value from each class represents a unique value, which when used to select a row of data from a table (or figuratively from an entity), results in only one row of data. The two primary key classes are:

- Surrogate key attribute
- Business attribute set

The surrogate key attribute is a crafted attribute whose value has no business meaning whatsoever. A surrogate key is always a single attribute and takes on names like Employee\_Id, or Invoice\_Id, or Customer\_Id. The form of the name for the attribute is <entity-name> + <Id>. This type of primary key should only be employed on implemented and operational data models.

***A surrogate key must never, never, never be used within the specified data model to “describe” the set of attributes whose values in combination create a unique value string through which a single rows (figuratively, of course) is obtained from the entity.***



The reason is simple: how will you ever know that you have achieved at least third normal form<sup>6</sup>? Third normal form is the minimum data quality characteristic that must be achieved in any database design of a specified data model.

The Business attribute set is that set of business fact attributes, which in combination produced a unique value across all rows for that particular entity. The business fact attribute set is what must therefore be used to fully identify and construct the entity's primary key or for use in defining the foreign key that must map back to the primary key.

### 5.2.1 Primary Keys

The primary key is that set of business fact attributes, which in combination produced a unique value across all the inferred rows for that particular entity. For example, for an employee entity, the data modeler [person] must examine the attributes assigned to the entity and ensure that it is third normal form by determining the set of naturally existing attributes that comprise the primary key. A candidate set of attributes would be the employee's full name, that is, first, middle and last. But for a large enterprise that may not be enough. Added may have to be the employee's birth date. Maybe that's not enough. Probably the last added attribute would be the person's birth location which might be city, state, and country.

Since some enterprises might feel the analysis to determine the "natural" primary key for an enterprise excessive, they often rely on a governmental agency to provide some sort of unique identifier for the person, such as the Social Security Number in the United States. While this is generally considered as really some form of a surrogate key, but just from a Governmental agency, it is still accepted as the employee's business attribute primary key. While other entities may also have naturally existing single attribute primary keys such as order number, and invoice number, some other entities may not. For example, entity, project assignment, may not have a naturally existing single attribute primary key. At first blush, the key might be project name (assuming that's naturally unique) and employee number. But if the person is assigned to the project more than once, the assignment date may have to be added. But, if a person is assigned to a project more than one time on the same date to serve in multiple roles, then the role name that the person is performing may also have to be present. In short, the analysis requires a good knowledge of the data.

---

6

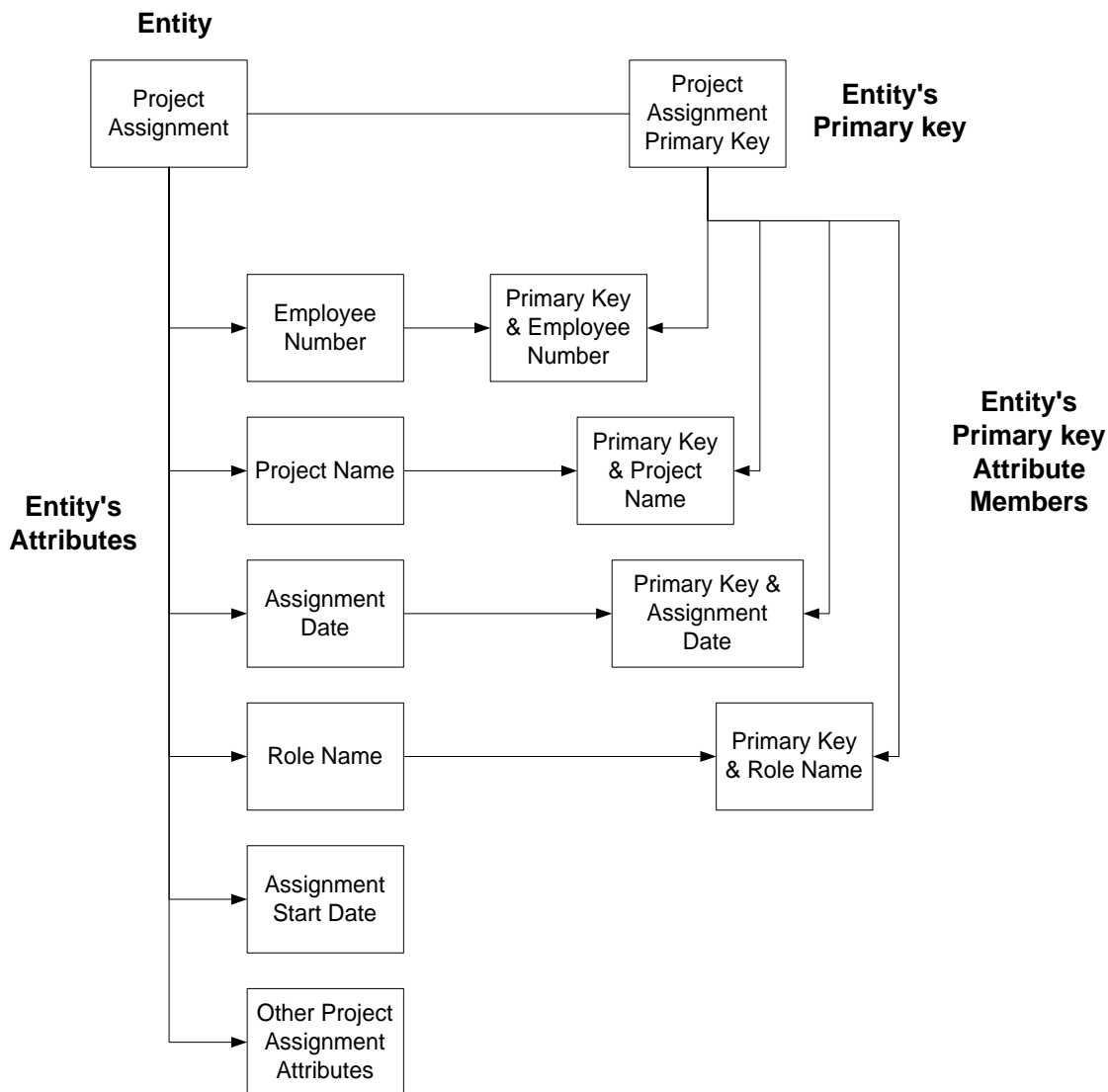
There are five normal forms. The first normal form represents data that has been organized into two dimensional flat tables. The second normal form represents data that is in first normal form and that additionally has all its non-primary key columns functionally dependent on the entire primary key. The third normal form represents data that is in second normal form and that additionally has no dependencies between any non-key columns and any other non-key columns. The fourth normal form represents data that is in third normal form and that additionally has its primary key related to all columns in the table such that it contains no more than one nontrivial multivalued dependency on the primary key. And, the fifth normal form represents data that is in fourth normal form and that additionally has the characteristic that if its primary key is a concatenated key such that none of the components of the concatenated key can be derived from another component of the concatenated key.



As to the creation of the primary key, Figure 24 presents the entity, attributes, primary key, and primary key attribute rows of data that illustrate the creation of the primary key for the project assignment entity.

In this example, the entity, project assignment is related to its primary key, Project Assignment Primary Key through a one-to-one relationship. That is because there is only one primary key for each entity. Associated with the primary key are the attributes from the entity that are members of the primary key. The set of values from these attributes become the value of the primary key. In this example, the primary key value would be: <Employee Number> + <Project Name> + <Assignment Date> + <Role Name>.

Each primary-key associated entity-attribute also contains a sequence number so that the



**Figure 24.** Meta data records that define primary key for Project Assignment Entity



exact value will be known. For example, the value string from Last Name, First Name and Middle Initial will produce a different value string than First Name, Middle Initial and Last Name. Thus, sequence is a necessary component of the primary key's semantics.

This particular example illustrates another benefit to the Whitemarsh approach to data modeling. Assume that the real basis for the company's employee number is the Social Security Number. In this particular case, its data element semantic template is Social Security Number. The fact that the attribute's name is Employee Number does not take away the simple fact that the numeric string is really not just some arbitrary computer generated identifier. Thus, through this approach, the metabase can be queried to find the names for all attributes (and later on for all columns and DBMS columns) that are based on the data element Social Security Number. That's a good first step down the road to data standardization.

This approach, because it is rooted squarely in database, enables reports, queries, and updates to be done through standard reporting and query facilities. Thus, the query, "In which primary keys does the Social Security Number participate?" can be quickly and easily answered.

## **5.2.2 Foreign Keys**

The foreign key is that set of business fact attributes, which in combination produced a value which maps exactly to a primary key from another entity. In the special case where the foreign key from an entity maps to the primary key from the very same entity, then the relationship is deemed to be recursive.

Figure 25 presents the data model that for the three entities, project, employee, and role that are "parents" of the entity, project assignment. Figure 26 then presents the metadata records that are implied by the meta data model in Figure 14 that support the relationships among these entities.

In Figure 26, the entities, Project, Employee, and Role are along the left side. Their attributes are depicted below them. Their primary keys are depicted to the right of each entity. The foreign keys for the Project Assignment entity is presented to the left side of project assignment. There are three foreign keys, Project Foreign Key, Employee Foreign Key, and Role Foreign Key. The meta records for each is immediately below. The relationship between the primary key of each parent entity is depicted. For example the relationship between the Project Primary Key and the Project Foreign Key.

From the presentation in Figures 15, 25, and 26, the required meta model relationships are:

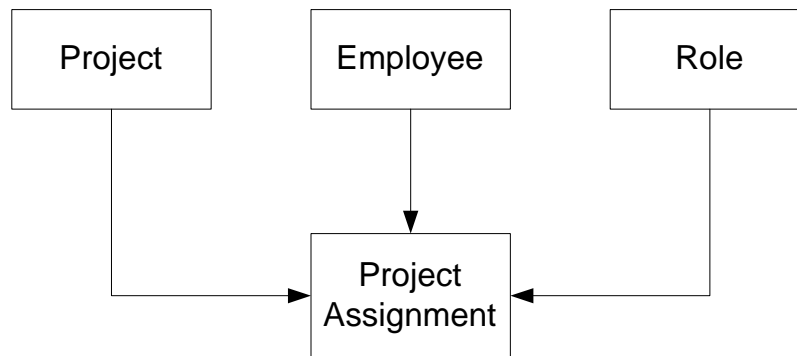
- A one-to-one between an entity and its primary key
- A one-to-many between any primary key of one entity and any related foreign keys from one or more related entities
- An intersection record relationship between any primary key and its related columns



- An intersection record relationship between any foreign key and its related columns

While this may seem like a lot of work, it is entirely accomplished through the use of tagging. In fact, for the accomplishment of the foreign keys, the Whitemarsh data modeler module merely requires the selection of the “parent entity.” All the intersection records and attributes are automatically created in the “related-to” entity. The default names for the attributes are the concatenation of the following values:

- The Source Entity Name
- The Phrase Must or May
- An action phrase
- Target entity Name
- Attribute Name from the corresponding Primary Key Attribute
- The String, “reference”



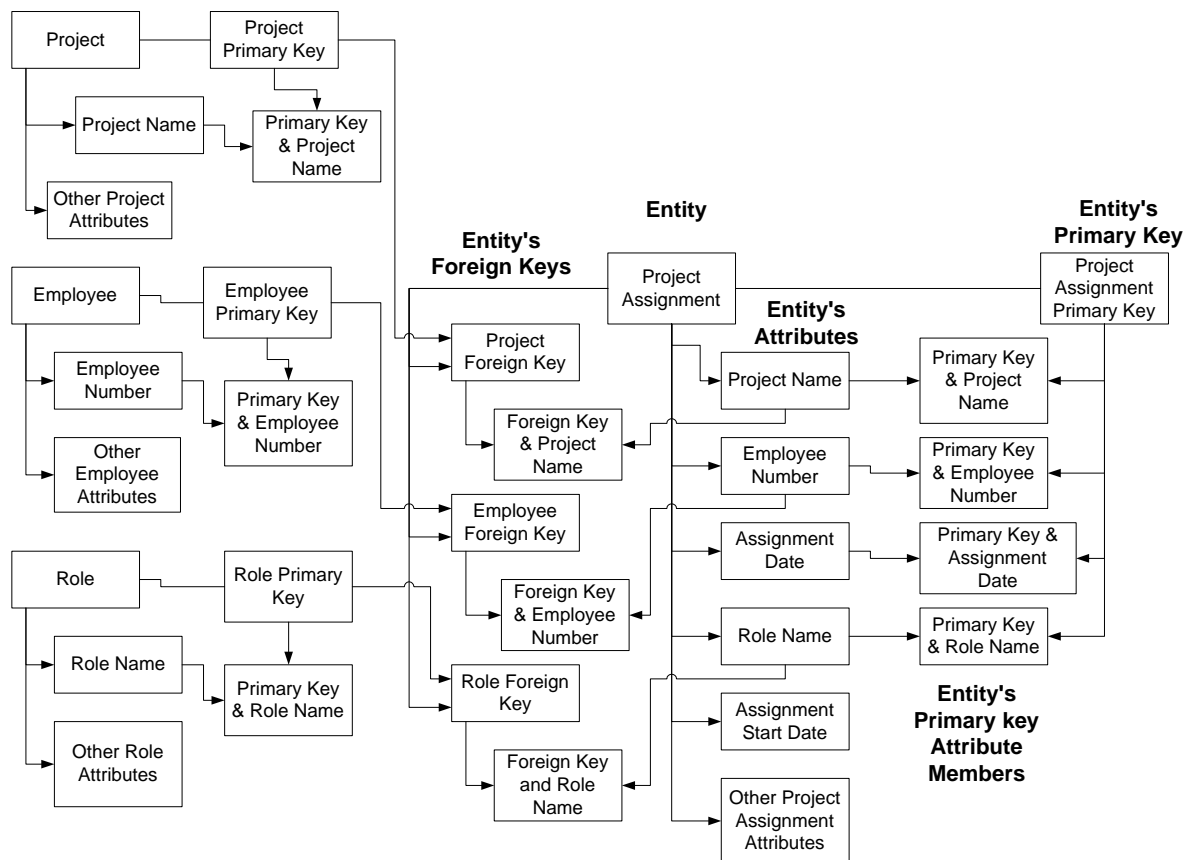
**Figure 25.** Data Model for Employee, Project, Role, and Project Assignments

The foreign key itself is automatically provided a name and it is the concatenation of the following:

- The Source Entity Name
- The Phrase Must or May
- An action phrase
- Target entity Name





**Related Entities and their Primary Keys**

**Figure 26.** Meta data records required to support primary and foreign key relationships among Employee, Project, Project Assignment, and Role.

### 5.2.3 Referential Integrity

Referential integrity is simply a statement of the rules that control the integrity of a reference that exists between the primary key from one entity and the primary key's presence as a foreign key in one or more related entities. The purpose of referential integrity is control the consequential actions taken when an update occurs to either a "child" or a "parent" entity.

The referential action rules are specified through controlled drop down lists when the foreign key is selected. Appendix 1 of this book presents the referential integrity rules that are supported by the metabase data modeler.

If two tables are the owner of another table, only one table can have the referential integrity action, cascade delete. If cascade delete is attempted for a second connected table, an error message will appear. Only after the cascade delete action is removed from an existing table can another cascade delete action be placed. The reason for this rule is to eliminate automatic deletion ambiguity.



No attribute that is part of primary key is allowed to be set-null as a referential action. While an attribute within a primary key may be allowed to be “set-null” in certain DBMSs, it is considered bad data modeling and is thus disallowed within the Whitemarsh data modeler. Except for foreign keys with the referential action of set-null, all foreign key attributes are automatically set to null not allowed. The only way to change the “nullness” of a foreign key attribute is to change the referential integrity action on the foreign key.” No attribute of a foreign key is allowed to be an attribute of a primary key. Primary key attributes and foreign key attributes are thus, orthogonal.

The choice of referential actions clause not only directly affects the behavior of the DBMS, it also changes the form of the “story” that is created for the foreign key’s name. The foreign key name is automatically created and takes on the form, <source entity name> <optionality phrase> <action phrase> <target entity name>. Foreign key attribute names are defaulted to <source entity name> <optionality phrase> <action phrase> <target entity name><primary key attribute name><‘reference’>. For referential actions that imply optionality, for example, one that is defined with SET NULL, the optionality phrase employed is “may.” Whenever the referential action chosen is mandatory such as with CASCADE, SET DEFAULT, RESTRICT, or NO ACTION, the optionality phrase is “must.”

#### **5.2.4 Referential Integrity**

Entity candidate keys represent a collection of attributes within an entity that when their values are collectively employed would result in the retrieval or update of a single row of data for that entity if that entity had actually been a table. There may be multiple candidate keys within an entity. Attributes of candidate keys are not allowed to overlap each other or the entity’s primary key.

### **5.3 Attribute Value Domains**

The process of assigning value domains to attributes is similar to that of assigning value domains to data elements except for that the assigned value domain must be a direct descendent of the data element’s value domain. This is clearly illustrated in Figure 27. The value domain for attribute hourly wage of the entity employee is a subset of the value domain for the data element salary. This permits value domain deployment tracking from its broadest definition to its most refined allocation. If an attribute did not have a value domain then the value domain assigned to its data element is automatically inherited by the attribute.





In a similar fashion, attributes may need to be reallocated to different data elements because of creation, refinement, and recasting of existing data elements. Finally, an attribute may surface that is really a data element. The promotion process causes a data element to be created from the essential semantics contained in the attribute. Once the data element is created, it is connected to the attribute, and all the semantics associated with the newly created data element are stripped from the attribute. These are still inherited by the attribute as the data element is its semantic template.

## **5.5 Specified Data Model Maintenance**

Maintenance of the specified data model conforms to a set of rules. These are:

- All primary key attributes are only not-null.
- Because of automatic foreign key attribute generation from the primary key, all foreign key attributes are initially set to not-null. If, however, the referential action for a foreign key is “set null,” then the foreign key attribute is automatically set to null-allowed.
- When an attribute is directly deleted, it can be deleted only if it does not participate in a primary or foreign key.
- When a foreign key is deleted, so too are all its attributes.
- When a primary key is deleted, all its attributes are deleted, and so too are all associated foreign keys and their attributes.
- When an entity is deleted, so too is its primary key, primary key associated foreign keys, entity contained foreign keys, all non-key attributes. An entity can be deleted if and only if its attributes are not related to any column within the implemented data model
- Whenever an attribute is deleted, deleted also are the associated semantics for meta-category values. An attribute can only be deleted if it is not related to any column of any table in the implemented data model.
- Whenever a subject is deleted so too are deleted all entities and contained attributes and keys. However, a subject can only be deleted if no entity has an attribute that is related to any column of any table in an implemented data model.



## 5.6 Specified Data Model DDL and Graphics

The data modeler, via the data model tree menu selection, supports a graphical tree-rendition of a data model. You can also cause the creation of an ASCII text file SQL data definition language if the selected specified data model.

The first graphic is presented immediately within the window as a graphical tree. In these graphic trees, the targeted root entity upon which the dynamically created tree is based, is colored black, descendent entities are colored blue, the ancestor entities are red, subtyped entities are cyan. As each entity is highlighted in the data model tree, all the attributes and keys are presented in “surrounding” list windows.

The SQL data definition file contains two different types of data models: Subject based and highlighted entity based. Subject based data models contain all related entities for which the highlighted subject is the parent. Highlighted entity based data models are the collection of entities that are related to the specific highlighted entity regardless of parent subject area. In the highlighted entity based data model, all direct ancestors and all direct descendants are produced. No “cousins” are produced.

The SQL data definition language file is able to be shipped to a SQL DBMS engine and compiled. From within the Specified data model all data types are Char(1). That is because the specified data model is purely logical and not cluttered up with database schema or DBMS data type details.

## 5.7 Specified Data Model Summary

The specified data model is a technology independent representation of entities and attributes within certain subject areas. The specified data model serves as a source of data model templates for both DBMS databases and for non-DBMS data structure definition facilities such as spreadsheets.

The specified data model attributes are quickly created through the identification of one or more data elements that are to be the semantic templates for attributes within entities. The inherited semantics from the hierarchies of meta category value types, meta category values, data element domains, data elements, and entity subject areas hierarchies all contribute critical components to the semantics of an attribute. Attribute definitions can be completely automatic with all the definition fragments that are immediately available from the attribute’s inherited semantics.

In the case where a data element is associated with more than one attribute in an entity as would be the case for home-, office\_, cell\_, and fax\_telephone\_number, the ability to create a local name and even a local definition is present. Available, of course are all the already inherited semantics.

Because of these work saving assists, creating a specified data model is an effort that is characterized by:

- Significantly shorter times to create entities because of all the inherited semantics



- Lowered risk because when things are the same they will be semantically defined the same
- Increased quality because there will be more time for important semantic component parts
- Increased productivity because the process of creating the specified data model can be accomplished by functional experts rather than just data administration staff.



## 6.0 Implemented Data Model

The implemented data model is the technology dependent transformation of a collection of entities from within the specified data model. The implemented data model may also represent a stand alone database design. That, of course, is not recommended.

While the Whitemarsh data modeler can read ANSI SQL data definition language streams and load the appropriate meta entities within the implemented data model area of the metabase, it is not advised. This is because there is really no ANSI SQL DBMS. Rather there are vendor DBMSs such as Oracle, DB/2, or Informix that conform in some way to the ANSI SQL language. Thus, all databases are operational databases. Consequently, these SQL DDL streams should be imported into the Whitemarsh operational data model, and are interrelated through DBMS column to Table column mappings.

The only situation where it is recommended to import SQL DDL streams into the implemented data model is when an enterprise has developed a library of DBMS independent data models that it then employs as “templates” for actual DBMS-bound database designs. If the data model design is really a production database, then it is advised to load the data model into the operational data model and then promote it to be an implemented data model. Thereafter the data model can be transformed as may be needed to fully name all the columns, allocate the semantics, relate the columns to data elements, and if necessary transform any non-third-normal form tables to third normal form. All the while the relationships to the operational data model will be automatically maintained.

A key difference between the specified data model and the implemented data model is that while the specified data model is to portray standard models or structures of data within the domain of a subject areas, implemented data models are to reside under the scope of a schema. In the context of the Whitemarsh data modeler, the concept–database–is linguistically represented through a schema expressed in the ANSI SQL language. Consequently, if the schema’s scope is broad, it is very likely to contain specified data model data structures from multiple subject areas.

Specified data models are of no particular data architecture class. In contrast, implemented data models are one of the five distinct classes, which are:

- Original data capture
- Transaction data staging area
- Subject area (Bill Inmon’s ODS (operational data store))
- Wholesale and retail (also called data marts) databases
- Reference data



Another key difference is that<sup>7</sup> tables within the implemented data model are able to belong to database objects, while, in the specified data model, entities are merely standard data structures that are able to be deployed as tables (possibly within database objects) of a particular database.

COTS, that is, commercial off the shelf packages like Oracle Finance or SAP's HR, generally contain their own self-contained data model. Some enterprises do not assess whether a COTS data model is of high quality or is designed well except as the COTS data model must be understood in light of required extra functionality for reporting, queries, data extracts for TDSA, and subject area data architecture class databases<sup>8</sup>. Under this situation, there is a need to both import the COTS data model through its DBMS specific SQL DDL as an operational data model and to then map the operational data model to an implemented data model, which should already be mapped to specified data model structures. This triple mapping is necessary so that the COTS data model data can be understood by the rest of the enterprise.

If an enterprise only had one COTS package, then this triple mapping clearly represents extra work. Most enterprises, however, commonly have multiple COTS packages within the same domain running on different platforms with overlapping and conflicting semantics. Without the implemented data model to which the COTS data models should be mapped there is no practical way to sort out these conflicting semantics.

Similarly, legacy systems may be seen as a set of COTS packages that have been developed over a variety of technologies, platforms, access methods, and DBMSs. Each of these legacy system file structures should be recorded into the data modeler in a SQL DBMS language form so that these legacy system "databases" can be mapped to the appropriate implemented data models.

Since an SQL-DBMS based COTS data model is implemented through a command file of DBMS schema, DBMS table and DBMS column statements, the imputed data semantics contained in the implemented and specified data models can then be used to tailor SQL views to shield the "unique/non-standard characteristics" of the COTS data model. For example, the DBMS data model COTS column, Social Security Number, which is an identifier for an employee, might be called PID. In an SQL view the name could be changed to the enterprise's name, Employee Social Security Number.

The meta model design for the implemented data model is presented in Figure 28. This diagram contains the following meta entity groups:

- Meta Category Value Types and Meta Category Values (upper left)

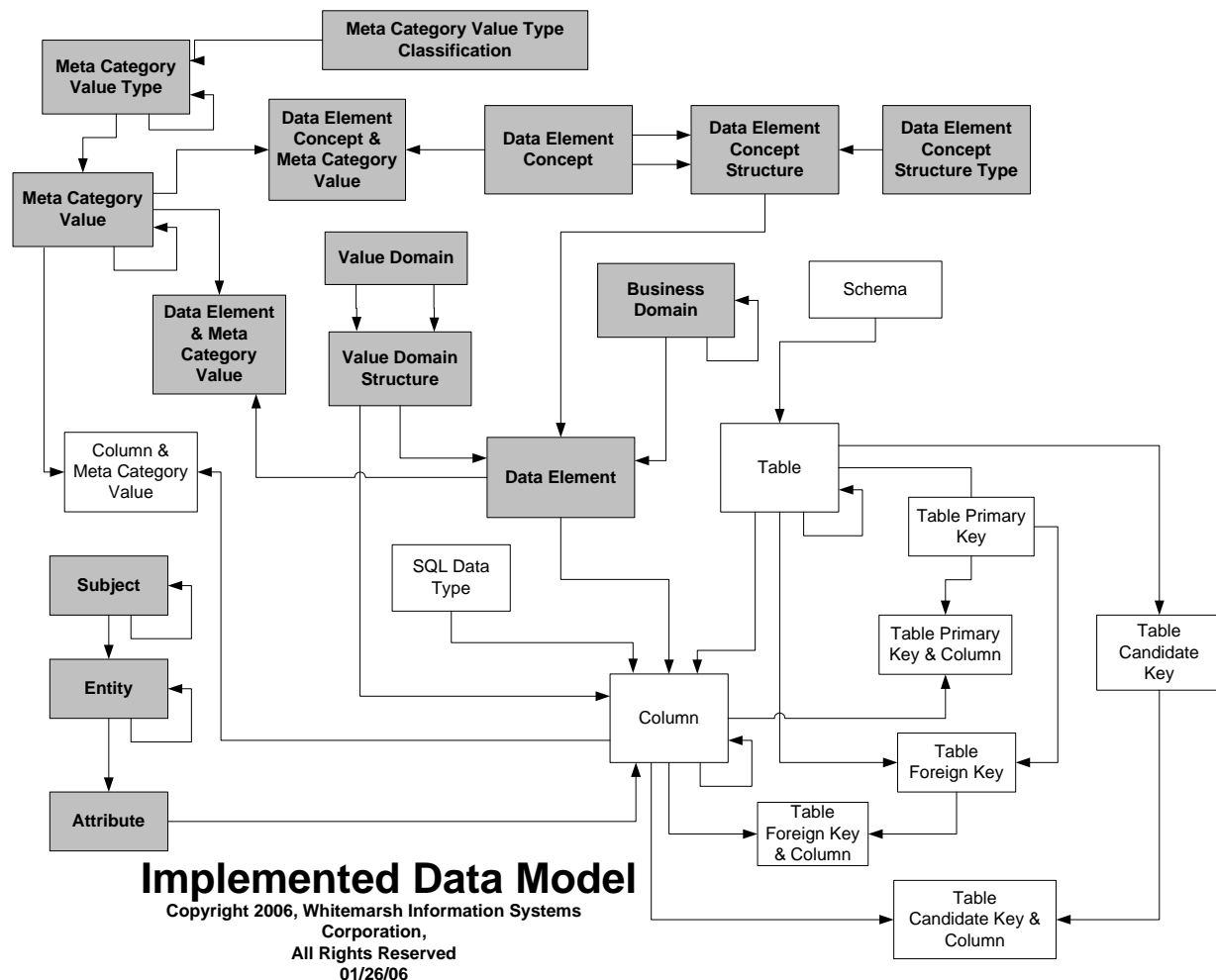
---

<sup>7</sup> Database objects represent the collection of traditional (that is, formatted and structured data) and nontraditional (that is, video, sound, and unstructured text) data. Database objects, expressed entirely within SQL:1999 proceed through precisely defined states and are squarely based on policy analysis for its data structure formulation, and on implemented processes that are embedded within the database object and that accomplish the database object's proper valuation, modification, migration, and reporting.

<sup>8</sup> It is the position of Whitemarsh, however, that no COTS package should every be procured unless it has a high quality, fully documented data model and unless this data model was used as the basis for the code generation of the majority of the COTS package.







**Figure 28.** Implemented data model meta mode.

- Data Element Concepts and Value Domains, Data Elements and Value Domains (middle and right)
- Subject, Entity, and Attribute (lower left)
- Schema Table and column including SQL data type (lower right)
- Primary and foreign key structures (lower far right)

The first three meta entity groups are shaded on the diagram. This means that they should already be valued in support of the data to be represented in the implemented data model. In the case there is a need to map to a data element there are two options: The process of automatically creating a data element from a column, and the explicit definition of a new data element but only through the data element module.



The data element module may be under special security to prevent the ad hoc definition of semantics. In such a case, the columns can still be added, but when they are without the appropriate mapping, reports containing these unmapped columns indicate that they are unmapped to standard data semantics.

Shaded meta entities are not allowed to be updated within the scope of creating the implemented data model. Rather they are accomplished in either the meta category value hierarchy classes, data element, or specified data model modules. The reason these functions are segregated is to prevent ad hoc data modeling behavior by allowing separate security over each distinct data modeler module.

As stated in the section on the Specified Data Model and in the section above, the purpose and intent of a schema along with its attendant tables and columns is to represent a database that is to be implemented on some technology dependent platform. That is, through one or more DBMSs and one or more specific computers. If a human resources schema is implemented under a SQL DBMS on three different platforms, for example, MVS, Unix, and Windows/NT, then while there would be only one Schema (and tables and columns), there would be three different sets of DBMS Schemas, DBMS Tables, and DBMS Columns.

Tables and columns within an implemented data model schema are created through:

- Forward engineering
- Original creation

## **6.1 Forward Engineering**

When the Schema and its attendant tables and columns is forward-engineered from one or more specified data models, the following steps occur:

- Schema creation
- Table and column creation
- Table maintenance
- Column maintenance
- Relationships maintenance (that is, primary and foreign keys)

### **6.1.1 Schema Creation**

The schema is created through the normal process of creating the metadata record, schema through a data modeler add screen. As can be seen from Figure 28, no direct relationship exists between schema and subject [area]. Clearly that means that the Implemented Data Model is not just a transformation of the Specified Data Model. The relationship is to the intersection record that connects the column of a table within a schema to the attribute of an entity within a subject area. Through these connectors, reports can be produced that identify which subject areas are represented in which schemas, and vice versa.



## 6.1.2 Table and Column Creation

The process of forwarding engineering tables for a schema is accomplished by importing entities. The entities are listed by subject area and are interrelated by primary and foreign keys. If an entity is selected for inclusion, a list of all entities related through primary and foreign keys is presented. When the apex of a set of related entities is tagged, all entities and sub-typed entities related to that entity are automatically included into the schema. This creates a series of table families.

When an entity is included in a schema, all the attributes of the entity are transformed to columns of a newly created table. In addition, to the creation of raw tables, the intersection records between a column of a table and the column of the related entity are also created.

The data modeler engineering is such that one or more entities can be imported into a table. Suppose, for example, a Specified Data Model had a very narrow scope and there was an entity that was just a person's name structure. That is, salutation, first name, middle name, last name, and name suffix.

If there was an implemented data model table for invoice header and there was to be a salesman contact, billing contact, and buyer contact, then all three would have the same person structure. This would require the importing of the Specified Data Model person's name structure three times, and then for each, the local names would change. The relationship from the specified data model to the implemented data model in this case is from the specified data model entity attribute set to the three different implemented data model table column set.

Similarly, a specified data model entity attribute set can be related to a single implemented data model table. In this case there might be an invoice header, but then a descendent table of invoice header key contacts. Where the first business column is invoice header key contact type to hold the value, salesman, billing, or buyer. And then a single set of columns for the person name structure.

Finally, there might be a specified data model entity that can be related to multiple implemented data model tables. First there might be the relationship between the specified data model person attribute set to the invoice header key contacts, and then to other tables within that implemented data model schema that might have person names such as product warranty representative, or region manager, and the like.

In all these cases, there is just one set of specified data model person name attributes that maps to all these different uses within the implemented data model tables. Clearly, not a transformation strategy. Rather a define-once use many-times strategy

In Figure 29, there are two subject areas, Project Management and Human Resources. In this example, the project management subject area contains three entities, project, role, and project assignment. In the Human resources subject area there is one entity, employee. Under the assumption that project, role, and employee are all connected with a mandatory action form of referential integrity, then if any of the three are identified then all three plus project assignment is brought over into the implemented data model. In this example, the entities, attributes, and relationships for project, role, project assignment, and employee would all be transformed into tables, columns, and relationships under the identified schema.



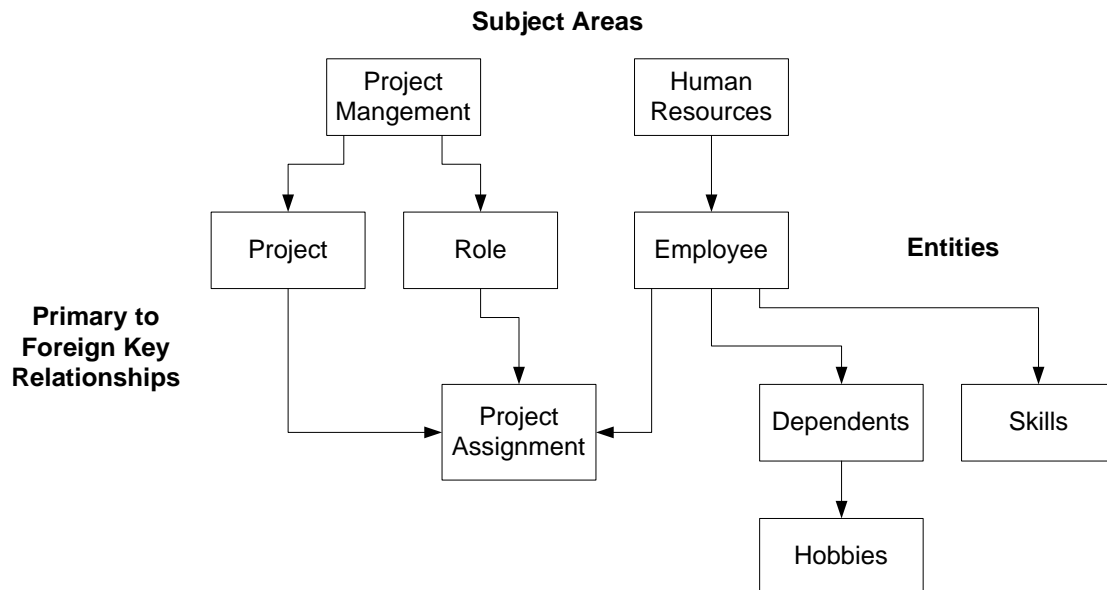
### 6.1.3 Table Maintenance

Once an initial set of tables and columns are created, tables can be “electronically pruned.” Three “pruning” cases occur:

- A table that is not a “parent” to another other table
- A table that is a “parent” to another other table
- A table that is a “parent” to a table that is in turn a parent to another table

Figure 29 illustrates these three types of tables. Skills is a table that is not a parent to another table. Dependents is an example of a table that is linked solely to the Hobbies table. Employee is an example of a table that contains dependents, which in turn, contains Hobbies.

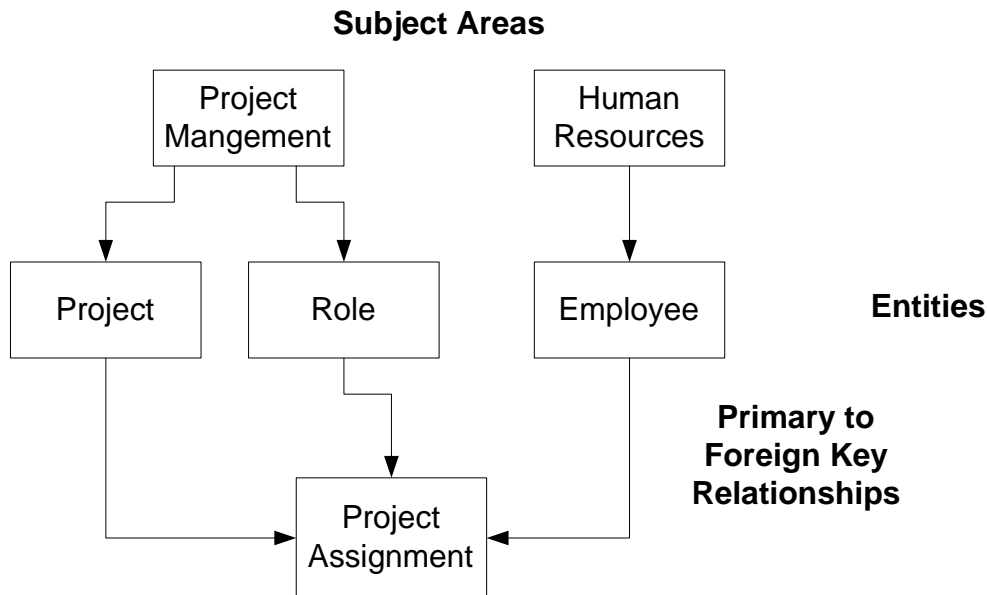
In the case of a table that is not a “parent” to another other table, for example, Skills in Figure 29, that table can be deleted in its entirety. The data modeler provides the ability to select a table, present it, and to delete the table through the delete button. If the table is not a parent of any other table, it is deleted. When this is done, the columns of that table and the intersection



**Figure 29.** “Pruning” cases for table maintenance

records to the appropriate attributes within an entity are deleted.





**Figure 30.** Entities that are transformed into tables of an implemented data model

In the case of a table that is a parent to one or more other tables, for example, Dependents in Figure 29, if the parent table is deleted so too are the foreign key and foreign key columns in the child table deleted.

In the case of a table that is a “parent” to a table that is in turn a “parent” to another table, for example, Employee which is the parent of Dependents, which in turn, is the parent of Hobbies, the process described above is cascaded to the “grandchild” table and evaluated before the action is taken on the “grandparent.” In this case, for example, if the “grandchild” table had a foreign key as part of its primary key, and if the “grandparent” was related to its “child” table through a cascade delete, then the delete operation on the “grandparent” would be disallowed because it would have caused a [cascade] delete on the “child,” which in turn, is prohibited because the “child’s” primary key, as represented in the “grandchild,” is part of the “grandchild’s” primary key.

When Dependents and Skills are thus deleted, the result is depicted in Figure 30.

## 6.1.4 Column Maintenance

Once an initial set of tables and columns are created, columns can be “electronically pruned.” Several cases occur:

- A column that is not part of a primary or a foreign key
- A column is part of a primary key
- A column that is part of a foreign key

Figure 31 illustrates the implemented metadata result from the transformation of the four entities, project, role, employee, and project assignment into tables within a schema, project management.

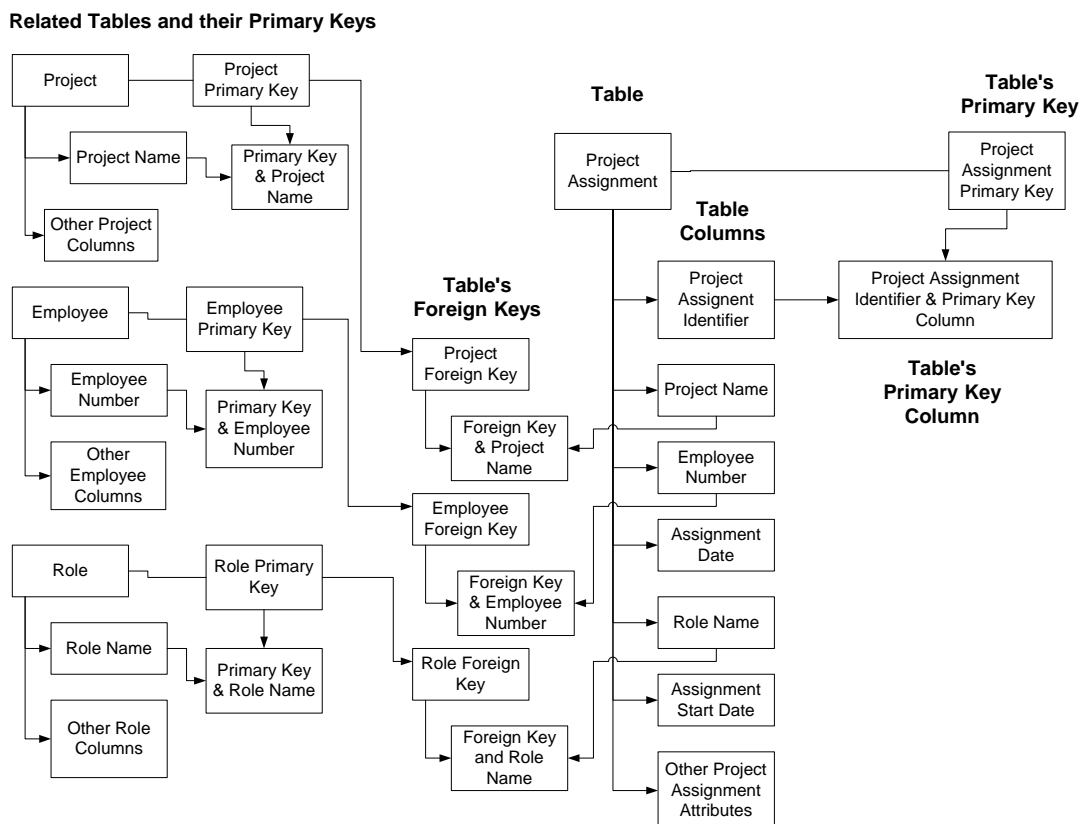


In Figure 31, Project Name (upper left) is a component of the primary key of project, while the “Other Project Columns” are not. Project Name is also part of the foreign key between Project and Project Assignment.

In this example, the column “Other Project Columns” could be safely deleted because “it” is not part of a primary or a foreign key, that column is deleted. Whenever a column is deleted, the intersection record that relates the deleted column to an attribute is also deleted.

Project Name, on the other hand is part of the primary key of project and thus, the delete operation is prohibited as it is likely to cause a fundamental change in the meaning of the table. In another example from Figure 31, if the primary key of Project Assignment consisted of Employee Number and Project Number, Assignment Date, and Role instead of just Project Assignment Identifier, then the delete operation which removed the Employee Number would then cause the fundamental nature of the table Project Assignment to be transformed from *one or more assignments of an employee to a project serving possibly different roles* to just that of role-based project assignments without regard to who was assigned or what role they performed. For that very reason, the Whitemarsh data modeler prevents the creation of a primary key that contains columns that are also members of a foreign key. In short, the columns of a primary key are orthogonal with the columns of any foreign key of that same table.

Continuing the case of a column that is part of a multi-column foreign key, deletion of

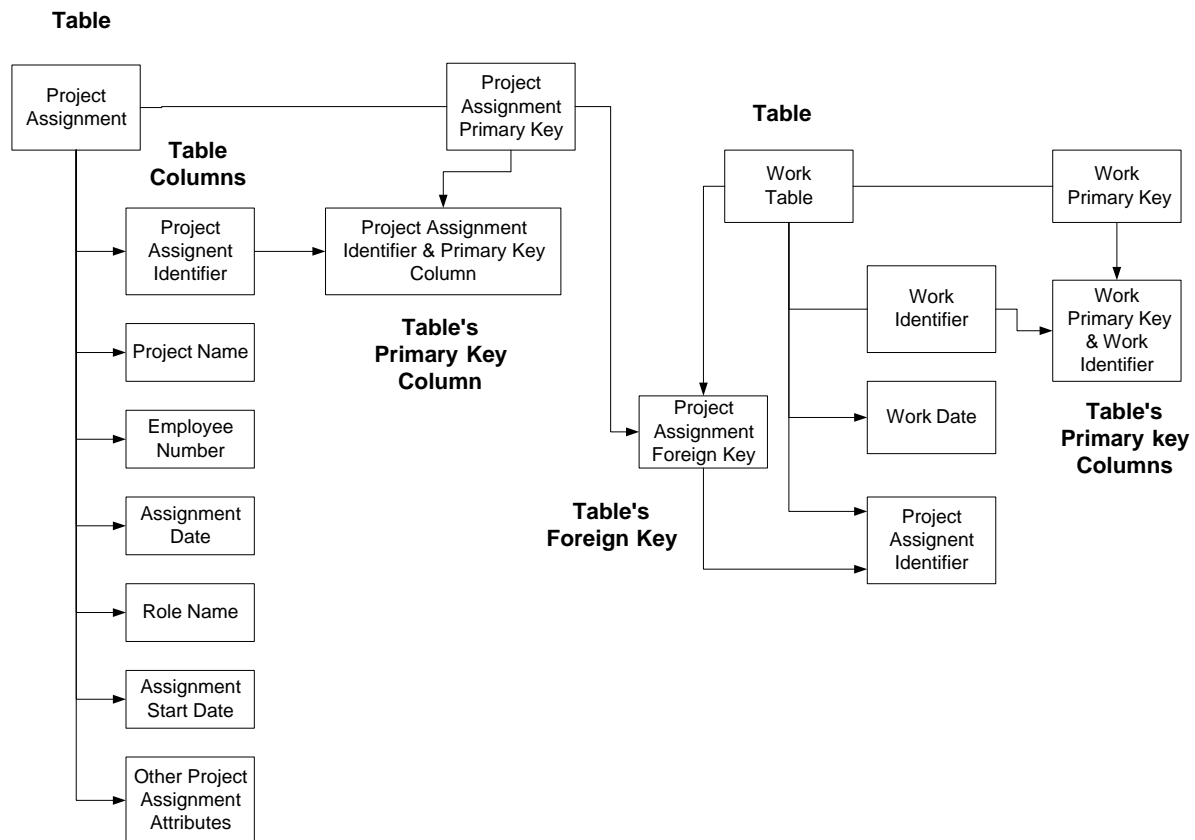


**Figure 31.** Stand alone, Primary, and Foreign Key Columns.



any one of the columns that comprise the foreign key is prohibited as that changes the very nature of the relationship between the table that is the source of the foreign key and the table containing the foreign key. The only operation that is allowed is the deletion of the foreign key. And, in that case, all the columns that are members of the foreign key are also deleted. Continuing with the example from Figure 31, if there was a table called Work, it would have to have a primary key of its own with columns solely resident in the Work table. It's relationship with Project Assignment would be based on the inclusion of the Project Assignment Identifier as the column of a foreign key in Work. Figure 32 shows the keys.

Once the appropriate set of columns are determined for each table, the other metadata attributes for each column must be created. This means selecting the appropriate SQL data type, the length, and quantity of decimal places. This metadata is necessary so that a report based on a schema can produce a ANSI SQL data definition language command file. Some ANSI SQL:1999 data types represent complex data structures such as user defined types, arrays, and nested structures. In these cases, the semantic mapping between an attribute of an entity to the columns of a table may only be approximate.



**Figure 32** Primary and Foreign Keys in Work.



## 6.1.5 Relationship Maintenance

Primary and foreign key support is automatically provided in most forward engineering cases. Whenever “pruning” a column potentially results in a change in the meaning of a relationship, that change is automatically prohibited. In these cases, the only way to accomplish this type of change is to perform an original creation. An example of what is not allowed is presented in the previous section and in Figures 32 and 33.

Relationships may also not exist when entities have been forward engineered into tables from multiple subject areas. This would be the case if the entities within a specified data model are restricted by convention to only single subject areas. For example, customers and sales persons may each be in their own subject area specified data model. Given the need to have a schema for sales and customer management, then entities from both subject areas would be forward engineered into tables belonging to one schema. In this case, there would be a need, for example, to create a table like customer salesperson assignment. This would be an intersection record and would require the creation of a primary key that contains two foreign keys, one to customer and another to salesperson. This type of table, column, and relationship creation is handled in the section that follows, Original Creation.

Figure 30 illustrates this situation as the entities from both the project management subject and human resources subject would not include Project Assignment if by user convention the specified data model was restricted to contain entities only from within the subject area. Note, however, the metabase system neither has nor enforces such a convention. It would only be after the implemented data model was constructed that the intersection table, Project Assignment would have been possible. And, in that case, columns from the both project and employee would be needed to construct the primary key for Project Assignment.

## 6.2 Original Creation

Original creation of a schema, attendant tables and columns involves these steps:

- Schema creation
- Table creation
- Column adjustments
- Relating columns within tables to attributes within entities of the specified data model
- Relationship creation (that is, primary and foreign keys)
- Column value domain creation





This “Relating columns ...”step is the very important because it enables enterprise wide semantics. Through this step, no matter in which table a column resides, its semantics, that is, those of the related entity attribute and related data element are able to be used to find that column.

### **6.2.1 Schema Creation**

The schema is created through the normal process of creating the metadata record, schema through a data modeler add screen.

### **6.2.2 Table Creation**

A table is created through the normal process of creating the metadata record, table. Since schema acts as a foreign key, the names for all schemas is provided for selection.

### **6.2.3 Column Creation**

The process of creating a column within a table is similar to that of creating an attribute within an entity. Once the table is selected, the task of creating columns starts. This is when the basic definition of data element comes into play. That is, as a semantic template for the business facts associated with the table. Column definition is accomplished in two stages:

- Employment of the data element semantic template along with the table to create the basic metadata for the column.
- Refinement of the column’s meta attributes through the creation of its remaining meta attribute values.

The first stage is accomplished through the Whitemarsh metabase technique of tagging. A three list window is displayed. In the upper left list is the list of tables. In the upper right window is the list of data elements. An table is tagged and then one or more data elements are tagged. Once



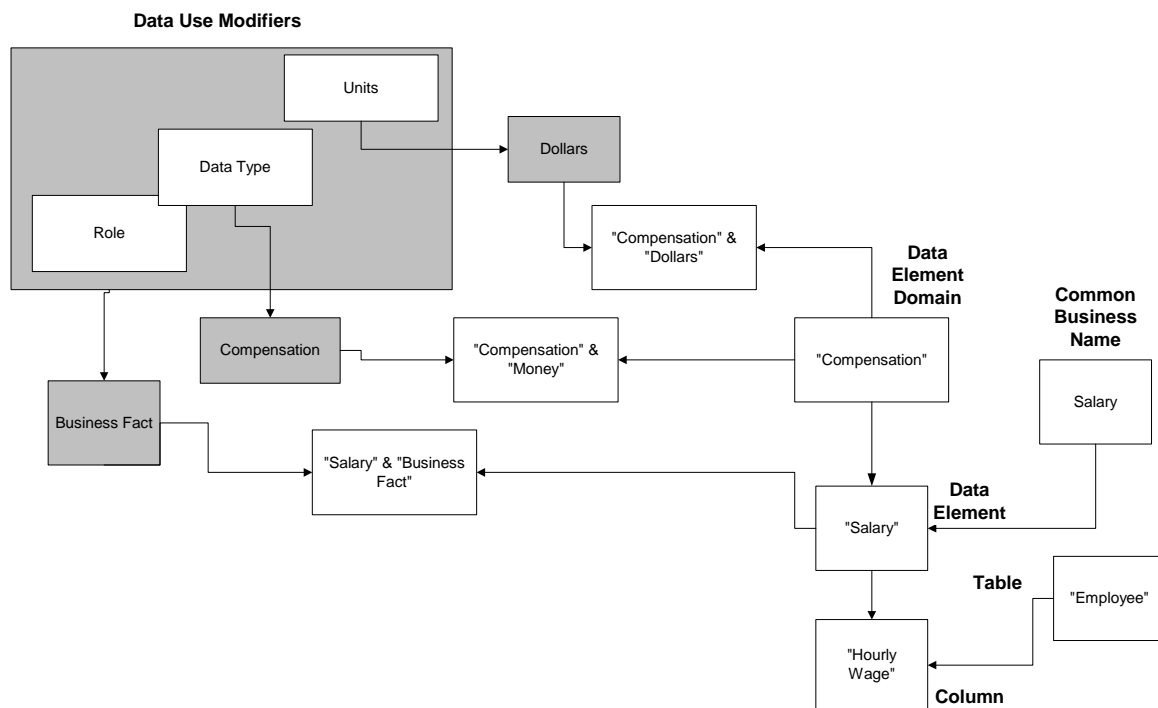
the “build” button is pressed the columns are built and displayed in the bottom list that runs across the full screen. That’s all there is to it. Because of inheritance, all the semantics of the data element are automatically assumed by the column. Figure 33 displays the result of associating the data element, Salary, with the table, Employee.

The semantics that are automatically inherited are:

- Data element domain and value restrictions
- Data element and value restrictions
- Data element’s business domain
- Data use meta category value types and values associated with the data element
- Schema of a table
- Table

The inherited semantics should be identical to those present on Figure 22. The column is related to the Salary data element. Missing from this figure, of course are the additional semantics resulting from completing the full meta data attribution of the newly created column.

The second stage involves selecting the appropriate SQL data type, length, quantity of decimal places, and the necessary semantic modifiers for that are directly tied to the column of the table rather than through the data element. Figure 34 presents the additional semantic modifiers that are suitable for the Hourly Wage column. In this example, the Geographic semantic modifier is the United States, and the Organization semantic modifier is Accounting.

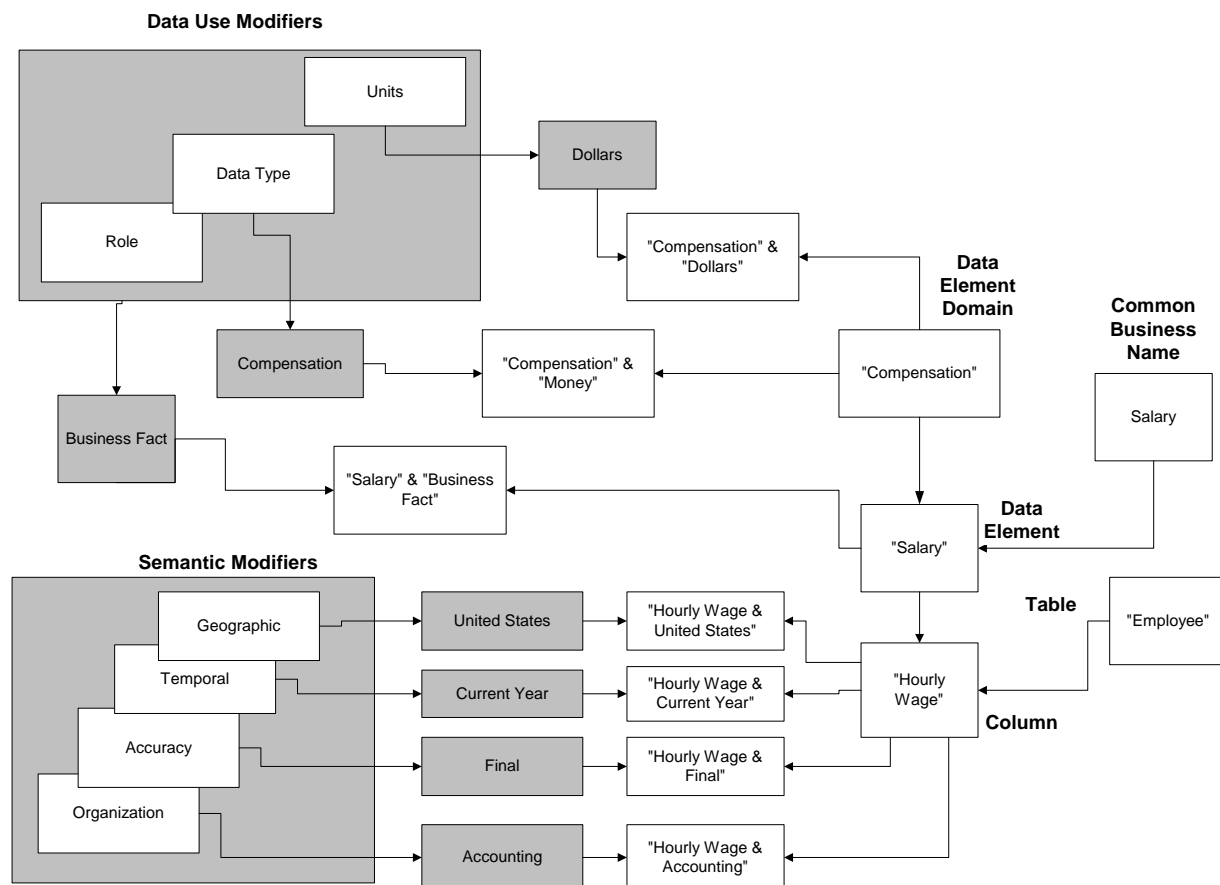


**Figure 33.** Associating a column to a data element.



Note that these are “semantically inside” the semantic modifiers of Figure 22. Still missing are any semantics that would be inherited when the Hourly Wage column is related to a suitable attribute.

In the case, however, when a data element is associated several times with an table, then while there may not be the need for additional or different data use or semantic modifiers there still needs to be some additional information to distinguish the different intent from what would appear to be “identical twin” columns. For example, suppose that the data element, telephone number is allocated to the Customer table two times as a way to represent the telephone number for the customer’s main and fax telephone numbers. In this situation, it may well be that only a slightly different local column name is necessary to represent the difference. So, data modeler provides the ability to change the local name for the column from its default name, that is, the data element name (which in turn has as its default name the common business name) to some other name. In this case, the local names would most likely be, main telephone number, and fax telephone number.



**Figure 34.** Additional semantics associated with the hourly wage column.



A key difference between a specified data model and a implemented data model can be the SQL:1999 capability to have “nested columns.” Prior to SQL:1999, all columns represented singly occurring data values all drawn from the same traditional data type. In SQL:1999, the very definition of data type has changed from its traditional meaning, e.g., Integer, Float, Decimal, Character, to a meaning that is closer to the computer science phrase, “data structure.” The five data structures (called data types in ANSI SQL:1999) are:

- Single value traditional
- Array
- User Defined type
- Row type
- REF

When the data structure of a column of a table is a single value traditional data type, then the data value represented by the column within a row of a table exists as a single value and is any of the traditional types, Integer, Character, Decimal, Float, etc. An example of a column with this data type would be Social Security Number, which of course is a fixed length numeric string.

When the data structure of a column of a table is an array, the “cells” are called elements, and the data value represented by the column within a row of a table exists as an ordered list of values all belonging to the same data type. An example would be Telephone Numbers, which could exist multiple times and all be of the same data type, that is, a fixed length numeric string. Because the list is ordered, semantics could be imposed that the first telephone number is the most common, and so on. The data type of an element within an array may be a traditional type (e.g., integer, date, character or decimal) or any of the other data types, that is, user defined type, row type, or a REF type. Each of these other data types are described in paragraphs that follow.

When the data structure of a column a user defined type (UDT), its contents are called attributes. A UDT attribute can be any SQL data type, including user defined types. Each attribute can have a domain name, a default clause, and a collate clause. In addition to data structure, the UDT contains functions through which UDT values are constructed, changed, or observed. UDTs can have instance ordering clauses. When ever a UDT is a subtype of another the parent UDT is identified.

The row type data structure is a technique to type a group of columns within a row. Each row type group is said to contain fields, and each field can be any data type. That is, it can be predefined type, a row type, a user defined type, etc. Each field can have a domain name and a collate clause. So, a table, Employee can have a row type, address as one of its columns. Within that address group can be another row type, zip code. Zip code can in turn have two fields, Zip and Zip +4.

The REF data structure represents a system generated “pointer” that references an object that is stored at another site.

The implication of these new SQL:1999 data [structure] types is profound. It has changed the SQL:1999 data model clearly from relational to its own data model. Within a table of an SQL:1999 database, full hierarchical records such as those within IBM’s IMS, or System 2000 can be defined. So too can “files” from Adabas, Model 204, and Inquire. Finally, because of the REF type, which can be the data type of an element within an array, Codasyl sets can be defined.



Because of these very sophisticated data structuring capabilities, database designers have to return to days in which there was a significant difference between the three types of data models, that is, specified, implemented, and operational.

The terminology within the new data types of the ANSI SQL:1999 standard can be confusing. For example, columns of a table, elements of an array, fields of a row type, and attributes of a UDT, and “data structure” is no where to be found. Not only was no alternative found, the mixed terminology was not really even noticed until late in the ANSI SQL:1999 ratification process. So, the exhortation, “get used to it” was put forward as the solution.

## **6.2.4 Mapping Columns of Tables to Attributes of Entities**

The final step of column creation is the mapping back to the attribute of an entity. This is done through the process of tagging. In this case, since an attribute may be related to multiple columns across multiple tables, the upper-left window of the intersection record screen has the subject area, entity, and attribute as the information in the upper left intersection record screen. In the right window, the data that appears is the schema, table and column. The process consists of going through the attributes of the entities until the proper one is found. That is then tagged. Then, the schema, tables and columns are traversed until the appropriate column is found. It is then tagged. If there are any more columns to be related to the attribute then these are tagged as well. Once all columns are tagged for the attribute, then the Build button is pressed. The intersection records are then built.

If however, no attribute seems to be appropriate, then one must be created to support enterprise-wide semantics. The data administration staff associated with attribute creation should be contracted to determine if there is an appropriate attribute. If the data administration staff cannot discover an attribute then an effort to create one must be undertaken. In this situation, two cases can occur:

- An attribute exists but is in a different form
- No attribute exists

In the first case, a column, for example, Employee Age at Hire is needed. If attributes with entities are properly created, then the closest attribute that could be used would be Employee Birthdate. If the actual data value for “Employee Age at Hire” is needed, then the column’s value would have to be the result of calculating the interval between the two dates. To compute the age value, a different column may have to be created to represent the event against which the employee’s age is computed. For example, if the Employee’s Age at Hire column is to represent the age when hired, then the column that would be required would be Employee Date of Hire. This column is likely to already exist in support of other human resources data. At that point, given there was an employee biographic table with the Employee Birthdate, the Employee Age at Hire could be computed on-the-fly by “subtracting” employee birth date from employee hire date and converting the interval result to years.

In the second case, that is, when no attribute exists, two subcases occur:



- The policy analysis under which the specified data model was created was insufficient, or
- Enterprise-wide data semantics is being created inductively.

In the first subcase, the methodology through which the specified data model was created must be examined to determine if it is sufficiently comprehensive. Projects and/or methodologies that are narrowly focused or that are function driven seldom result in specified data models that are appropriately generalized. The remedy is to base specified data models on the database domains within enterprise missions.

The second subcase arises when the enterprise is arriving at its semantics inductively. That is, individual databases are created to serve the needs of specific applications or business functions. In either case, the data model is likely to have application or function based names for both the tables and columns. Once these are created, it would be a mistake to just create clones of these schemas, tables and columns and make them subject areas, entities, and attributes. Analysis should be undertaken with “adjacent” organizations to determine if they have similar applications and functions, or if they share data. In either case, a set of names and definitions for these schemas, tables and columns should be generalized if at all possible. This process of semantic generalization then results in a wider-space of common semantics. Eventually, through this approach, enterprise-wide semantics is achieved.

Once an attribute has either been found or created, it still may be the situation that the column represents a semantic subset of an attribute. In this case, the previously allocated semantic modifiers meta category value types and meta category values should reflect the extra restrictions. With respect to the Employee column, Hourly Wage, Figure 34 illustrates that the data element contains a more restricted set of semantic modifiers when compared to the attribute, Hourly Wage depicted in Figure 22.

In the event an attribute is immediately found, then no additional semantic modifiers are required. The only requirement is to create the intersection record between the column of the table and the attribute of the entity. In the event that the attribute is connected to a data element different from the one connected to a column, an error message is presented.

From the point of view of this single column, Hourly Wage, Figure 34 and Figure 22 represent a significant amount of semantics that becomes available solely on the basis of inheritance from the other meta entities to which the column is associated. When all the definition fragments from all these inherited semantics are collected together into a loose definition, nothing more should be required. In short, the only thing that may have to be originally entered is the local name of the column. And, even in most cases, the common business name that is carried down may be sufficient.

The reason why columns of tables are related to attributes of entities is because a schema table can be an arbitrary collection that does not belong to a single subject area as would be inferred if tables were related to entities. This type of mixed subject area mapping is especially common in the data architecture class, data warehouse. In the case of data architecture classes, original data collection, TDSA, and in subject area databases (i.e., Bill Inmon’s ODS), single subject area mapping to entities is commonly maintained within tables of schemas.



Mixed mappings also supports the Whitemarsh data modeler concept that the implemented data model is not just a technology binding or transformation step on the way to a DBMS schema. That is, first specified data model, then implemented data model and finally operational data model. While such a binding and/or transformation process is possible if all the schema tables and columns are drawn from entities within a single subject area, such a mapping is just not practical given today's databases from different data architecture classes. The Whitemarsh data modeler was not created to support some theoretical concept. Rather, it was created to support and then possibly improve the real practice of data modeling that must be accomplished on a day to day basis.

Columns that comprise surrogate keys are a special mapping case. Surrogate keys are a technique employed by implemented and operational data model data-modeler [persons] to replace a collection of business-based columns with a "computer-generated" integer value. This integer value is then employed as the primary key of a table and also as a foreign key in other tables. The ultimate purpose of a primary-foreign surrogate key pair is to act as a relationship between two tables.

In summary, while it is clearly possible to create columns without relating them to attributes, such actions should be considered professional malpractice. The metabase supports the production of reports that identify all columns that are not related to attributes. If these type of relationships had existed prior to the "Y2K" crisis, it probably would not have been a "crisis." Rather, "Y2K" would have been just another software maintenance event. That is, only painful, risky, and costly rather than possibly the reason why some businesses fail.

Whenever there are multiple levels of meta category value type and meta category values assigned to a data element domain, then to a data element, then to an attribute, then to a column, each semantic set must be a subset or a narrower interpretation of the next higher set.

### **6.2.5 Key Support**

The implemented data model supports three types of keys.

- Primary,
- Foreign, and
- Candidate

There are two classes of primary keys. The collective value from each class must be unique, which when used to select a row of data from a table results in only one row of data. The two primary key classes are:

- Surrogate key column
- Business column set

The surrogate key column is a crafted column whose value has no business meaning whatsoever. A surrogate key is always a single column and takes on names like *Employee\_Id*, or *Invoice\_Id*,



or Customer\_Id. The form of the name for the column is <table-name> + <Id>. This type of primary key is only employed within implemented and operational databases.

The business column set is that set of business fact columns, which in combination produced a unique value across all rows for that particular table. The business fact column set is what must therefore be used to fully identify and construct the entity's primary key or for use in defining the foreign key that must map back to the primary key.

### 6.2.5.1 Primary Keys

As in the case of the specified data model, a primary key is a named set of business fact columns, which in combination produced a unique value across all rows for that particular table. For example, for an employee table, the data modeler [person] must examine the columns assigned to the table and ensure that it is third normal form by determining the set of naturally existing columns that comprise the named primary key. A candidate set of columns might be the employee's full name, that is, first, middle and last. But for a large enterprise that may not be enough. Added may have to be the employee's birth date. Maybe that's not enough. Probably the last added column would be the person's birth location which might be city, state, and country.

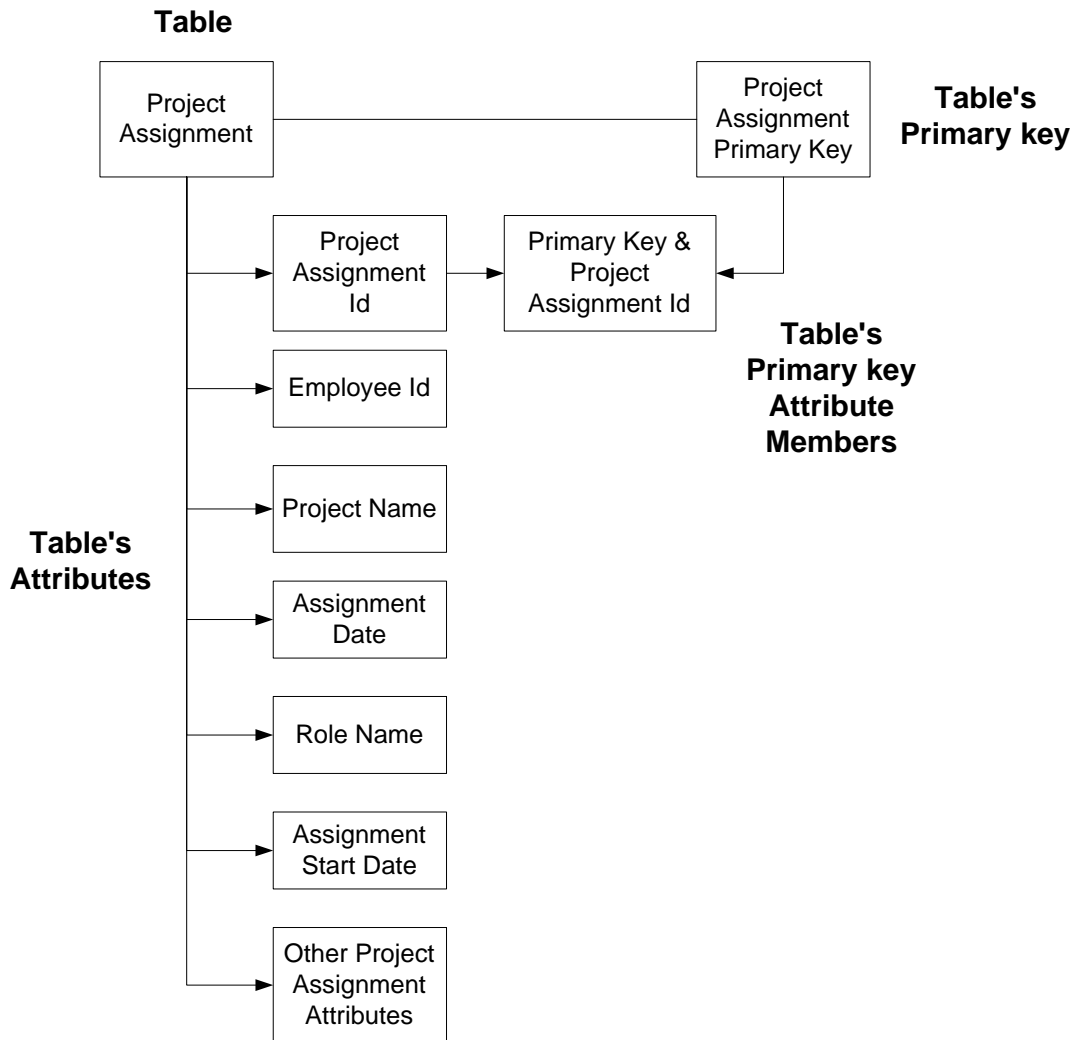
In the event a surrogate column is employed to represent the collective set of business facts, then the table that contains the surrogate key as a primary key must also have each of the business facts. In subsequent tables, for example, an Employee Dependent's table, the primary key may also be a surrogate key. The mechanism that would connect the Employee Dependents to the Employee would be the inclusion of the Employee Id surrogate key as a foreign key in the Employee Dependents table. While creating these types of keys is easy, this easy process must never replace the effort to derive a well engineered third-normal-form design for the table. Without the third-normal-form design step, it may never be known—until too late—that the database will contain redundant data that is updated unevenly.

As to the creation of the primary key, Figure 35 presents the table, columns, primary key, and primary key column rows of data that illustrate the creation of the primary key for the project assignment table in which all columns are surrogate keys. If the surrogate key is employed as the primary key of a table in the implemented data model, the mapping must be to the attributes of an entity in the specified data model that comprise the business data attributes for which the surrogate primary key is the alternative. If the surrogate key is a foreign key, however, there cannot be a mapping from the implemented data model to the attributes of the specified data model entity for which the foreign key occurs as a surrogate because these entity attributes are in the entity's "parent" entity. A data modeler report overcomes this problem by tracing the "lineage" of the surrogate keys until a full set of business data attributes is available for printing.

When surrogate columns are defined within tables, the assigned semantics should clearly show the column's properties as an identifier. Figure 36 depicts the meta category value type and meta category values that should be created so they can be allocated to the columns.







**Figure 35.** Project Assignment table with surrogate primary key

Since every column must be associated with a data element, a special data element such as “surrogate identifier” must be created so all surrogate columns can be associated with that special purpose data element. It is because of this special data element that the Whitemarsh data modeler can then “know” the key is a surrogate key in order to then perform a “lineage trace” to the ultimate set of business fact attributes for which the surrogate key is acting.



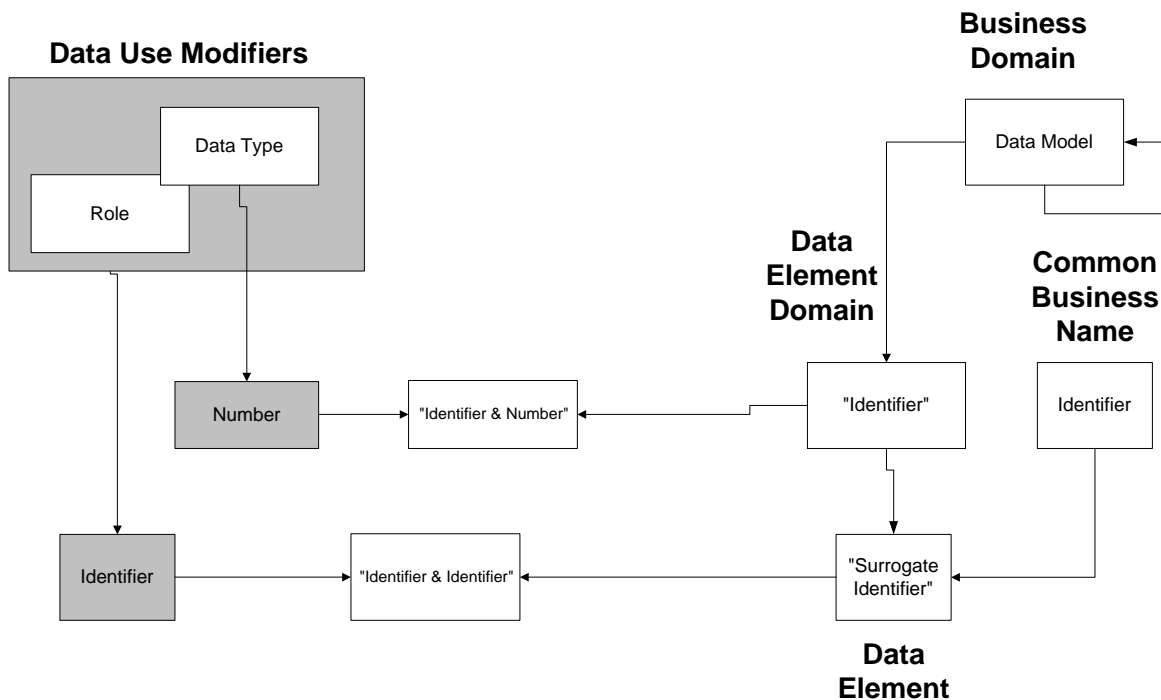
### 6.2.5.2 Foreign Keys

The foreign key is that set of business fact columns, which in combination produced a value which maps exactly to a primary key from another table. In the special case where the foreign key from a table maps to the primary key from the very same table, then the relationship is deemed to be recursive.

Figure 26 presents the specified data model that for the three entities, project, employee, and role that are “parents” of the table, project assignment. For the purposes of this example, when all these four entities are transformed to tables, all the primary and foreign keys are represented as surrogate keys. Figure 37 then presents the metadata records that are implied by the meta data model in Figure 25 that support the relationships among these entities.

In Figure 37, the tables, Project, Employee, and Role are along the left side. The columns necessary to represent the surrogate key based columns are depicted below them. The surrogate key based primary keys are depicted to the right of each table. The foreign keys for the Project Assignment table is presented to the left side of project assignment. There are three foreign keys, Project Foreign Key, Employee Foreign Key, and Role Foreign Key. The meta records for each is immediately below. The relationship between the primary key of each parent table is depicted. For example the relationship between the Project Primary Key and the Project Foreign Key.

From the presentation in Figures 26, 36, and 37 the required meta model relationships are:

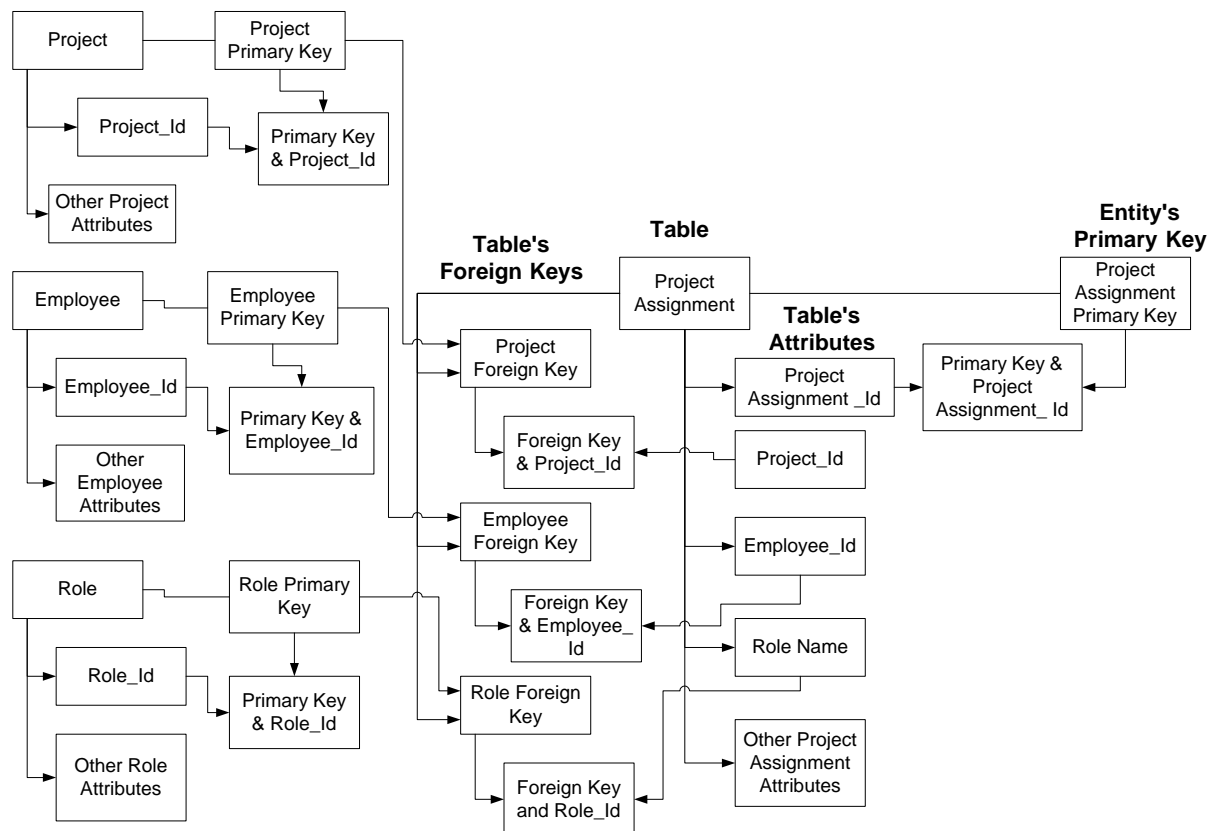


**Figure 36.** Meta category value type and meta category values allocated to Identifier data element.



- A one-to-one between a table and its primary key
- A one-to-many between any primary key of one table and any related foreign keys from one or more related tables
- An intersection record relationship between any primary key and its related columns
- An intersection record relationship between any foreign key and its related columns

**Related Tables and their Primary Keys**



**Figure 37.** Metadata required to support surrogate primary and foreign key relationships among Employee, Project, Project Assignment, and Role.



While this may seem like a lot of work, it is entirely accomplished through the use of tagging. In fact, for the accomplishment of the foreign keys, the Whitemarsh data modeler module merely requires the selection of the of the “parent table.” All the intersection records and columns required to represent the foreign key are automatically created in the “related-to” table. The default names for the foreign key columns are in the following format:

<parent table name> +<parent column name> + <foreign key>

In this case, the attribute names are thus:

- Project project name foreign key
- Employee employee number foreign key
- Role role name foreign key

Since the local name for a column is able to be changed without changing any of the semantics, the redundant words should be removed to improve readability. In the case where there are additional levels of data model, for example in the case where time charges would be recorded against a project assignment, the default attribute names for just these three attributes that would comprise the foreign key from Time Charges to Project Assignment would be:

- Project assignment project name foreign key
- Project assignment employee number foreign key
- Project assignment role name foreign key

In this case there would be no redundant words.

### **6.2.5.3 Referential Integrity**

Referential integrity is simply a statement of the rules that control the integrity of a reference that exists between the primary key from one table and its presence as a foreign key in one or more related tables. The purpose of referential integrity is control the consequential actions taken when an update occurs to either a “child” or a “parent” table.

The referential action rules are specified through controlled drop down lists when the foreign key is selected. Appendix 1 of this book presents the referential integrity rules that are supported by the metabase data modeler.

If two tables are the owner of another table, only one table can have the referential integrity action, cascade delete. If cascade delete is attempted for a second connected table, an error message will appear. Only after the cascade delete action is removed from an existing table can another cascade delete action be placed.

No foreign key field that is part of primary key field is allowed to be set-null as a referential action. While a column within a primary key may be allowed in certain DBMSs, it considered bad data modeling and is thus disallowed within the Whitemarsh data modeler.



Relationship stories, a standard report in support of the implemented data model employ the same “may be” and “must be” conventions used in the specified data model.

## **6.3 Reverse Engineering**

Within the context of the implemented data model, reverse engineering relates only to five areas:

- Reassigning of column to attribute
- Reassigning of column to data element
- Reassigning of column to SQL data type
- Reassigning of a column to a table
- Reassigning tables to schema
- Reassigning tables to tables
- Promote implemented data model to a specified data model
- Promote implemented data model table to specified data model
- Promotion of a column to a data element
- Remove column meta category values
- Remove column attribute assignments

In general, the implemented data model is created through the use of specified data model entity templates. However, the implemented data model may also be created through reverse engineering of SQL DDL streams into the operational data model and then data model promotions into the implemented data model. In such situations there are no specified data models.

When there is no specified data model, implemented data model columns are mapped to the single data element, unknown. The attribute, unknown, is also the parent of the columns. To properly resolve this “mapping to the unknown” situation, the implemented data model allows the reassignment of a tagged set of columns to data element. Similarly, a tagged set of columns can be allocated to one attribute. If the column is already allocated to a data element, the allocation process of the column to the attribute occurs only if the semantics of the attribute are compatible with the column’s already allocated data element.

During the iteration process of refining an implemented data model, data types may initially be vague. In such situations, modelers can subsequently tag one or more columns and reassign them to a different tagged SQL data type.

The last type of simple reverse engineering is the promotion of a discovered column as a data element. The promotion process causes a data element to be created from the essential semantics contained in the column. Once the data element is created, it is connected to the column, and all the semantics associated with the newly created data element are stripped from the column. These are still inherited by the column as the data element is its semantic template.

The largest type of reverse engineering is the promotion of a table and all its columns and keys (but not foreign key as it’s really the primary key of another table) to be an entity and attributes within a specified data model. Once promoted the subject of the entity needs to be



identified as it's initially set to "unknown." Attributes are set to the data element of the promoted column.

Table promotion is needed as it is likely that original data capture databases are initially created through an operational data model promotion process, which was in turn created through a SQL DDL import process from a commercial off the shelf package that was purchased by the enterprise. Once an implemented data model is promoted from an operational data model, the process of promotion should occur one more time to the specified data model to ultimately arrive a set of subjects, entities and attributes. It is these re-worked entities and attributes that will enable the cross referencing of implemented data models via their column-mapped attributes.

Then, once the specified data models exist, "down-stream" data architecture databases, such as TDSA, subject area databases, reference table databases, and the myriad of data warehouse databases can be created through already described forward engineering capabilities

## **6.4 Column Value Domains**

Irrelevant of the technique employed for implemented data model development, that is, forward engineering or original creation, column value domains are a critical component of semantics. The column value domains are similar to those of data element and attribute value domains. There are extensions and possible subsets.

In the case of the column, Employee Age at Hire, which is within the semantic scope of the Age data element might be between the values of 16 and 65 to conform to the corporation's employment rules of only hiring persons between those ages. In contrast, the data element, Age, could range from a value just greater than zero to one of no real limit as the data element could apply to rocks as well as to employees. Figure 38 illustrates the metadata that support the hierarchical set of value domains. As can be seen from this example, whenever there are multiple value domain "levels," each level must be a semantic subset of the previous (i.e., "higher") level.

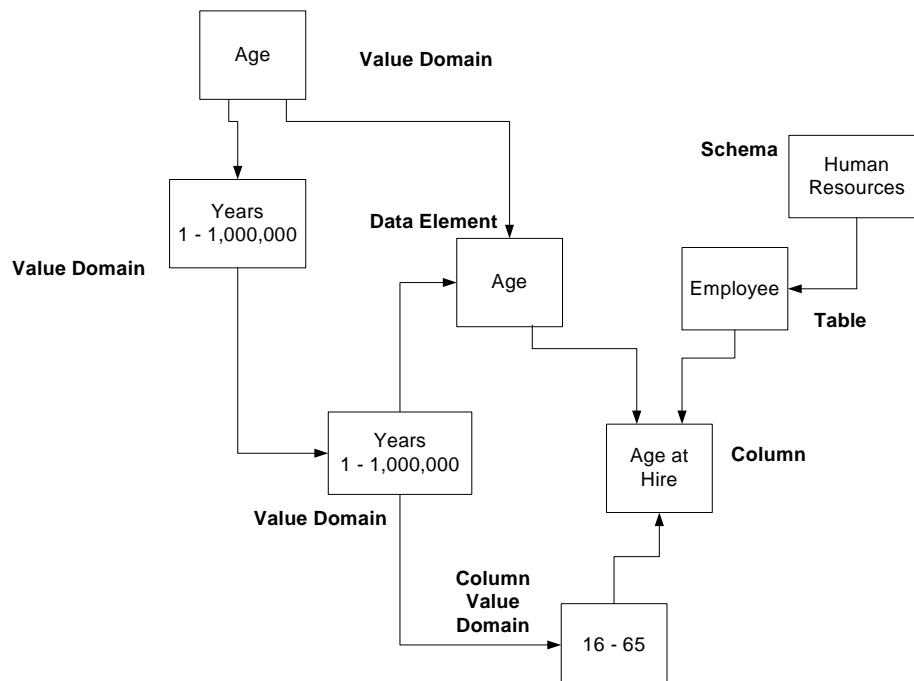
## **6.5 Implemented Data Model DDL and Graphics**

The data modeler, via the data model tree menu selection, supports a graphical tree-rendition of a data model. You can also cause the creation of an ASCII text file SQL data definition language if the selected specified data model.

The first graphic is presented immediately within the window as a graphical tree. In these graphic trees, the targeted root table upon which the dynamically created tree is based, is colored black, descendent entities are colored blue, the ancestor tables are red, and sub-typed tables are cyan. As each table is highlighted in the data model tree, all the columns and keys are presented in "surrounding" list windows.

The SQL data definition file contains two different types of data models: Schema based and highlighted table based. Schema based data models contain all related tables for which the highlighted schema is the parent. Highlighted table based data models are the collection of tables that are related to the specific highlighted table within the specific schema of the highlighted





**Figure 38.** Hierarchy of value domains

table. In the highlighted table based data model, all direct ancestors and all direct descendants are produced. No “cousins” are produced.

The SQL data definition language file is able to be shipped to a SQL DBMS engine and compiled.

## 6.6 Implemented Data Model Summary

The implemented data model is distinct from the specified data model. It may merely be the technology dependent transformation of a collection of entities from within the specified data model whereby subject areas, entities, and attributes serve as data structure templates for tables and columns within an implemented database. The implemented data model may also represent a stand alone database design that has not been mapped to the specified data model. That of course is not recommended. The implemented data model may have a multiple subject-area scope and thus may represent interrelated collections of entities, attributes and relationships from the different subject areas.

Specified data models are of no particular data architecture class. In contrast, implemented data models are one of the five distinct classes, which are:

- Original data capture



- Transaction data staging area
- Subject area (Bill Inmon's ODS (operational data store))
- Wholesale and retail (also called data marts) databases
- Reference data

Another key difference is that tables within the implemented data model are able to belong to database objects, while, in the specified data model, entities are merely standard data structures that are able to be deployed as tables (possibly within database objects) of a particular database.

A key value of the implemented data model is to convey a COTS package vendor's data model in a form that is understood by the rest of the enterprise. If TDSA data architecture databases are built with push-data from the COTS package, then downstream pull-database applications such as subject area databases and data warehouse databases can be built.

The implemented database's main purpose and intent of a schema along with its attendant tables and columns is to represent a database that is to be implemented on some technology dependent platform. That is, through one or more DBMSs and one or more specific computers. If a human resources schema is implemented under a SQL DBMS on three different platforms, for example, MVS, Unix, and Windows/NT, then while there would be only one Schema (and tables and columns), there would be three different sets of DBMS Schemas, DBMS Tables, and DBMS Columns.

Creating the implemented database is accomplished either through forward-engineering or original creation. Forward engineering is accomplished initially by identifying the relevant entities and pressing the build button. The underlying data modeler software creates all the tables, columns, and for those already related, the relationships in terms of primary and foreign keys. When a schema is to contain relationships among multiple unrelated tables, then the data modeler supports the creation of these new relationships. If there are more tables, columns, and relationships than necessary, data modeler supports "electronic pruning" of the aspects of the implemented data model that are not needed.

Original creation is accomplished by creating a schema, a set of tables, columns (through data element tagging), and appropriate relationships. The resultant columns should be related to existing attributes. If none exist then the data administration group should be enlisted to either discover existing attributes, or create other columns that support the production of the right values for the needed column. A byproduct of original creation is the inductive creation of additional data semantics within the enterprise.

The greatest benefit from this approach to the development of the implemented data model is the ability to employ predefined semantics from the specified data model. That is, from subject areas, entities, attributes, data elements, meta category value types, and meta category values. After a period of time, this approach should enable enterprises to develop entirely new schemas, tables, columns, etc., with a minimum of original effort. The data architecture classes that benefit most are subject area databases, wholesale and retail (data mart) databases, and TDSA databases. Original data capture and reference data databases benefit the least from this approach as these databases are the source of the majority of data structures and semantics for the enterprise.







The meta entity groups within this meta model are:

- Meta Category Value Types and Meta Category Values (upper left)
- Data Element Domains and Value Domains, Data Elements and Value Domains (middle)
- Schema, table and column including SQL data type (lower middle to left)
- Database, DBMS, DBMS schema, DBMS table, and DBMS column (middle right)
- DBMS Column primary and foreign key structures (lower right)

The first three groups should already be valued if the process of creating the operational data model is proceeding from specified to implemented and now operational data models. While it is possible to begin only with operational databases and to proceed inductively to enterprise database, the road will likely be longer and more difficult.

Most likely, organizations will begin the journey to enterprise database in a both top-down and bottom-up fashion. That is, top-down from missions to database domains to database objects and then to specified data models for one or more subordinate mission areas such as finance, human resources, inventory, sales, distribution, and customers. Then, as this effort proceeds, existing operational databases can be fed--bottom-up--through the reverse engineering process to create the realistic portrayals that must be present to give implemented databases a real rather than a theoretic existence.

As stated in the section on the implemented data model and in the section above, the purpose and intent of a DBMS schema along with its attendant DBMS tables and DBMS columns is to represent a database that is operational on some technology dependent platform. That is, through some DBMS and on a particular computer. If a human resources schema is implemented under a DBMSs on three platforms, for example, MVS, Unix, and Windows/NT, then there would be three different sets of DBMS Schemas, DBMS Tables, and DBMS Columns.

This three data model strategy is illustrated in Figure 40. In this figure, a specified data model is depicted with its four subject areas and a partial set of entities are shown. In this specified data model, none of the entities from the different subject areas interrelated. These entities are then stylistically combined into one implemented data model that supports sales and marketing. To achieve this, additional tables and relationships would have to be created. Finally, this single Sales and Marketing database is transformed into three different operational databases, one with all the data for the enterprise, one to serve the Canadian telecommunications network for the enterprise with only Canadian data and possibly data transformations to serve Canadian currency and metric measures, and finally, a whole series of very small databases that would be a downloaded at least once a week to every salesperson to support their weekly sales plan and activities.

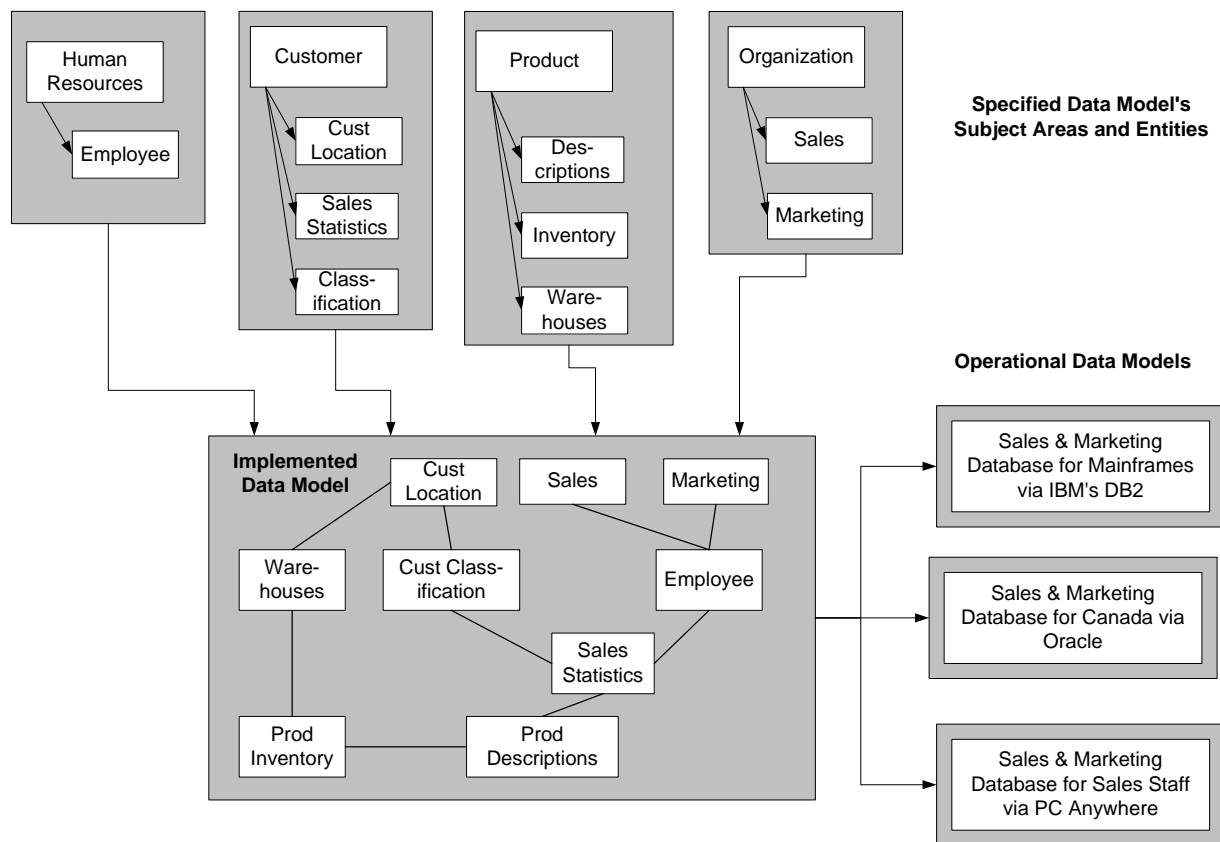
Supporting this three data model strategy would have to be operational original data capture data architecture databases that would provide the data for all four subject areas. These



original data capture databases might be on different platforms as well and operate under different COTS packages. Once the data is collected it would be pushed to TDSA databases and then pulled by possibly the mainframe version of the Sales and Marketing database application which might be a subject area database. Finally, the two operational sales and marketing databases, one for Canada and the others for each sales staff would likely be retail data warehouse databases (also known as data marts).

The operational data model DBMS tables and DBMS columns that belong to a DBMS schema can be created through:

- Forward engineering
- Original creation
- Reverse engineering



**Figure 40.** Relationship among Specified, Implemented and Operational Data Models



## 7.1 Forward Engineering

When the DBMS schema and its attendant DBMS tables and DBMS columns is forward-engineered from an implemented data model, the result represents one and only one operational data model, and is accomplished through the following steps:

- DBMS schema creation
- DBMS table creation
- DBMS column adjustments
- Relationships adjustments (that is, primary and foreign keys)

Prior to the start of the actual forward engineering, the database for which the DBMS schema is created and the DBMS must be identified. These are accomplished through normal creation and then selection screens.

### 7.1.1 DBMS Schema Creation

The DBMS schema is created through the normal process of creating the metadata record and DBMS schema through a data modeler add screen. Since the DBMS schema exists within the context of a database and a DBMS, both must exist prior to the development of the DBMS schema.

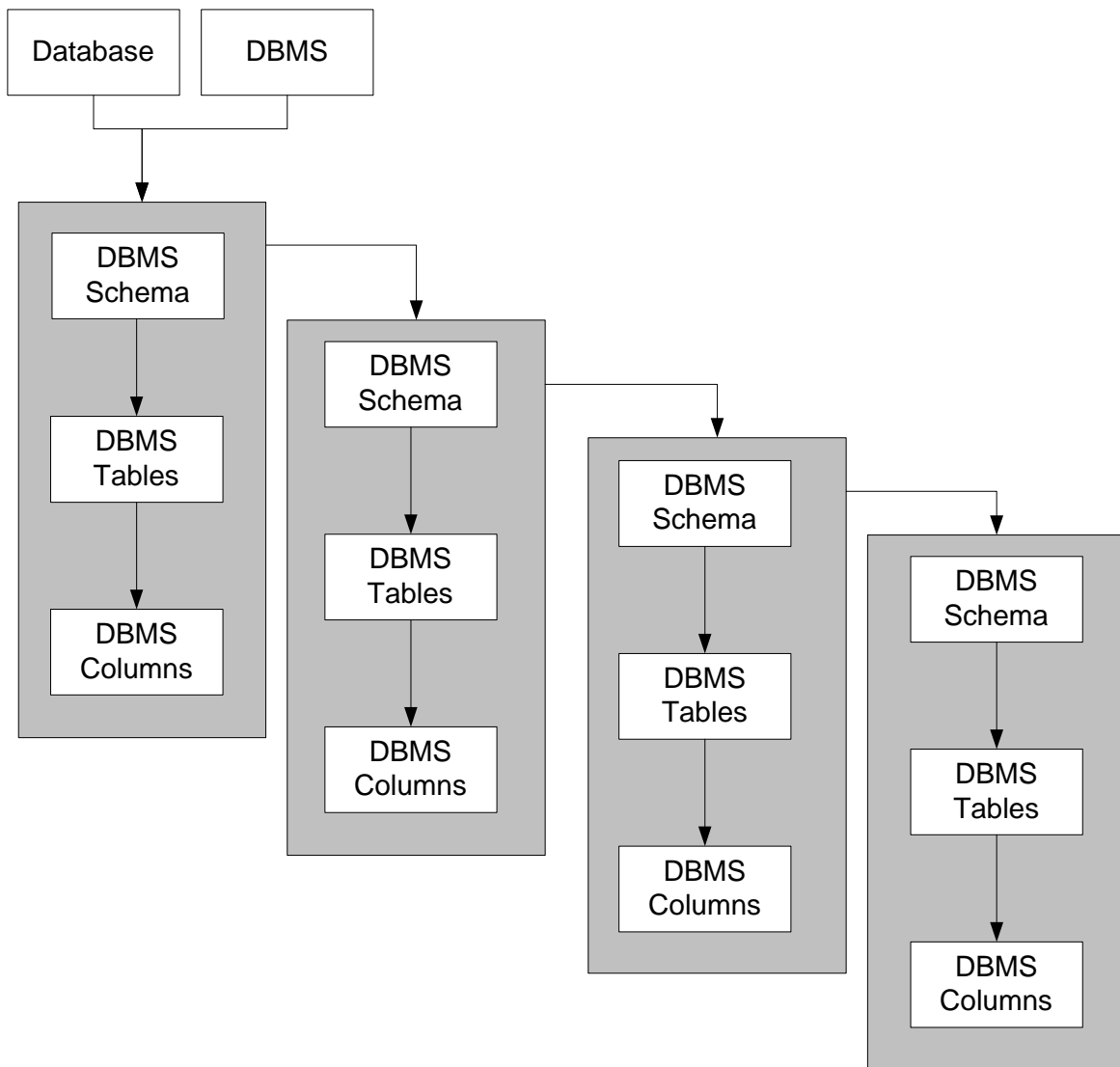
Database represents the overall domain of all the DBMS tables, columns, and relationships. As such there could be more than one different DBMS schema for a particular database. Since each set of DBMS schema, DBMS tables, DBMS columns and relationships are to represent the same database, the differences between the different sets could be attributed to editions or evolutions. To assist in tracking the evolution, the date of last update is visible along with a meta attribute for version. These are present in the meta entity, DBMS schema. A metabase process permits the bulk copying of the meta entity instances for metadata under the domain of a DBMS schema from which a new DBMS schema version can be created. This is illustrated in Figure 41. Reporting the changes from one version to the next is provided from the vantage point of the DBMS schema.

Also supported is the concept of having one single database, such as a human resources subject area database that is implemented through three different DBMSs because each DBMS schema (and associated DBMS tables, columns and relationships) are to be implemented on different computing platforms. This is illustrated in Figure 40.

### 7.1.2 DBMS Table Creation

The process of creating the DBMS tables consists of employing the add screen to create the DBMS tables for the DBMS schema. This is done by selecting tables from within a list of implemented data models. The tables within an implemented data model are listed by schema





**Figure 41.** Multiple versions for a particular operational data model DBMS schema and supporting metadata within the domain of a particular database and DBMS.

and are interrelated by primary and foreign keys. If a table is selected for inclusion, a list of all tables related through primary and foreign keys is presented. When the apex of a set of related tables is tagged, all tables related to that table are automatically included within the DBMS schema. This creates a series of DBMS table families.

As a DBMS table is included in a DBMS schema, all the DBMS columns are included automatically. In addition, the intersection records between a DBMS column of a DBMS table and the DBMS column of the related DBMS table are also created.

When an operational data model is created there may be several DBMS data types that have been chosen for the implemented data model that may not be available the DBMS that is to



“host” the operational database. In this case, the data type for the DBMS column is set to “unknown.” The report on the forward engineering process documents any data type transformations that are not possible. Some of the transformations will require the addition of different data structures within the operational data model. For example an ANSI SQL:1999 data type, array, may require an entirely new table within the operational data model to represent the multiple values from the array. A significantly greater sophistication would be required for other ANSI SQL:1999 data types such as row type, REF, and user defined types.

The ability to construct operational data models from implemented data models has the same flexibility as creating implemented data models from specified data models. That is, one or more implemented data model tables can be combined into any one operational data model table. Or any one implemented data model table can be brought into more than one operational data model table. Section 6.1.2 provides a good example of the flexibility.

### 7.1.3 DBMS Table Maintenance

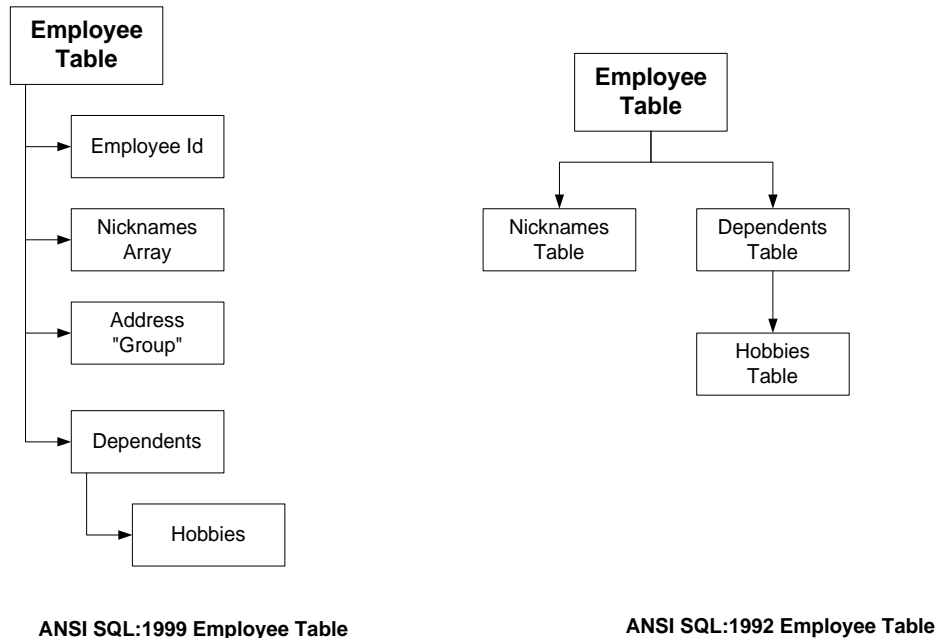
Once an initial set of DBMS tables and DBMS columns are created, through forward engineering, DBMS tables may result with some columns that have “unknown” data types. This means that the DBMS did not “understand” the data type for the column from the implemented data model. Analysis must be undertaken to understand the actual intent of the intended data type. Several cases can occur:

- A DBMS column needs to be transformed into a new DBMS table
- A DBMS column contains a substructure that needs to be transformed into multiple DBMS tables

In the case where a DBMS column needs to be transformed into a new DBMS table, as would be the case for the array data type, the new DBMS table must contain the primary key of the DBMS table that originally contained the array column. The primary key must also be supplemented with a sequence number as elements of an array are to be ordered by the sequence in which the values are entered rather than by an ordering based on the value of the array’s element. Finally, the new table should finally contain the column that represented the array’s element. Figure 42 illustrates the transformation process that represents the “before” and the “after” of this effort. In this example, the array column of Nicknames is on the left and the newly created table for Nicknames is on the right.

In the case where a DBMS column contains a user defined type that represents nested data structures such a hobbies within dependents of an employee, then two additional tables must be created: dependents and hobbies. The primary key for dependents must be the primary key from employee plus an appropriate additional column to identify the dependent. The primary key for hobbies must be the primary key for dependents plus additional columns to then uniquely identify the hobbies. To accomplish both these implemented to operational data model transformations, original creation activities (described below) must be accomplished. Once these original creation activities are complete, the “offending” columns that were marked as having





**Figure 42.** SQL:1999 DBMS Column transformations that may be required to accommodate SQL:1992 DBMSs.

“unknown” data types must be removed. This is accomplished within the column maintenance section that follows. Figure 42 also illustrates the transformation process that represents the “before” and the “after” of this effort. In this example, the dependents and hobbies within the dependents is on the left and the newly created table for Dependents and Hobbies is on the right.

### 7.1.4 DBMS Column Maintenance

Column maintenance within forward engineering should consist solely of one or more DBMS column deletions as either the column is an acceptable data type match, or it requires the original creation of new DBMS tables with columns to “flatten out” the SQL:1999 column based data structures.

### 7.1.5 Key Support

The keys supported in the operational data model are:

- Primary
- Foreign



- Candidate, and
- Secondary

Primary, foreign, and candidate keys are the same as in the implemented data model. Secondary keys are keys where the unique value is not required. These keys are intended to directly support the index capability of most DBMSs. There should be no maintenance of primary and foreign keys. The only work in this area should be in support of the creation of new primary and foreign keys.

## 7.2 Original Creation

The original creation of DBMS tables and DBMS columns may be required because of “unknown” data types resulting when an implemented data model is forward engineered into an operational data model involves these steps:

- DBMS table creation
- DBMS column adjustments
- Relating DBMS columns within DBMS tables to columns within tables of the implemented data model
- Relationship creation (that is, primary and foreign keys)

### 7.2.1 DBMS Table Creation

A DBMS table is created through the normal process of creating the metadata record, DBMS table. Since DBMS schema acts as a foreign key, the names for all DBMS schemas is provided for selection.

### 7.2.2 DBMS Column Creation

The process of creating a DBMS column within a DBMS table is similar to that of creating a column within an table. Once a table is identified, the task of creating DBMS columns starts. In the case of an implemented data model, this task is accomplished by tagging a data element and pressing a build button. At that point, the column is automatically created. In this particular case, this is not possible because every DBMS column first maps to an implemented data model table column, which in turn maps to a data element. Consequently, the column creation screen will present a set of tables and contained columns. The column that should be chosen should be the one that is the source of the transformation from the ANSI SQL:1999 data type that cannot be





mapped. This way, the semantics will be properly allocated. The data type of the new DBMS column will still have to be chosen.

For example, if the implemented data model table Employee contained an array data type column, Nicknames, then the newly created DBMS table is Employee Nicknames, the data type for the new DBMS column, Nickname, should also be character. The transformation process is illustrated in Figure 42.

Because of this strategy of mapping the newly created DBMS column to the appropriate and already existing table column, then the semantics that are automatically inherited are:

- Data element domain and value restrictions that are related to the column
- Data element and value restrictions that are related to the column
- Data element's business domain that are related to the column
- Data use meta category value types and values associated with the data element
- Schema of a table
- Table
- Column of the table
- Attribute of the column

All these DBMS column semantics are present solely because of the already defined semantic instances from other meta tables. There's been no keying of any additional data. In addition to all these inherited semantics, all the definition fragments associated with these meta table types are employed to provide a full definition of the DBMS column. The sum total of all the inherited semantics are those contain in both Figures 23, 35 and 37.

### **7.2.3 Key Support**

The keys supported in the operational data model are:

- Primary
- Foreign
- Candidate, and
- Secondary

Primary, foreign, and candidate keys are the same as in the implemented data model. Secondary keys are keys where the unique value is not required. These keys are intended to directly support the index capability of most DBMSs. There are two classes primary keys. The collective value from each class represents a unique value, which when used to select a row of data from a DBMS table results in only one row of data. The two primary key classes are:

- Surrogate key column
- Business column set



### **7.2.5.1 Primary Keys**

Since the primary keys are to be related to those of existing tables, such as the creation of the nicknames table for an already existing employee, the first choice for a primary key should either be a surrogate key or one that contains the DBMS columns from the “parent” DBMS table plus what every other new column is necessary to construct a proper primary key.

### **7.2.5.2 Foreign Keys**

Again, since the foreign keys are to be the primary keys from the “parent” table of the newly created table, the choice is simple and straight forward.

### **7.2.5.3 Referential Integrity**

In the case of data structures that are resulting from transformed columns, the only proper referential integrity actions should be cascade delete. That is because, if the column did not have to be transformed, then when the “parent” row was deleted, the contained column values would be deleted. Hence, the only logical referential integrity action is cascade delete.

Because of the cascade delete referential integrity action, the produced relationship stories will all be for the “must be” variety.

## **7.3 Reverse Engineering**

The purpose of reverse engineering within the operational data model of the Whitemarsh data modeler is to import a SQL DDL command file for a particular DBMS that was:

- Created manually
- Already being used by a DBMS,
- Generated by a CASE tool such as ERwin
- Generated by an application development environment such as Clarion for Windows.

If all the critical statements within the file are correctly processed the metadata appropriate for DBMS schema, DBMS tables, DBMS columns, and relationships is created. If the DDL file cannot be acceptably processed, then no metadata is created. Finally, a report is produced that indicates the action taken on the input file on a command line by command line basis.



The DBMS schema and its attendant DBMS tables and DBMS columns are created through the following:

- DBMS identification
- Database identification
- External file identification
- DBMS schema creation
- DBMS table creation
- DBMS column creation
- Relationship creation (that is, primary and foreign keys)
- Relating DBMS columns within DBMS tables to DBMS columns within entities

The first two steps are accomplished through normal add or select screens. The steps, DBMS schema creation through DBMS column creation, are accomplished automatically by the metabase operational data model reverse engineering software.

The last step is accomplished manually and is the most important step because it enables enterprise wide semantics. Through this step, no matter in which DBMS table a DBMS column resides, its semantics are mapped first to the implemented data models, then to specified data models, and finally through the implemented data model to the data elements. Supplementing all these mappings are the allocated meta category value type and meta category values and the multiple layers of value domains.

### **7.3.1 DBMS identification**

The particular DBMS that governs the processing of the SQL command file must be selected. Once selected, this activates the appropriate processing for the different lines within the command file. A report from the data modeler indicated which types of command file lines are able to be processed for each different DBMS.

### **7.3.2 Database Identification**

The name and description of the database for which the SQL DDL is to be analyzed and configured into the metadata appropriate for an operational data model is entered through a normal metabase update screen.

### **7.3.3 External File Identification**

A normal file name selection dialog permits the identification of the SQL DDL command file to be processed.



### 7.3.3 DBMS Schema Creation through Relationship Creation

Once the import button is pressed, the command lines from the external file are processed one by one. The command lines must be in proper order. That is, the schema line first, then a series of command line sets for table, column, and relationships. As each command line is processed, the appropriate metadata is created for DBMS schema, DBMS table, DBMS column, and DBMS relationships (that is, primary and foreign keys).

### 7.3.4 Mapping DBMS Columns of DBMS tables to Columns of Tables

The final step of operational data model DBMS column creation is the mapping back to the specified data model column of a table. This is done through the process of tagging. In this case, since a table column may be related to multiple DBMS columns across multiple DBMS tables, the upper-left window of the intersection record screen has the schema, table, and column as the information in the upper left intersection record screen. In the right window, the data that appears is the DBMS schema, DBMS table and DBMS column. The process consists of going through the specified data model table columns until the proper one is found. It is then tagged. Then, the DBMS schema, DBMS tables and DBMS columns are traversed until the appropriate DBMS column is found. It is then tagged. If there are any more DBMS columns to be related to the tagged specified data model table column then these DBMS columns are tagged as well. Once all DBMS columns are tagged for the specified data model table column, the Build button is pressed. Intersection records are then built.

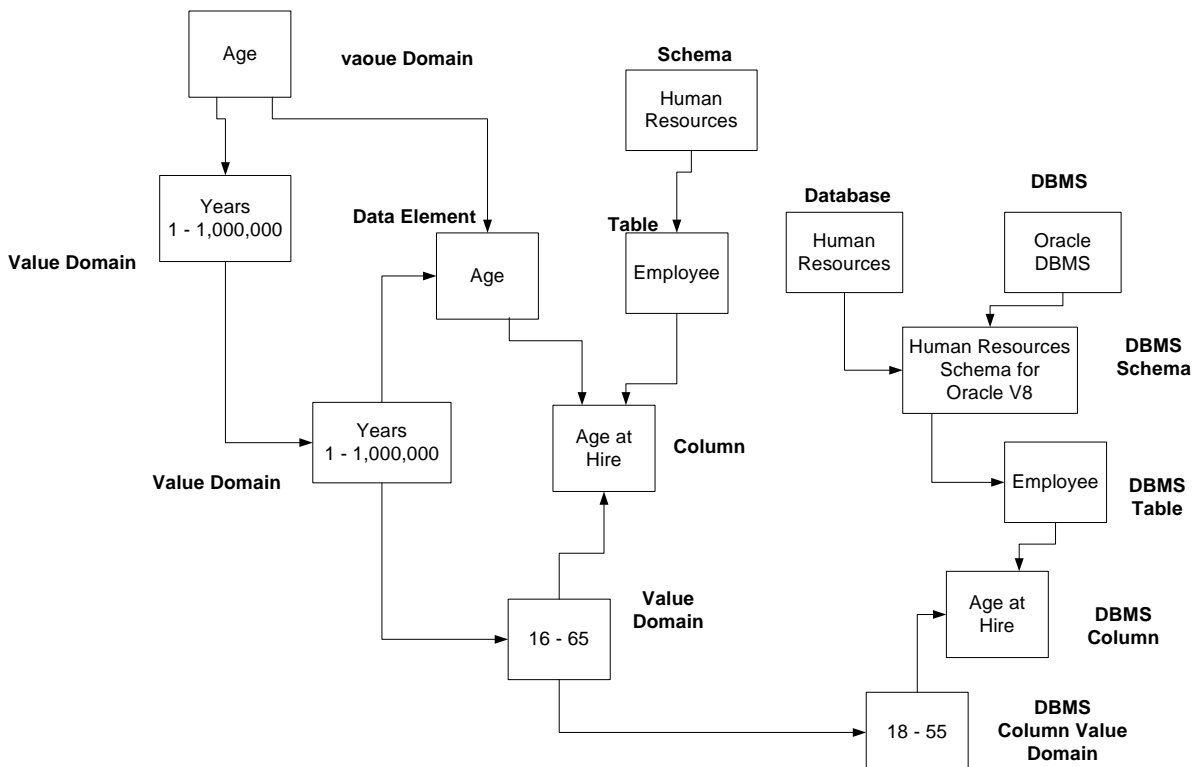
The reason why DBMS columns of DBMS tables are “manually” related to columns of tables is because the advanced data structure capabilities of SQL:1999 may not have been implemented in every DBMS that is to host a database. In these cases, this arbitrary mapping is necessary.

While it is clearly possible to create DBMS columns without relating them to specified data model table columns, such actions should be considered as professional malpractice. The metabase supports the production of reports that identify all DBMS columns that are not related to specified data model table columns. As stated previously, if these type of relationships had existed prior to the “Y2K” crisis, it probably would not have been a “crisis.” Rather, “Y2K” would have been just another software maintenance event.



## 7.4 DBMS Column Value Domains

The DBMS column value domains are similar to those of column value domains. They are extensions and possible subsets. In the case of the DBMS column, Hourly Salary, which is a specified data model table column within the semantic scope of the data element salary, which in turn is within the scope of the data element domain, compensation, the value domain for Hourly Salary must be within that of Salary. Figure 43 presents the next level of value domains, that is the value domain that would address Hour Wage.



**Figure 43.** Hierarchy of value domains from data element value domain through DBMS column value domain.



## 7.5 Operational Data Model DDL and Graphics

The data modeler, via the data model tree menu selection, supports a graphical tree-rendition of a data model. You can also cause the creation of an ASCII text file SQL data definition language if the selected specified data model.

The first graphic is presented immediately within the window as a graphical tree. In these graphic trees, the targeted root DBMS table upon which the dynamically created tree is based, is colored black, descendent entities are colored blue, the ancestor DBMS tables are red and sub-typed DBMS tables are cyan. As each DBMS table is highlighted in the data model tree, all the columns and keys are presented in “surrounding” list windows.

The SQL data definition file contains two different types of data models: DBMS schema based and highlighted DBMS table based. DBMS schema based data models contain all related DBMS tables for which the highlighted DBMS schema is the parent. Highlighted DBMS table based data models are the collection of DBMS tables that are related to the specific highlighted table within the specific DBMS schema of the highlighted DBMS table. In the highlighted DBMS table based data model, all direct ancestors and all direct descendants are produced. No “cousins” are produced.

The SQL data definition language file is able to be shipped to a SQL DBMS engine and compiled.

## 7.5 Operational Data Model Summary

As stated at the start of the operational data model section, the operational data model is not only DBMS specific, it is also targeted to a specific operating environment. If the only set of data models that an enterprise retains are those that reflect the operating DBMSs, then there will be no context independent business semantics from which the business can be understood and changed.

As clearly shown in Figure 40, there can be three operational data models, each with a somewhat different design due to DBMS characteristics and performance requirements for every database. Additionally the data models contained in the data structures within the specified data models could appear in multiple implemented and operational data models.

Because most enterprises do not define, interrelate and track the different deployments of the same set of business semantics they do not really have any control over the most valuable resource, information.

Operational data models are created through three avenues: forward engineering, original creation (due to capabilities that may be missing in certain DBMSs) and reverse engineering. Most often, the enterprises will employ a combination of top-down and bottom-up techniques. Top-down from techniques drive from missions to database domains to database objects and then to specified data models for one or more subordinate mission areas such as finance, human resources, inventory, sales, distribution, and customers.

Bottom up techniques proceed via reverse engineering to create the realistic portrayals necessary to give implemented databases a real rather than a theoretic existence. Once the top-down and bottom-up are joined, enterprise database begins to emerge.



What may seem to be an inordinate amount of work in order to achieve enterprise database is really not. The model building strategy of the Whitemarsh data modeler ensures that there is maximum reuse of already defined business based semantics through automatic copy and tagging techniques.

## **7.6 View Data Model**

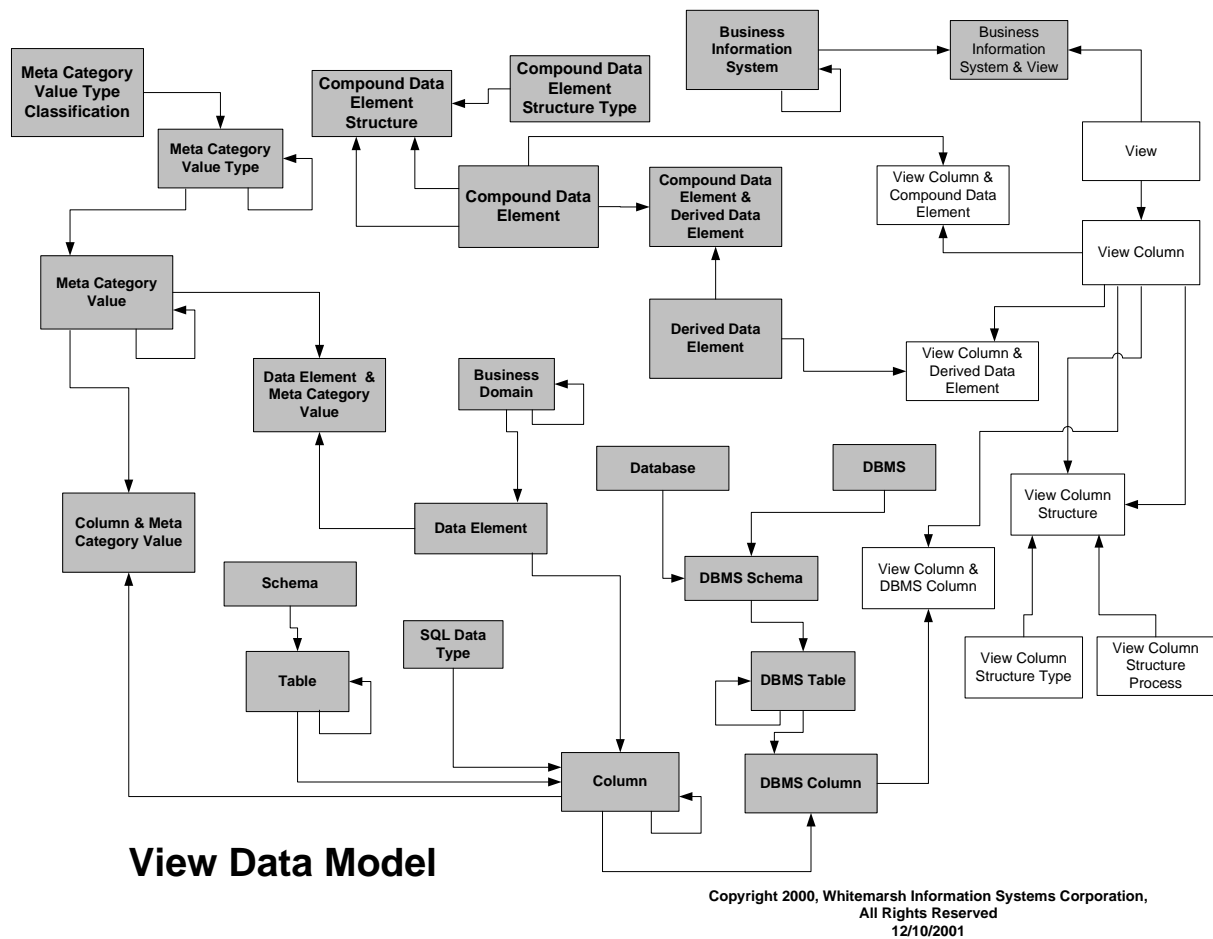
The view data model is where the “rubber meets the road.” If there were nothing but unimplemented data models there never would have been a Y2K problem. It was only because data models were created, implemented, and maintained in an unintegrated and non-enterprise-wide manner that the Y2K problem became the multi-hundred-billion dollar waste of corporate resources that it is.

It is not that data is specified wrong that is the “real” problem. It’s that it’s not known where the data is operating, under what systems, computers, and operating systems, and within which DBMSs or other file access methods that is the source of the real problem. If the data location was known in any efficient way then it could be fixed efficiently.

Finally, once the data is brought within the domain of a computer program it may be consigned to local program variables that transform and change it, and then possibly write the data back out to other computer-based files. While there is nothing within the Whitemarsh metabase data modeler that can “see inside” computer programs, the application view data model can track data use to the “door” of the application. That is, the SQL view that may exist between the operational database and the application system.

The meta model for the application view data model is provided in Figure 44. As can be seen from this figure, almost all the meta entities are shaded. The view and view column meta entities are not shaded and the data represented by them is created by loading ANSI SQL views. The other four meta entities that are not shaded are intersection records between view column and other shaded meta entities. These intersection record meta entities are created through normal tagging and intersection record build buttons. The view model also support relating one or more view columns of one view to one or more view columns of a different view.





**Figure 44.** View meta model data model.



## 8.0 Summary and Conclusions

The approach described in this book is simply this: create a re-usable cache of semantic rich data elements and then use that cache to support:

- The definition of database columns,
- The interrelationship of the defined columns through the data element metadata, and
- Both forward engineering of new databases and reverse engineering of existing databases.

The approach to creating a re-usable cache of data elements begins with the adoption of a data element definition that fosters reuse. That is, *a data element is a context independent business fact semantic template*. But how does it become that? It does when the data element represents an amalgamation of familiar concepts expressed as single words brought together under a single name and then represented as a value. The value is not the data element, the collection of semantics is. The value is merely a discrete value-based alternative representation of the semantics.

Critical to reusing the underlying data element concepts that form the semantics of a data element is a metadata model that exists within a repository type database. Finally, since all data elements are not just elementary atomic fact templates, both compound facts, and derived facts must be represented. Full name, and age are common examples.

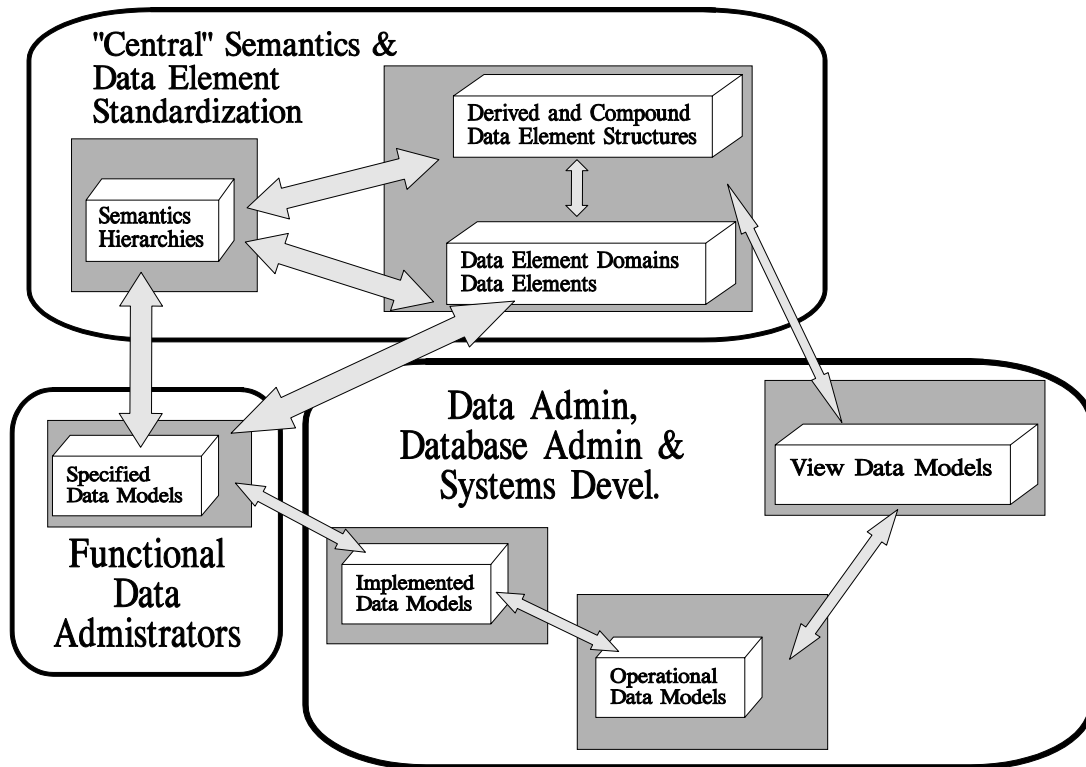
Putting all this together into a single data element meta model is depicted in Figure 14. While an enterprise could develop this data element meta model into a single data element metadata repository, it is neither critical nor practical. After all, if the ideal is reached, that is all the metadata is defined once, stored once and then used throughout the enterprise, and if that metadata repository suffers a catastrophic disaster then conceivably all IS development and maintenance would cease. While some might argue that humanity would be better off, it is not appropriate to either advance or refute such arguments.

If the enterprise supports a distributed metadata repository environment, then sophisticated distributed DBMS synchronization concepts would have to be employed. Shared for certain would be the certain of the meta models that are identified below, such as semantic hierarchies and data elements, and specified data models at least at the subject level.

Once implemented, if data element metadata resides solely within the confines of a metadata repository as opposed to being employed within a complete CASE (both lower and upper) environment, the benefits of the data element metadata will be greatly limited. In short, the more “active” the data element metadata is, the greater its positive effect on the goal of achieving enterprise-wide data standardization. Figure 45 presents an integrated environment in which data element metadata is integrated within a CASE-Repository environment. Within this environment there are the following submodels:

- Data Semantics





**Figure 45.** Integration of semantics and data element model with other essential to enterprise-wide data standardization models.

- Data Elements
- Specified Data Model
- Implemented Data Model
- Operational Data Model
- Application View Model

These six models and their inferred data represent the executed policy of the IT organization of the enterprise with respect to data metadata. If key data metadata is not defined, collected or is not able to be properly employed in the efficient and effective development, use, and maintenance of databases and information systems then blame should squarely lay at the feet of data administration, database administration, and subject matter functional experts.

There are many other meta models besides these six that comprise full IT systems development. Collectively these six and the other meta models should form the complete metadata repository that is at the heart of a completely integrated CASE (Computer Aided System Engineering) environment. If all these meta models are fully defined and integrated into the critical path of information technology then productivity and quality will increase, and costs and risk will decrease to such an extent that the overall cost of building such environments will be negative



## 8.1 Semantic Hierarchies and Data Element Model

The semantic hierarchy and data element meta model database should be a data administration project. That is, it should be their responsibility to determine requirements, build the database design, specify and possibly implement the data element metadata repository subsystem, to develop training, documentation, and to assist users in its proper use as they work to achieve enterprise-wide data standardization. The other meta models that are described below should be a joint responsibility of data administration, database administration, and subject matter functional experts.

The semantic hierarchy and data element models are essential first models to be built and implemented because flowing out of these models is the ability to achieve enterprise-wide data standardization, and the ability to “see” where data sharing can occur.

## 8.2 Specified Data Models

The specified data model represents a first step in achieving data standardization in support of enterprise-wide database. That is, the creation of data model templates. These represent commonly used collections of entities, attributes, and relationships that are organized by subject areas.

Each entity is a coherent policy-based collection of attributes that are in at least third normal form so as to guarantee policy based homogeneity. Data elements map to the attributes across and within these specified data model templates.

Semantic hierarchies from the data element meta model in Figure 14 are mapped to attributes that more precisely define the semantics within the attributes of their containing entities. The semantics of the attributes must, of course, represent a subset of the semantics previously assigned to the data elements.

Specified data models are not database designs. Its paradigm is subject, entity, attribute. Relationships connect various entities, and can related entities across subject areas. These models are best accomplished by functional data administrators or experts. These specified data models are then available to database designers as they employ these data model templates to create implemented databases.

The specified data model attributes are quickly created through the identification of one or more data elements that are to be the semantic templates for attributes within entities. The inherited semantics from the hierarchies of meta category value types, meta category values, data element domains, data elements, and entity subject areas hierarchies all contribute critical components to the semantics of an attribute. Attribute definitions can be completely automatic through the definition fragments that are immediately available from the attribute’s inherited semantics.

In the case where a data element is associated with more than one attribute in an entity as would be the case for home-, office\_, cell\_, and fax\_telephone\_number, the ability to create a local name and even a local definition is present. Available, of course are all the already inherited semantics.



Because of these work saving assists, creating a specified data model is an effort that is characterized by:

- Significantly shorter times to create entities because of all the inherited semantics
- Lowered risk because when things are the same they will be semantically defined the same
- Increased quality because there will be more time for important semantic component parts
- Increased productivity because the process of creating the specified data model can be accomplished by functional experts rather than just data administration staff.

### **8.3 Implemented Data Models**

Database data models exist in two necessary forms: DBMS independent and DBMS dependent. Both these models are needed because databases that are actually deployed may have table designs changed to meet the needs of different DBMSs, operating systems, data architecture classes, or computer hardware capacities. In such cases, the “real” database design is not intrinsically changed, just deployed differently. Consequently, the DBMS independent database designs are needed.

Independent database data models are thus database data model templates for operational database data models deployed across the enterprise. The triple of the implemented data model are schema, table, and column. Data elements are mapped to columns. Attributes are mapped to the columns. Semantic hierarchies mapped to columns give a more precise layer of semantic subsetting of the attributes and in turn of the data elements that are mapped to the columns. A database is thus a collection of tables of columns that map to attributes of entities. Attributes from more than one entity may map to columns of a single table.

A table then is not necessarily a homogeneous set of attributes within a policy based entity as would be the case in a well designed specified data model entity, but may be artificially contrived to meet the needs of a particular data architecture class such as original data collection, Transaction Data Staging Area (TDSA), subject area database, data warehouses (wholesale or retail), or reference tables.

The implemented data model is distinct from the specified data model. It may merely be the technology dependent transformation of a collection of entities from within the specified data model whereby subject areas, entities, and attributes serve as data structure templates for tables and columns within an implemented database. The implemented data model may also represent a stand alone database design that has not been mapped to the specified data model. That of course is not recommended. The implemented data model may have a multiple subject-area scope and thus may represent interrelated collections of entities, attributes and relationships from the different subject areas.



Another key difference is that tables within the implemented data model are able to belong to database objects, while, in the specified data model, entities are merely standard data structures that are able to be deployed as tables (possibly within database object classes) of a particular database.

A key value of the implemented data model is to convey a COTS (commercial off the shelf) package vendor's data model in a form that is understood by the rest of the enterprise. If TDSA data architecture databases are built with push-data from the COTS package, then downstream pull-database applications, such as subject area databases and data warehouse databases, can be built.

The implemented database's main purpose and is to represent a database that is to be implemented on some technology dependent platform. That is, through one or more DBMSs and one or more specific computers. If a human resources schema is implemented under a SQL DBMS on three different platforms, for example, MVS, Unix, and Windows/NT, then while there would be only one Schema (and tables and columns), there would be three different sets of DBMS Schemas, DBMS Tables, and DBMS Columns.

The greatest benefit from this approach to the development of the implemented data model is the ability to employ predefined semantics from the specified data model. That is, from subject areas, entities, attributes, data elements, meta category value types, and meta category values. After a period of time, this approach should enable enterprises to develop entirely new schemas, tables, columns, etc., with a minimum of original effort. The data architecture classes that benefit most are subject area databases, wholesale and retail (data mart) databases, and TDSA databases. Original data capture and reference data databases benefit the least from this approach as these databases are the source of the majority of data structures and semantics for the enterprise.

## **8.4 Operational Data Models**

DBMS data models for the operational set of schemas that exist within specific hardware, operating systems, and DBMSs. The triple here is DBMS schema, DBMS table, and DBMS column. DBMSs columns are mapped to columns from the implemented data model tables. Because of the divergence of DBMS vendor implementations of SQL standards because of different performance characteristics of operation systems and hardware, there may be different operational data model variations of the same implemented data model.

As stated at the outset of this section, the operational data model is not only DBMS specific, it is also targeted to a specific operating environment. If the only set of data models that an enterprise retains are those that reflect the operating DBMSs, then there will be no context independent business semantics from which the business can be understood and changed.

Thus, there can be multiple operational data models, each with a somewhat different design due to DBMS characteristics and performance requirements for every database. Additionally the data models contained in the data structures within the specified data models could appear in multiple implemented and operational data models.



Because most enterprises do not define, interrelate and track the different deployments of the same set of business semantics they do not really have any control over the most valuable resource, information.

Operational data models are created through three avenues: forward engineering, original creation (due to capabilities that may be missing in certain DBMSs) and reverse engineering. Most often, the enterprises will employ a combination of top-down and bottom-up techniques.

Bottom up techniques proceed via reverse engineering to create the realistic portrayals necessary to give implemented databases a real rather than a theoretic existence. Once the top-down and bottom-up are joined, enterprise database begins to emerge.

What may seem to be an inordinate amount of work in order to achieve enterprise database is really not. The model building strategy described in this approach ensures that there is maximum reuse of already defined business based.

## **8.5 View Data Models**

Application view data models, that is, view models that interface the operational data models to the database applications consist of views and their view columns with the associated hierarchies of joins and selects to give applications a flat “record set” for processing. Included in the views are rename clauses and on-the-fly calculations. View elements map to DBMS columns and as appropriate compound data elements and derived data elements.

## **8.6 Approach Summary**

Data elements mapped through attributes of the specified data model and/or through columns of the implemented data model enable fact based semantic homonyms regardless of name changes. Semantic hierarchies that are attached to data elements, which in turn are mapped to attributes or columns enable the identification of semantic homogeneous or related context dependent business facts that exist ultimately as DBMS columns.

Data elements form the semantic templates that can be employed to control the semantics of attributes within specified data model entities. This saves great time in specifying attributes and also enables rapid where-used reports across entities of different subject areas.

Specified data model templates, in turn, serve as templates for complete or partial tables within the design of databases. This too saves resources in specifying database tables, and because the data elements are, by inheritance, mapped to the database table columns, where-used reports across tables from different schemas can be produced. And, because implemented data models are mapped to specified data models through the mapping of columns to attributes, cross reference subject, entity, and attribute reports can be produced.

Implemented data model collections of schemas, tables, and columns can finally be employed to deploy different operational data models. Different operational data models may be needed to accommodate different DBMS-based database designs that had to be created as a consequence of difference in hardware configurations, operating/system differences, and in the different types of data models that are able to be created by any given DBMS.



To complete the process, DBMS data model columns are mapped to application view columns to thus enable the full complement of IT staff to understand which business information systems are collecting, reporting, and updating enterprise business facts.

The complete integration of these models, that is, semantic hierarchies, data elements, specified data models, implemented data models, operational data models and application view models finally gives enterprises the metadata through which integrated, shareable, enterprise-wide data standardization can be achieved.

The strategy for researching, identifying, defining, and deploying data elements within an enterprise can be highly efficient. It is fundamentally based on the premise that enterprises have a finite set of business facts that are used over and over in different contexts and computing environments. That this assertion is true is supported by the examples in this book.

While the sensibility of this approach is intuitive, and widely recognized, it is almost never employed. That is because the approach for data element metadata development has been isolated into stand-alone repositories and not integrated within a CASE environment through which the benefits of “define-once use many times” can be realized. Without wide deployment and without CASE integration, isolated data element repositories have almost always fallen out of favor, and in times of financial distress, have been difficult to support and are often discontinued. Given that prior proponents of such projects are then identified as having wasted scarce corporate resources, future proponents of such projects are seen as being foolhardy at the very minimum.

Make no mistake, however, the benefits have always been present, but seldom realized due to poor implementation and integration strategies. Thus, properly designed and implemented meta data, CASE-based, repository projects have always been justifiable. The implementation costs from such projects, if implemented through pre-existing designs, CASE and code generators are commonly returned on the first use project.

A key determining characteristic as to whether your organization has implemented an environment that leads to enterprise-wide data standardization is just not the existence of a data element registry, but the existence of a use environment that parallels the other data meta models such as Specified, Implemented, Operational, and Application View. Without these other data meta models the inventory of data elements will essentially be the same as the inventory of columns. You will not have achieved the 20-30 to 1 ratio between columns and data elements. Rather, if you have 50,000 columns you will probably have about 30,000 data elements. The reason for the difference in quantities is that you are 20,000 “data element” definitions behind in your work. By the time that recognition occurs, you will probably then realize that for every database of 1500 column that comes into existence you now have an additional set of 1500 data elements to define. Not a “pretty picture.” In short, the faster you go, the “behinder you get.”



## Appendix 1

### Referential Integrity Actions

Referential Integrity and Actions			
Selected Referential Integrity Action	Target of Referential Integrity Action	If the specified referential action is	Then the Referential Action Consequence is
On Update	Parent	No action	Referential integrity is not checked. If a relationship error is caused, it remains.
		Restrict	<p>If there is an attempt to change a parent's primary key from one value to another value that does not already exist, then</p> <p>CASE: if there are already existing children that already have that intended new value then the attempt to change the parent's primary key value is rejected.</p> <p>CASE: if there are no children then the change to the parent's primary key value is allowed.</p> <p>If there is an attempt to change a primary key value from one value to another and if the intended new primary key value already exists, the change request is rejected on its face as duplicate primary keys are not allowed.</p>
		Cascade	<p>If there is an attempt to change a parent's primary key from one value to a different value, then</p> <p>CASE: if the intended new primary key value does not exist, then the corresponding old values that exist in the child records are changed to the intended new value.</p> <p>CASE: if the intended new primary value already exists then the update of the primary key value of the parent is rejected on its face as duplicate primary keys are not allowed.</p>





Referential Integrity and Actions			
Selected Referential Integrity Action	Target of Referential Integrity Action	If the specified referential action is	Then the Referential Action Consequence is
		Set Default	<p>If there is an attempt to change a parent's primary key from one value to a different value, then</p> <p>CASE: if the intended new primary key value does not exist, the corresponding old values that exist in the child records are changed to the default value established for that column.</p> <p>CASE: if the intended new primary value already exists then the update of the primary key value of the parent is rejected on its face as duplicate primary keys are not allowed.</p>
		Set Null	<p>If there is an attempt to change a parent's primary key from one value to a different value, then</p> <p>CASE: if the intended new primary key value does not exist, the corresponding old values that exist in the child records are changed to the ZERO or Null value.</p> <p>CASE: if the intended new primary value already exists then the update of the primary key value of the parent is rejected on its face as duplicate primary keys are not allowed.</p>
	Child	No action	Referential integrity is not checked. If a relationship error is caused, it remains.
		Restrict	If the intended new value in the foreign key of the child does not exist as a primary key value of some parent record then the update is rejected.
		Cascade	not applicable



<b>Referential Integrity and Actions</b>			
<b>Selected Referential Integrity Action</b>	<b>Target of Referential Integrity Action</b>	<b>If the specified referential action is</b>	<b>Then the Referential Action Consequence is</b>
On Delete		Set Default	If the intended new value in the foreign key of the child does not exist as a primary key value of some parent then the foreign key value of the child is set to the default value established for that column.
		Set Null	If the intended new value in the foreign key of the child does not exist as a primary key value of some parent then the foreign key value of the child is set to Zero or to Null.
	Parent	No action	Referential integrity is not checked. The parent record is deleted and the child records are left with erroneous references to the non-existent parent.
		Restrict	If there are any child records that have the primary key value of the parent as their foreign key value, then the request to delete the parent record is rejected.
		Cascade	If there are any child records that have the primary key value of the parent as their foreign key value, then these are deleted at the same time the parent record is deleted.
		Set Default	If there are any child records that have the primary key value of the parent as their foreign key value, then the foreign key value in the child records is set to the default value established for that column.
		Set Null	If there are any child records that have the primary key value of the parent as their foreign key value, then the foreign key value in the child records is set to ZERO or to Null.
	Child	No action	Not applicable
		Restrict	Not applicable



Referential Integrity and Actions			
Selected Referential Integrity Action	Target of Referential Integrity Action	If the specified referential action is	Then the Referential Action Consequence is
		Cascade	Not applicable
		Set Default	Not applicable
		Set Null	Not applicable



## Index

American National Standards Institute .....	8
ANSI .....	6, 8, 11-14, 16, 56, 64, 69, 70, 87, 89, 96
ANSI SQL .....	6, 11-13, 56, 64, 69, 70, 87, 89, 96
ANSI SQL:1999 .....	6, 64, 69, 70, 87, 89
Architecture .....	i, vii, 6, 7, 13-15, 18, 22, 38, 56, 57, 71, 72, 79-81, 83, 101, 102
Assertion .....	1, 35, 104
Binding .....	20, 72
Business Event .....	5
Business Function .....	5
Business Information Systems .....	vii, 5, 40, 104
Business Policy .....	5, 40
Call Level Interface .....	12
COBOL .....	4, 9
CODASYL .....	69
Column .	2, 3, 6, 8-10, 14, 18, 19, 22, 27, 29, 30, 35, 44, 45, 53, 56-60, 62-74, 77-79, 81-83, 85-94, 96, 101, 102, 104, 106, 107
Compound data element .....	32-34
Concatenated key .....	46
Concept of Operations .....	i, vii, 15
Contract .....	41
Data administration .....	10, 55, 70, 81, 99-101
Data Architecture .....	6, 7, 13, 14, 18, 38, 56, 57, 71, 72, 79-81, 83, 101, 102
Data element	1, 2, 4, 6, 8-10, 14, 21, 22, 24-27, 29-35, 37, 38, 40, 42, 44, 45, 48, 51-54, 58, 59, 66-68, 71, 72, 74, 75, 78, 79, 81, 83, 89, 90, 94, 98-100, 104
Data element domain .....	29, 30, 42, 44, 67, 72, 90, 94
Data element name .....	22, 45, 68
Data Elements . . .	1, 2, 4, 5, 8, 9, 11, 14, 15, 18, 21, 24, 26, 29, 31-36, 38, 40, 42, 45, 51-54, 56, 58, 66, 81, 83, 92, 98-104
Data file .....	8
Data integrity .....	12, 33
Data integrity rule .....	33
Data Loading .....	8
Data Models . .	vii, 1, 4-8, 11-15, 27, 30, 37, 38, 45, 54, 56, 57, 59, 70, 71, 78-80, 82-85, 87, 92, 95, 96, 98, 100-104
Data Semantics .....	vii, 4, 6, 8, 15, 17, 19, 23, 34, 37, 38, 57, 59, 71, 81, 98
Data Standardization .....	1, 2, 4-6, 9, 13, 14, 19, 35, 36, 48, 98-100, 104
Data Structure .....	6, 10, 18, 19, 32, 54, 57, 69, 80, 93, 101
Data type .....	10, 21, 22, 31, 44, 54, 58, 64, 67, 69, 78, 83, 87-90
Database Management System .....	1, 6
Database object .....	5, 13, 57, 102
DBMS .	1, 4, 6, 8, 9, 11, 12, 14, 18, 21, 27, 29, 30, 37, 45, 48, 51, 54, 56, 57, 59, 72, 80-95, 98, 101-104



DBMS column .....	6, 8, 9, 27, 29, 45, 56, 57, 83, 85-90, 92-94, 102
DBMS schema .....	6, 57, 72, 83-86, 89, 91-93, 95, 102
DBMS table .....	6, 9, 27, 57, 83, 85-87, 89-93, 95, 102
Derived data .....	24, 33, 35, 103
Derived data element .....	33
Domain .	vii, 1, 9, 10, 24-27, 29, 30, 35, 42, 44, 51, 52, 56, 57, 65, 67, 69, 72, 79, 85, 86, 90, 94, 96
Enterprise . .	vii, 1, 2, 4, 6, 8, 13, 14, 17, 22, 23, 26, 33, 35, 37, 40, 46, 56, 57, 66, 70, 71, 73, 79, 81-83, 92, 95, 96, 98-104
Extent .....	99
File .....	4, 6, 8, 9, 13, 24, 35, 37, 54, 57, 64, 79, 80, 91-93, 95, 96
Foreign key .....	8, 9, 38, 46, 48-51, 53, 58, 62-66, 72, 73, 75-78, 83, 89, 106, 107
Form .....	4, 12, 23, 26, 29, 42, 45, 46, 51, 56, 57, 60, 70, 73, 81, 82, 98-100, 102, 103
Hierarchy .....	17, 18, 20-22, 38, 40, 44, 59, 80, 94, 100
Implemented data model .	5, 6, 8, 9, 15, 16, 27, 35, 37, 53, 56-61, 65, 69, 72, 73, 78-81, 83, 85-87, 89, 90, 92, 99, 101-103
Information system .....	5
ISO .....	8, 25
Job .....	5
Knowledge Worker .....	4, 5
Knowledge Worker Framework .....	4, 5
Life Cycle .....	40
Map .....	38, 46, 57, 58, 73, 82, 100, 101, 103
Meta Models .....	98-100, 104
Metabase .....	i, vii, 2, 4, 12-14, 16, 23, 24, 40, 42, 48, 50, 56, 65, 66, 72, 77, 85, 92, 93, 96
Metadata . .	1, 8-13, 30, 34, 35, 40, 48, 59, 62, 64, 66, 75, 76, 79, 85, 86, 89, 91-93, 98-100, 104
Metadata Repository .....	9, 12, 30, 98-100
Metric .....	31
Mission .....	5, 35, 40, 83, 95
Multivalued dependency .....	46
Object .....	5, 13, 57, 69, 102
ODBC .....	12
Operating Systems .....	12, 96, 101, 102
Operational data model .....	5, 6, 8, 13, 15, 27, 35, 38, 56, 57, 72, 78, 79, 82-93, 95, 99, 102
Operational Data Store .....	56, 81
Ordered .....	69, 87
Organization .....	vii, 1, 5, 8, 14, 40, 43, 44, 67, 99, 104
Primary key .....	45-51, 53, 62-65, 72, 73, 75-78, 87, 90, 91, 105-107
Project .....	6, 33, 35, 36, 46-50, 60, 62-65, 73, 75-77, 100, 104
Project Management .....	60, 62, 65
Recursive .....	17, 48, 75
Reference Data .....	7, 18, 56, 81, 82, 102
Referential integrity .....	9, 50, 51, 60, 77, 91, 105-107
Report .....	9, 10, 23, 24, 64, 73, 78, 82, 87, 91, 92



Repository .....	9, 12, 30, 98-100, 104
Resource .....	5, 40, 95, 103
Role .....	vii, 21, 22, 29, 44, 46-50, 60, 62, 63, 75-77
Row .....	4, 18, 45, 51, 69, 70, 72, 87, 90, 91
SC32 .....	8
Schema .....	6, 8, 9, 11, 12, 54, 56-60, 62, 64-67, 70-72, 78, 79, 81-86, 89-93, 95, 101, 102
Screen .....	9, 10, 31, 35, 42, 59, 66, 67, 70, 85, 89, 92, 93
Semantics .....	vii, 2, 4-6, 8, 9, 11, 13-15, 17-24, 26, 29-31, 33, 34, 37, 38, 40, 42-44, 48, 53, 54, 56, 57, 59, 66-71, 73, 77-79, 81, 82, 89, 90, 92, 95, 96, 98-103
Specified data model ..	5, 6, 8, 15, 27, 37-39, 45, 46, 52-57, 59, 60, 65, 69, 71-73, 75, 78-81, 83, 93-95, 99-103
SQL ...	2, 6, 8, 11-14, 16, 31, 35, 38, 54, 56-59, 64, 67, 69, 70, 78-81, 83, 87-89, 91-93, 95, 96, 102
SQL:1992 .....	88
SQL:1999 .....	6, 8, 13, 31, 57, 64, 69, 70, 87-89, 93
Staff .....	6, 12, 35, 36, 55, 70, 84, 101, 104
Table ..	ii, 1, 2, 6, 8-11, 18, 24, 27, 29, 31, 35, 45, 46, 50, 51, 53, 56-73, 75-80, 83, 85-95, 101- 103
Third normal form .....	12, 46, 56, 73, 82, 100
Time charges .....	77
User .....	vii, 11, 18, 21, 22, 40, 64, 65, 69, 87
Value Sets .....	24
Warehouse Databases .....	79, 81, 82, 84, 102
WG3 .....	8

