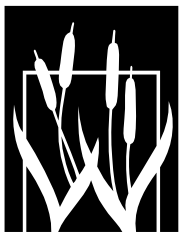


Whitemarsh
Information Systems Corporation

Database Objects

The Foundation Stones of Enterprise Database

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

Table of Contents

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Table of Figures

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

README.1ST

There are generally considered to be three classes of objects: display objects, wholly contained process objects, and business objects. Display objects embrace buttons on a screen, a drop list of menu choices, a graphical user interface (GUI), or complete engineering drawings. Wholly contained process objects are for example, the COSINE function, a nautical distance function that when given two geographical coordinates returns the geographic distance between them, or a well defined process that takes standard arguments and returns a specific value such as asking for the net asset value for a business given all assets and liabilities. Finally, business objects encompass business components like an insurance policy that performs in a certain manner.

All three object classes have their proponents and detractors. What all three object classes have in common is that they are first and foremost self contained software in the form of an executable or embedded process that behaves according to certain fixed rules. *This book is about none of them*. So, if you want a book about any of the these three well known objects, then this is not that book.

This book is about database objects, a concept that shares some common names and some common definitions as the other three object types. Because of common names and some common definitions, the reader may initially be confused as to the nature of database objects. Database objects however, are unique to database. They are identified, designed, implemented, operated through, evolved, or maintained through just one type of data processing facility, a database management system (DBMS). If the available DBMS is an ANSI SQL:1999 DBMS then database object definition and use can be direct. Otherwise, database objects can only be indirectly approached through proprietary facilities in one or more DBMSs.

Notwithstanding the availability of SQL:1999 and 2003 DBMSs, database objects are absolutely essential to understand, specify, implement, and maintain world-wide heterogeneous database. Without adopting a database object approach enterprise database failure is certain.

Database objects are not new. They were started in certain DBMS types in the late 1960s. Relational DBMS, however, stopped the march to database objects. It was not until the ANSI SQL:1999 data model moved away from the relational model and not until a whole programming language was incorporated into ANSI SQL:1999 that the march towards database objects was restarted. Even if the twenty-year delay had not happened, computers, networks, languages and operating systems were just not sophisticated enough to make database objects successful.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1

WHY DATABASE OBJECTS?

Database is the application of quality organization, planning, and management. Central to these organizational characteristics are carefully crafted policies and procedures. Designed well, business policies and procedures become database objects[1] that can be deployed throughout the organization in a client/server fashion to maximize sharing and consistency while minimizing data hoarding and irregularity.

Database objects are the foundation stones for enterprise database. Organizations not pursuing their specification, implementation and evolution are condemned to complicated, redundant specifications, expensive implementation and difficult operation, evolution and maintenance. This book, a continuation of the Whitemarsh database books (1984, QED; 1991, QED; 1994, John Wiley & Sons; and 2006, Whitemarsh):

- Shows that database objects are not new, rather they are an optimization of true concepts from the past.

- Details how database objects are now practical to specify, implement and maintain through CASE, code generators and repositories.

- Defines the changes that database objects bring to traditional systems analysis, design, implementation, testing, and maintenance activities.

The database object concepts in this paper are not new. They were formulated almost 30 years ago by Matt Flavin. During the Seventies, Matt who worked for Infodata of Rochester, NY and Fairfax, VA. Infodata accomplished very early database management system research and development. Infodata's DBMS, Inquire, was widely used in the U.S. Federal Government. Matt represented Infodata to the ANSI Database Languages committee, H2, in the late Seventies.

Matt clearly knew the difference between DBMS and database. The former a technology, and the latter the application of quality planning, organization, and management. When Matt joined Yourdon, Incorporated in New York City, he began development of an information modeling discipline based on database objects. As Matt would often say, "database objects are squarely based on an enterprise's policy." Matt insisted that fundamental business policy discovery and formulation was the very first step in discovery. Matt's Yourdon Monograph,

---

[1] Database objects are unique to database. Database objects are defined through, updated, manipulated, and deleted through the facilities of database management systems. Not just any DBMS however. Only those DBMSs containing the database object facilities contained in ANSI standard SQL/3. Simply stated, a database object is an ANSI DBMS controlled expression of data that is transformed through a discrete set of value states according to a fixed set of rules and processes matching a well formed and defined business policy. A database object therefore contains four interrelated parts: a data structure, data integrity rules, value state transformation processes, and a set of predefined states.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Fundamentals of Information Modeling, set out the basic steps to identify and specify database objects. Matt's contribution to database was cut short with his untimely death in 1984.

Database objects existed only in non-relational DBMSs in some form during the 1960s and 1970s. The relational model specifically precludes any capability for database objects. The SQL language, based on the relational model also precluded database objects. This preclusion existed for the SQL:1986, SQL:1988, and SQL:1992 standards. It was not until the ANSI database languages committee, H2, working since early 1993 on the specification of the SQL after SQL:1992, formulated the essential linguistic components of database objects. These exist under the name of SQL:1999. This book presents database objects first as requirements, then as an example, then through an enumeration of the SQL:1999 facilities that enable them. Additional facilities supporting database objects are included in SQL:2003.

Database objects, but not through SQL, were validated 20 years ago through a large scale database project for the U.S. Army.

> An Army General in 1985 wanted PECO, a DoD contractor located in Iowa, to develop 10 systems in one year when the contractor had previously developed 2 systems in three years; and at less cost. By implementing the Whitemarsh methodology that is based on CASE, code generators, and repositories, the mission was accomplished. The systems, instead of costing 1000% (10 times 100%) only cost 360%, a reduction of 64% per system.
>
> As to the long term benefit to the Enterprise? The Army General proposed a modification to the fundamental set of algorithms that governed Reliability, Availability and Maintainability (RAM). The 10 systems were already deployed. These systems were collecting data world-wide and producing the RAM studies the Army desperately needed to predict its material reorders. When the change request came down, the repository was immediately put to use isolating the exact area of the specifications that had to change. The specifications led to the systems; the systems led to the programs; and the programs led to the modules. The changes were identified. When it was reported that all 10 systems could be changed and reassembled, all documentation changed, all user-manuals regenerated, reprinted, and redeployed in two weeks, the message came back down, "Stop! We merely wanted to give you six month's notice!"

Everything in this book has been accomplished many times since 1985 with commercially available software and hardware. Based on that foundation of accomplishment, this book adds the formal definition of database objects, complete examples from real world applications, the ANSI SQL clauses appropriate for specifying database object completely within the firewall of an ANSI SQL based DBMS, the methodology to accomplish database objects, and the design of the repository to store database object metadata.

Database is fundamentally a multiple-user information system component. Databases are the least useful when they are private. Fully developed and world-wide, databases enable enterprises to share policy, plans and work products. At the core of database is its most critical component, database objects because:

- Businesses are world wide, heterogeneous, and client server.
- The only common component of database information systems through out the enterprise is SQL.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Consequently, database objects must be completely specified within the firewall of SQL if there is any hope for consistent, world-wide semantics.

1.1 The Business Case for Database Objects

Distributed data and processes either through client/server or the Internet-base transactions, are here to stay, and rightly so. Not only are they empowering, they are essential because enterprises are highly distributed and world wide. Enterprises must be able to respond to local needs, laws, customs and mores. But, if business are designed and tuned to respond to local situations, how can they act in concert within their world wide communities? How can you have world-wide consistency and semantics without suffocating local needs and practices? How can both ends of the information resources spectrum be satisfied?

Business data needs far exceed today's DBMS's two dimensional table capabilities. Businesses cry out for semantically rich data management to meet business needs across world-wide, heterogeneous hardware and operating system environments. Business data management environments must behave consistently regardless of their host computing hardware environment and must be easy to specify, implement, use and maintain.

Businesses require hierarchies of complex data tables, collections of integrated rules for data integrity, well defined procedure sets, and fixed transformations that move a business policy from one well defined state to another. Examples of business needs include insurance policies and claims, court cases and documents, public safety incidents, sales and marketing databases that contain customers, sales organizations, forecasts, orders, deliveries, and product sales statistics, inventory control and deployment, and human resources. The business case for database objects is compelling:

> Two managers were trying to produce a three year marketing plan. One manager stated that the sales in the East were up. The other said they were flat. The first showed numbers to prove the point. The second showed an equally impressive set of numbers that proved the counter point.
>
> Finally, it was discovered that one manager was using “sales” based on sales organizations credited for specific sales, and the other was using “sales” based on addresses of product deliveries. In exasperation, they both exclaimed: “How can we plan when we’re not working off the same *sheet of music*!”
>
> What should be on the *sheet of music?* The notes for the oboe’s part, the violins, or the orchestra director? The orchestra director’s score not only contains a unified set of notes for all parts, but also the rhythm (cadence, meter, pulse), tempo (momentum and speed), articulation (clearness, distinctiveness), and expression (phraseology and style).
>
> The marketing plan certainly required much more than just notes. To be effective, accurate, and able to respond to unforseen emergencies (first violinist’s broken string), it requires both the static (sales numbers) and the dynamics (all the environmentals). With both, agreements (quality music) can be reached. Plans can be executed, tracked, and adjusted, just like a good symphony.

This book is all about the development of the symphonic scores through which organizations plan, conduct and modify their enterprise (establishment (static) and campaigns (dynamic)). In

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

today's information system's parlance, the symphonic scores are database objects[2]. Because we're talking about enterprise-wide database, the term *database objects* becomes an artful shorthand for what is involved in a successful business symphony.

But, what forms the basis of an database object? Simply, it is a business' policies and procedures. While policies can exist without procedures, the converse is not true. This ontological priority dictates that procedure is dependent on policy. Not only do they go together like hand and glove, the glove (procedure) serves no useful purpose without the hand (policy).

A database object is a person, place, or thing that has internal consistency, and is transformed from one state to another through well defined rules. The minimum value states are null and valued. The internal behavior of a database object as it transforms from one state to another is immaterial to its user. Database objects conform to the requirements of business rather than the converse.[3]

Policies and procedures, that is, database objects, bring order, consistency, and predictability. The larger the enterprise, the greater the dependence on policies and procedures. Data is the evidence of policy execution. An employee's record is proof that policies have been carried out. Procedures are the techniques, methods, or processes by which policies are carried out. If an enterprise has the policy is to be profitable, then its balance statement, produced by processing all the general and subsidiary journals is the measure of adherence to the policy. If policy is met, the enterprise must be profitable.

Within an enterprise, policies, and in turn, data exist in two major areas: infrastructure and programmatic. Infrastructure areas address internal policy, such as human resource management, finance, and support services (e.g., plant security, information systems, and facilities). Programmatic areas address external products that are designed, manufactured, marketed and sold. For a traditional business this might be steel products, building products,

---

[2] In this book, a database objects exist in two forms: defined meta data and instances. The context of its use as metadata or instance is clear. When ambiguous, then the term *database object metadata* or the term *database object instance* is employed.

[3] The internal specifications of a database object are independent of its implementation. Because database object specifications are ANSI SQL standard, different SQL DBMS vendors (here "DB" stands for database object) are free to implement the ANSI SQL specifications as they like just so long as two conditions are true: the database object specifications are portable from one SQL/DBMS to another, and the behavior of the same database object is same, from the user point of view, even though the database object is implemented by different SQL/DBMS vendors. database objects range from the trivial to the complex. A *trivial* database object is: 1) is a simple data structure (a set of single value fields), 2) is instantiated through simple databases processes (INSERT, MODIFY, DELETE) that are 3) part of one encapsulating information system, and 4) takes on a minimum of two values states: null and valued. A complex database object is: 1) a complex data structure with multiple segments containing single, multi-valued, groups, repeating groups, and nested repeating groups of fields, 2) in instantiated through collections of database processes that are 3) part of one or more collections of complex information systems, and 4) that take on a whole series of discrete business policy recognizable states from null to any number of discrete valued states back to a null state. In short, the full life cycle of a business resource (employee, contract, asset, etc.).

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

automobiles, or houses. For an intellectual product business these might be mortgages, insurance policies, courses, and students.

Policies and their associated data, address the well bounded infrastructure and programmatic areas. The data takes on common every day names such as employees, facility, mortgage, insurance policy, and student. The data representing these common names are complex, that is, whole multiple-level structures.

The procedures are named, and their data actions are associated with specific subsets of the named data structure. The names of the procedure sets represent data structure transformations from one recognizable state to another. Each state represents a determined value set within the business. Procedure examples include: establishing an employee requisition, accomplishing employee hiring, and performing employee assignment.

Enterprise database is an organizational operating condition in which there is both defined policy coherence and integrity as well as consistency in policy transformations throughout the enterprise irrespective of functional and organizational style and irrespective of policy transformation technology (that is, computers, operating systems, programming languages, and database management systems).

Organizations not pursuing database object specification, implementation, and management information system (MIS) evolution through database objects will never achieve enterprise database. Rather, they will be left with complicated, redundant MIS specifications, expensive MIS implementation and inconsistent difficult MIS operation, evolution and maintenance.

Enterprise database is the expression, population, use, and manipulation of all database objects. Enterprise database contains not only all “real” database objects, but also all the policies and procedures surrounding their specification, implementation and evolution. Value is not only in the “data,” but also in the specification of the data. The information technology assets of the enterprise are both its database objects and also its enterprise meta-objects. If only the former were valued, then only musical notes would be needed for a great symphonic score. Performances are differentiated however, from the grade-school band to first-rate orchestra because of the musicians’ talent that is coupled with the quality of the orchestra director’s interpretation of the score’s rhythms, tempo, articulation, and dynamics.

1.2 Not All Objects are the Same

While there are no silver bullets, objects can certainly help. Objects come in a variety of forms. There's traditional software based objects like screen "buttons" or software routines like COSINE, SQRT, etc. In short, there are three fundamentally different classes of objects:

- Database objects (described above),
- Business objects
- Process objects (display objects or wholly contained process objects).

A process object is a well defined unit of programming language code that is compiled and during its execution accomplishes a specific operation. Display object examples include icons,

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

boxes, and windows on a computer screen. Mathematic operations such as SUMs, AVGs, or the trigonometric operation sine or cosine have been known the data processing community almost since its inception. Other process objects include arrays, I/O channels, or complete end-user objects such as a word processing system.

A business object as specified by the business object management special interest group (BOMSIG) of the Object Management Group (OMG) is its 1995 OMB Business Application Architecture White Paper, Draft 2, is a representation of a thing active in the business domain, including a least its business name and definition, attributes, behavior, relationships, and constraints. A business object may represent, a person, place or concept. The representation may be in a natural language, a modeling language, or a programming language. Business objects are employed to represent whole insurance policies, automobile accident reports, patient medical records, and the like. Because a business object is, at its core, a process with encapsulated data it is not the same as a database object. Regardless of type, all objects share common properties: encapsulation, inheritance, and polymorphism[4].

While all objects bring together data and process, business objects when fully defined and deployed within the business environment, bring together enterprise policy and procedure. Business data is the consequence of business policy execution. Procedures are the business' methods. Quality business objects are reflections of quality business policy and procedure.

While it is obvious that business objects are needed, it is not at all obvious how to define, deploy, and manipulate them in world-wide, heterogeneous hardware and software environments while both empowering but not suffocating local needs and practices.

1.3 Traditional Computing Environments Inadequate

Traditional computer programming languages do not contain sufficient data modeling, access and processing facilities to fully handle business objects. Further, computer programming languages encapsulate data to such an extent that the defined and contained business objects are truly captive of the programming language within which they are defined, captured, stored, and manipulated. Simply, traditional programming languages are “data poor.”

Once programming language based objects are created they are so bound within the language's constructs and to the business' local needs and practices that any attempt at a world wide community of business objects is impossible. Finally, there is no world-wide standard business programming language. While “C” is certainly world-wide, it lacks robust DBMS

[4] Encapsulation means that the object is shielded from the “influences” of its outside environment. Standard money arithmetic processes defined within the insurance policy object regarding premium computation can be made independent of the currency of the money through which the premium is to be paid given via the arguments of dollars exchange rate and other standard inputs.

Inheritance means that any contained objects can presume on the properties of any containing object. A woman, a contained object, assumes the all properties of a human, the containing object.

Polymorphism means that an invoking command to compute REMAINING BALANCE may in fact invoke different processes depending on the “invoking environment.” That is, for example, whether the required remaining balance is for a loan, and invoice, or a real-estate mortgage.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

qualitites and is not able to be used by the “mere mortals.” C is the language through which compilers, DBMSs, and other end-user tools are created.

Similarly, traditional relational data management systems (DBMSs), the successors of previously far richer network, and hierarchical DBMSs, treat data far too simply to handle business' complex policies and procedures. Traditional relational DBMSs are “data and end-user language poor.” The only way to make traditional relational DBMSs handle the needs of business objects is to fully encapsulate the relational DBMS data within procedure rich programming languages. Once that is done however, we're right back to the programming language environment in which business objects are so bound to the language's constructs and to the business' local needs and practices that any attempt at a world wide community of business objects is impossible.

SQL is the language through which relational databases are defined and data is entered, manipulated, reported and protected. SQL is world wide, and because of the very strong ANSI and ISO standards activities SQL is essentially identical where ever it is used. Through 1992, however, the SQL language was only able to manipulate relational, two dimensional collections of database data. While elegant and simple, these data collections are clearly incapable of handling the complex needs of business, without significant, labor intensive systems analysis, design, and programming efforts.

Business objects, as described by the Object Management Group, are software products that live within traditional computing environments. OMG's business objects are defined, employed, and are manipulated by traditional object oriented languages such as C++, Smalltalk, 4GLs such as Sybase's Power Builder, Oracle Forms, Clarion for Windows, or Microsoft's Visual Basic, and are stored either in traditional file structures, relational database management systems, or hybrid object-relational database management systems. Because all these language environments operate differently on different computing environments, OMG's Business Objects cannot satisfy the demands of business environments for objects which:

- Are easy to specify, implement, use and maintain
- Operate on world-wide, heterogeneous hardware and operating system environments, and
- Behave consistently regardless of their host computing hardware environment

1.4 ANSI SQL:1999 SQL:2003

Since 1992, the ANSI and ISO database languages committees were expanding SQL well beyond its relational strictures. SQL:1999 is now powerful enough to handle business's need for objects, hereafter called database objects. SQL:2003 expanded these capabilities. SQL:1999 now contains both the data and manipulation facilities to handle both traditional data such as columns and rows, and also non traditional complex data structures for groups, repeating groups, abstract data types of arbitrary complexity, binary and character large objects (BLOBS and CLOBS), and full text processing. SQL:1999 also contains a full data manipulation language for stored

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

procedures, and a full complement of data integrity rules, actions, and procedures. Finally, SQL:1999 contains fully developed facilities for transaction management.

SQL:1999 does not however have language facilities to interface directly with the end user. That is, SQL does not have the necessary screen painters and full report writing languages that produce eye-pleasing end-user screens and reports. Not only is that not bad, that's good! SQL was designed expressly to be employed through end-user programming language environments, that is, through C, COBOL, and any myriad of fourth generation languages such as Oracle Forms, Sybase's Power-Builder, Information Builder's FOCUS Six, or SoftVelocity's Clarion for Windows. It is through the vendor proprietary, hardware and operating system specific facilities that SQL:1999 can satisfy the requirements of heterogeneous hardware, operating system, and end-user presentation and reporting environments.

It is precisely because SQL does not have end-user facilities that database objects are equally and commonly accessible from these languages. A SQL object is not tied to or captive within any one programming language. A SQL object is able to be defined independently from but commonly accessed through all the standard programming and end-user languages.

If SQL did have all the standard programming and end-user languages facilities then it would become just another business object language environment, which when fully employed would result in business objects that are so bound to the language's constructs and to the business' local needs and practices that any attempt at a world wide community of business objects would be impossible. In short, SQL is not just one of the languages through which business objects can be deployed. Rather, it can be used as the sole language for specification, implementation, and evolution. Because of this significant difference, the database objects can be defined at a level sufficient for world wide semantics without having to be suffocated by the needs of the local needs, customs, and mores.

1.5 Database Objects

Database objects "live" entirely within the domain of the DBMS. Database objects can be both persistent and non-persistent, and can be either single or multiple-table objects.

A persistent database object is one that is stored and is retrievable over long periods of time. An example is an insurance policy along with its full compliment of payments, renewals, and claims. Another example of a persistent object are rotating-three dimensional views of a mechanical part.

A non-persistent database object is one that is materialized and displayed but is not able to be re-materialized because some of its components are not retained after the database object's display is terminated.

Non-persistent objects are dynamically produced from database data and exist only for the life of their "display." An example might be evening news weather displays. The weather map, that is, the states, cities, streams, rivers, etc are all persistent database data. The actual streams of clouds, high and low pressure fronts, cloud formations, and the like are time-sequenced BLOBS that are dynamically displayed across the screen. While the displayed database objects may be recorded via videotape and redisplayed at a later time, the detailed components, which upon retrieval make up the non-persistent database objects, is not stored.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

By the time the news cast is over, the BLOB parts are discarded. Other than for a videotape replay, the complete set of the non-persistent database objects are gone. The persistent part is traditional data structures with the appropriate quantity of indexes. The BLOBs are just non-indexed streams of binary data that are stored in a very primitive format.

Persistent database objects are those that are stored in a database on a permanent basis. Included are traditional "relational" data, abstract data types of complex structures (like an entire auto accident claim that might include BLOBs, free text streams, etc.).

Single table database objects are those that are fully defined within a single row of an SQL table structure. The database object may further be stored within a single cell within a column. With SQL:1999, very complex structures can be defined within a column. This capability is quite common in hierarchical DBMSs like System 2000 and in independent logical file data model DBMSs such as Adabas, Model 204, Inquire, and Datacom/DB. Not only can a single column support single valued items, it can also support lists, sets, multi-sets, and abstract data types of arbitrary complexity.

For example, in a product sales database, the single table called sales has product number as the primary key with other columns for product name, product description, and the like. The sales column in contrast, contains product sales by year by month by region, district and territory by salesman. That's a single column with six dimensions of values. Prior to SQL:1999 such product sales information would require multiple tables. Since the salesman's object identifier is contained as a integral component of the sales data, the salesman's full set of data is accessible through normal SQL language processing. A referenced database object, that is, the referenced salesman's data is not considered a formal part of a single table database object.

Multi-table objects are those that are implemented across multiple tables. For example, an insurance policy may have several dozen tables that make up its full definition. One and only one table is considered as the database object's root table. The database object root table contains among many things, the object's identity column. A row from the root table is the head-row of the database object. All other related tables within the multi-table database object contain other information related to the object. In the insurance policy example, a claim might be a contained in one or more database object tables. Other database object tables would contain the underwriting information regarding the person about whom the policy was issued. The person is a different single or multi-table database object. Each table within a multi-table object may consist of single valued columns, it can also support lists, sets, multi-sets, and abstract data types of arbitrary complexity.

Not only do most businesses contain multi-table database objects, the majority of business applications are examples of multi-table database objects. A quick look at business applications reveals that inescapable conclusion.

Database objects, regardless of persistence and regardless of whether single or multi-table contain the same four-part composition:

- Database Object Structure: the set of data structures that map onto the different value sets for real world database objects such as an auto accident, vehicle and emergency medicine incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Database Object Process: the set of database object processes that enforce the integrity of data structure fields, references between database objects and actions among contained database object tables, the proper computer-based rules governing database object table insertion, modification, and deletion. For example, the proper and complete storage of an auto accident.

- Database Object Information System: the set of specifications that control, sequence, and iterate the execution of various database object processes that cause changes in database object states to achieve specific value-based states in conformance to the requirements of business policies. For example, the reception and database posting of data from business information system activities (screens, data edits, storage, interim reports, etc.) that accomplish entry of the auto accident information.

- Database Object State: The value states of a database object that represent the after-state of the successful accomplishment of one or more recognizable business events. Examples of business events are auto accident initiation, involved vehicle entry, involved person entry, and auto accident DUI (driving under the influence of alcohol/drugs) involvement. Database object state changes are initiated through named business events that are contained in business functions. The business function, auto accident investigation includes the business event, auto-accident-incident initiation, which in turn causes the incident initiation database object information system to execute, which in turn causes several database object processes to cause the auto accident incident to be materialized in the database.

A database object is specified to the SQL DBMS through the SQL definition language (DDL). All four components of a database object operate within the "firewall" of the DBMS. This ensures that database objects are protected from improper access or manipulation by 3GLs, or 4GLs. A DBMS that only defines, instantiates, and manipulates two dimensional data structures is merely a simplified functional subset of the DBMS that defines, instantiates, and manipulates database objects.

Database objects are discovered through the systems analysis and design technique called resource life cycle analysis. This technique, formulated by Ron Ross[5]. A resource life-cycle is the set of essential steps for manipulating a critical corporate resource. Examples of corporate resources are: employees, contracts, customers. At the highest level, each resource is a database object. Also at this high level, each resource life-cycle represents the highest level set of states that the database object proceeds through from creation to termination.

[5] Ron Ross's Resource Life Cycle Analysis is presented in the book, Resource Life Cycle Analysis, The Database Research Group, Boston, MA. 1994

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1.6 Database Objects Summary and Benefits

The database object benefits include:

- Whole containment within SQL DBMS
- Access to both type and instance components
- Complete expression through syntax
- Import and export through ISO/ANSI standard SQL facilities
- Ability to be distributed and consistent operations via all SQL compliant DBMSs
- Independence from presentation-layer and operating-system bindings

Because database objects are wholly contained within SQL DBMS they can be centrally accessed and manipulated regardless of the end-user environment, that is, batch, on-line, stand-alone "fat" clients, thin clients, Internet, or traditional client/server.

SQL DBMS databases contain both type and instance data. Type-data is metadata. Instance data is traditional data. For an employee database, type data are the table, column, integrity definitions, stored procedures and the like. Instance data are the actual employee records.

Type data is critical for distributed data and process databases because it can be queried to then determine the exact semantics required for database access operations. For example, if there is a currency exchange data object on a server, a query can determine the arguments and data types of the required inputs. Transactions can then be formulated and successfully accomplished.

Database objects are completely expressible as syntax enabling SQL compliant DBMSs to receive new database object syntax for inclusion, requirements to delete database objects from the type and instance data, and command and data strings that cause updates to standard reference data that can act as dynamic integrity constraints.

A fundamental requirement of any compliant SQL DBMS is that it be able to import and export both type and instance data through standard SQL commands. Applications are able to export or import data through standard character set strings. When a new computer site is established, a central server can be activated to down load command strings of syntax and standard reference data. Once down loaded and stored within the SQL:1999 DBMS, the database objects are immediately operational regardless of the DBMS brand, operating systems type, hardware vendor, or 3rd or 4th generation language tools that access and manipulate the newly installed database objects.

Because of ISO and ANSI standards, database objects operate consistently regardless of the SQL DBMS, operating system, and vagaries of the different presentation layer facilities. Enterprises are able to then have centralized semantics that control the fundamental operations of the business objects that are essential to world wide, heterogeneous computing environments.

Finally, database objects are independent from presentation-layer and operating-system bindings. This enables use of local language conventions within the confines of standard policy essentials. For example, regardless of their local abbreviations and local names, there are only two sexes. Additionally, local vendors may provide their own presentation layer facilities, report writers, formats, paper sizes, screen formats, and the like. Given database objects, these localized

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

peculiarities can be accommodated. And, because the database object environment is DBMS based, additional and localized database objects can be easily created and deployed with automatic integration into the standard database object environments essential for effective world-wide, heterogeneous environments. In summary, database objects:

- Are easy to specify, implement, use and maintain
- Can operate on world-wide, heterogeneous hardware and operating system environments, and
- Can behave consistently regardless of their host computing hardware environment.

1.7 Genesis of ANSI H2 Database Objects

At an ANSI H2 meeting in 1991, a member presented a paper that when decided clearly set database objects apart from process or business objects. Process or business objects are defined first and foremost as self contained processing units within the context of a programming language. Once they proceed through the necessary stages to become processing units they can stand-alone. At the H2 meeting the key question was whether a database would store objects that were defined and created by an object-oriented programming language like OO-COBOL, C, or C++, or whether objects are defined by and stored within a database such that it is accessible and employed by different programming languages such as OO-COBOL, C, or C++.

Data processing as seen through the evolution of computing languages shows that data has always been a self-contained component. For example, while data is both an important component of both COBOL and FORTRAN, each programming language treats data differently. The differences start with fundamental data types, definable data structures, and techniques for access. This fundamental difference between the languages caused the great data redundancy and semantics mismatch of the mid 1960s through the mid 1980s.

When ANSI database standards were started in 1978[6], one of its most critical objectives was to separate data structure definition, loading, update, deletion, and protection from the computing languages that access and use the data. With advent of ANSI database language standards, data and process were formally defined as separate.

Possibly in reaction to separateness or possibly as a natural evolution to programming languages, objects arose. H2, rather than allow objects to grow--differently--within the very different computing languages such as COBOL and C++, decided right from the very beginning to develop database language extensions to fully encompass objects. Hence, database objects.

[6] ANSI's committee, X3 (Computers and Information Systems), created a technical committee, X3H2, now just known as H2, to standardize languages for persistent data definition, loading, data update, access, and protection. The history of ANSI database standards is provided in the Whitemarsh book, *ANSI Database Standards*.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Because database objects are defined within the realm of database management systems rather than a programming language, the descriptive characteristics of objects are seen differently.

There are several key differences between business/process objects and database objects. Business/process objects stand and operate alone in an acceptable computing environment once they proceed through a compile and execution unit creation stage. Database objects, in contrast, require the continued existence of the database management system. The business and process unit contains all its parts (data, methods, etc.).

Business/process objects are commonly not accessible through various programming languages. Rather they are captive within the language through which they are developed. Further, if stored in a database, they are seen as BLOBs (binary large objects). Hospitable computing environments such as Microsoft's Window's 95 may allow information to be presented to or obtained from process objects because of pre-defined conventions. The meaning of the business/process object is however fixed and is not determinable.

Database objects are completely defined through syntax. Once defined, the syntax is able to be read and then acted upon by any ANSI SQL:1999 DBMS, regardless of vendor and computing platform. The syntax of database objects is stored in SQL schema information tables. Once schema information tables are loaded with database object syntax, the DBMS can then use the database object's semantics to enter, store, and manipulate store the data component of database objects. The database object processes and the database object information systems are automatically invoked and operate on the entered, stored, or manipulated data in support of the database object's transformation from one value state to another.

Because database objects are represented as syntax that is stored in schema information tables, end-users through their programming language agents are able to request that databases determine the location, nature, and characteristics of database objects. Requests can thus be launched to find the complete set of employee-candidate database objects. As the request is processed through the network of possible database object locations, the schema information tables can be accessed to first determine if employee database objects exist. If they do, then it can be determined whether there are employees in the employee-candidate state. Once found, the employee-candidate object themselves can be retrieved and copies returned to the requesting language agent. The reason this two part query process is possible is because database objects exist in two forms: type and instance, where the type is represented through metadata (the full syntax of its four parts) and the instances are represented by the actual database objects.

Because of this dual existence paradigm (type and instance), programming language agents can query database object schema information tables as to the state of an object. If a database object is known to exist a request can be launched to both find it and to return its state name.

Programming language agents can be created to dynamically format screens (the characteristics of business information systems) based on the required characteristics of the next state of a database object.

Requests can be made to properly report a set of database objects that share a common characteristic. For example, the set of all female persons within a personnel database might produce some that are employee candidates, new hires, are eligible for reviews and/or promotions, or are retired. Each database object existence would have to be found, its state

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

determined, and then its metadata consulted so as to determine the type, kind, and format of the data to be reported.

Finally, a request can be launched to give an employee a promotion. That request would determine not only the current state of the employee, but also next allowed state. If the promotion state is allowed a change would be instigated. If not allowed, the database object change to the promotion state would be disallowed.

While the differences between the business/process objects and database objects are significant indeed, each type of object has its unique role, characteristics and important uses. Not only can all three types exist, they must co-exist to fully realize the full benefits of objects.

Unique to database objects is the type-instance paradigm that is absolutely essential for businesses to achieve heterogeneous, world-wide enterprise database. Businesses have been attempting database objects for at least thirty years. Only now through ANSI standard SQL:1999 are database objects now possible. Database objects represent a technology independent, vendor, computing platform and operating system independent marriage between data and process semantics.

1.8 Centralized Versus Decentralized Data and Process Semantics

Among the very first reasons for commercial DBMS (database management systems such as IDMS, IMS, Total, IDS, and DMS-1100) was to preserve the investment in COBOL programs by removing the data definition from the program to another persistent object called the DBMS's data dictionary. As DBMS grew in popularity, the quantity of out-of-control files was replaced with the same quantity of out-of-control databases.

Files containing data that were defined through an ad hoc semantics definition process were replaced with databases defined through the same ad hoc semantics definition process. Simply, nothing had improved.

Corporations attempting enterprise-wide semantics must have a single organizational unit that defines or coordinates enterprise database object semantics. Figure 1.1 provides a critical set of business questions and the sets of answers that are possible when semantics and data are either centralized or decentralized. Even when this figure was first presented in the Whitemarsh course, *Managing Database: Four Critical Factors* in 1981, it was quite clear that corporations were headed towards distributed database.

During the next ten years (1982-1992), the entire data processing industry was turned on its head three different times: first with mini-computers, then PCs, client/server, and now the Internet. Mainframes that occupied large rooms were replaced with more powerful computers that occupied the space of a single desk. In 1981, the cost ratio of computer to staff year was 26 to 1. Today, the ratio has changed to about 1 to 6. That's a ratio change of 156 times. Because of this tremendous ratio change, not only are computers everywhere, there is also real demand to reduce staff costs.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

In the book, *Enterprise Database in a Client/Server Environment*[7], Figure 1.1 was supplemented with Figure 1.2 that cited the same questions and contrasted them to the answers for centralized versus decentralized development and execution control. It was very clear that without centralized control over process definition that enterprise-wide client/server database was absolutely impossible.

Questions regarding data distribution effects	Semantic Control			
	Centralized		Decentralized	
	Data Storage Control			
	Centralized	Decentralized	Centralized	Decentralized
Is data able to be shared among sites?	yes	yes	no	no
Is concurrent processing of the same data possible?	yes	maybe	no	no
Are common or corporate reports possible?	yes	yes	no	no
Can there be an overbearing "big brother" feeling?	yes	maybe	no	no
Is there local control and ownership?	no	maybe	yes	yes
Does there need to be common data standards & policies?	yes	yes	no	no
Can local data requirements be satisfied?	maybe	yes	maybe	yes

Figure 1.1. Centralization and Decentralization of Semantics and Data

Since 1994, two events have occurred that now force centralized definition of database objects to have any hope for enterprise-wide database.

- A dramatic increase in different programming language development environments
- The ANSI SQL call level interface (CLI)

Organizations are now able to use SQL based DBMSs through program development environments that are neither controlled by nor owned by the DBMS vendors. Prior to 1994, end users developed database applications using 3GLs such as COBOL and embedded access

[7] The book, *Enterprise Database in a Client/Server Environment* was published by John Wiley & Sons in 1994. This book is now available through Whitemarsh Press.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

commands to the DBMS, or through the DBMS vendors own self-contained 4GL facilities such as Oracle's Forms. In these environments, central definition control was possible because either the database's semantics were centrally defined in the 3GL programs, or they were commonly accessed through the DBMS vendors' self-contained 4GL procedures.

Questions regarding system distribution effects	Development Control			
	Centralized		Decentralized	
	Execution Location			
	Centralized	Decentralized	Centralized	Decentralized
Is the same program able to be shared among sites?	yes	yes	no	no
Is concurrent processing of the same data possible?	yes	maybe	no	no
Are common or corporate reports guaranteed?	yes	maybe	no	no
Can there be an overbearing "big brother" feeling?	yes	maybe	no	no
Is there local control and ownership?	no	maybe	yes	yes
Does there need to be common processing standards & practices?	yes	yes	no	no
Can local processing requirements be satisfied?	maybe	yes	maybe	yes

Figure 1.2. Centralization and Decentralization of Semantics and Systems

Since 1994, however, through Microsoft's ODBC[8] (open database connectivity) drivers, the DBMS engine has become completely independent from the program development environment. Organizations can adopt multiple end-user program development environments such as Oracle's Forms, Sybase's Power-Soft, Information Builder's Focus, SoftVelocity Corporation's Clarion for Windows, Borland's Delphi, or Microsoft's Visual Basic. Through any of these different program environments, databases under the control of any SQL vendor can be accessed and updated. While this may seem to be a great increase in development and deployment flexibility, it is a complete disaster for enterprise-wide business semantics because there is no longer a central design, development, and deployment authority for business semantics.

[8] An ODBC driver enables a 4GL to invoke calls to an SQL engine through reinterpretations by the ODBC driver to call a specific SQL engine. Thus, any 4GL can be a data manipulation language agent for any SQL engine. The ODBC is thus a universal translator.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Enterprise database environments that consisted of a relatively small set of well controlled main-frames through which business semantics[9] were centrally defined and controlled no longer exist. In their place are from hundreds through tens-of thousands of clients and servers. Because of the great spread of clients and servers, columns 2 from both Figures 1.1 and 1.2 can only be achieved by:

- Forcing all end-users to employ the same combination of DBMS, computer brand, operating system and program development environment, or
- Having a highly educated business subject matter expert data processing staff for each unique combination of DBMS, computer brand, operating system and program development environment

Both these alternatives are both impractical and politically impossible for world-wide, heterogenous organizations

To regain control over business semantics, enterprises must have business semantics control inside the ANSI SQL DBMS fire-wall. This can be done through database objects. Through database objects, the business' semantics can be centrally defined and controlled through the ANSI SQL engine and their consistent use is independent of any program development environment. Because of database objects, end-user business information systems programmed in the program development environments cited above that access business databases will always receive the same behavior, and be subject to the same business rules.

The database object development environment is accomplished through database object oriented systems analysis and design activities that result in a series of ANSI SQL syntax statements that fully define the database objects. The defined database object language streams are portable from one ANSI SQL DBMS to another and are activated through the database's schema creation process.

As a consequence of ANSI SQL database objects, Figure 1.3, which is an amalgamation of Figures 1.1 and 1.2, results. An enterprise database environment in which semantic control is extended only to data (Figure 1.1) will never achieve enterprise database. An enterprise database environment in which semantic control is extended to process--but separately--will also never achieve enterprise database. Simply put, semantic control must be accomplished in a lock step manner for all four components of database objects (data structures, processes, information systems, and state) for organizations to have a chance at enterprise database success. Thus, Figure 1.3 is not just an update to Figures 1.1 and 1.2, it is an essential replacement.

Database objects can only have a positive effect on the computer to staff cost ratio because a relatively small quantity of business subject matter expert staffs, supplemented with a ANSI SQL database object construction experts, can now define the business's database objects, once and for all. Lesser business subject matter expert staffs can be employed or contracted to use the program development environment "de jour" to obtain, store, and maintain business's

[9] The business semantics were defined through closely matched pairs of DBMS data definition language (data structures and integrity constraints) and 3GLs or vendor contained 4GLs (transactions, computation, and database access logic).

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

database object instances according to the particular style and computing environments of the individual world-wide business units.

Questions regarding database object distribution effects	Semantic Control			
	Centralized		Decentralized	
	Development Control			
	Centralized	Decentralized	Centralized	Decentralized
Are database objects able to be shared among sites?	yes	yes	no	no
Is concurrent processing of the same database object instance possible?	yes	maybe	no	no
Are common or corporate reports possible?	yes	yes	no	no
Can there be an overbearing "big brother" feeling?	yes	maybe	no	no
Is there local control and ownership?	no	maybe	yes	yes
Does there need to be common data standards & policies?	yes	yes	no	no
Can local data requirements be satisfied?	maybe	yes	maybe	yes

Figure 1.3. Centralization and Decentralization of Semantics and Database Objects.

1.9 Contents of the Book

The full exposition of enterprise database, that is, database objects, requires presentation through all three levels of abstraction (foundation, specification, and application)[10]. This book therefore presents enterprise database (database objects, database object application examples, and meta database objects) through the following chapters:

- 2.0, Database Management Systems, which presents the evolution of database management systems (DBMS), the four DBMS data models, their affect on the definition and use of database objects, and the actions of the database languages

[10]. These three levels of abstraction were employed during the development of the ANSI IRDS. Each level, foundation then specification then application existed in two forms: type and an instance. The foundation had a fundamental type for which there were three instances: entity, attribute, and relationship. These instances then became types on the specification level. Specification level types have instances (e.g., employee for entity, birth date for attribute, and contains for relationship). These, in turn, become types on the application level. Application level types have instances (e.g., Name: Alan Goldfine, Birth date: 04/25/43, Organization: NIST).

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

committee (H2) to build an ANSI database language specification to support business centered data.

- 3.0, Successful Computing Environments, which provides a brief history of the evolution of computer languages, information systems, hardware and operating systems, and peopleware along with their effect on the requirements of modern enterprise wide information systems.

- 4.0, Database Objects, which provides a comprehensive definition of database objects

- 5.0, Transportation Public Safety Database Objects, which provides a comprehensive example of the use of database objects to frame a state-wide set of database system specifications for public safety data collection, mid and upper level analysis and reporting.

- 6.0, Courts Database Objects, which provides a comprehensive example of the use of database objects that embrace the data and processing needs for all the different types of courts throughout a state.

- 7.0, Sales and Marketing Database Objects, which provides an example of the use of database objects to define, and implement sales and marketing database to manage product sales, distribution, sales organization management, and sales reporting and planning.

- 8.0, Database Object Summary, which provides a summary of the requirements for a complete database object environment.

- 9.0, Methodology Support, which provides a work break down structure for identification, specification, and implementation of database objects.

- 10.0, Metadata repository Support, which provides an overview of the metadata model, and metadata repository processes necessary to create, store, update, and report database objects.

- 11.0, Database Object Summary, a summary of the goals, objectives, and road map to achieving database objects is world-wide enterprises that must rely on heterogeneous environments of hardware, software, languages and cultures.

The Whitemarsh metabase, a CASE/Repository system fully supports the storage and manipulation of database object classes. Documentation of the metabase if available from the Whitemarsh website, www.wiscorp.com.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2

DATABASE MANAGEMENT SYSTEMS

In general, a database management system (DBMS) is a computer based system that employs languages to:

- Define data structures
- Load data according to the data structures
- Select, format and report data that has been stored
- Update and delete data that has been stored
- Protect stored data from unallowed access

Database management systems have existed for about 50 years. They were created to satisfy certain classes of persistent data problems, thus DBMSs are broadly grouped into the following classes:

- The National Defense Data Management Systems
- The Commercial Database Management Systems
- Independent Logical File DBMS
- Relational Database Management Systems
- ANSI Database Management Systems

2.1 The National Defense Data Management Systems

Database started in middle 1950s. The Korean War had just ended; the cold war was raging. Large scale strategic information systems were critical to national defense.

Database from the very beginning has always been a vehicle for representing complex data structures. Complex means hierarchies and networks. By the early 1960s, the first DMS (data management system), ADAM (Advanced Data Management), was running in DoD "think tanks" (MITRE, Lincoln Labs, and the System Development Division of the Rand Corporation). These DMS had three characteristics in common: English query language, hierarchical data structures, and full-inversion (value then context) access. In some DMSs the only data that really existed outside the indexes and relationships were the non-indexed fields. Data was directly accessed through disk drives and drums.

By the middle 1960s, DMS were fully operational, supporting command and control databases for the Strategic Air Command, the Defense Early Warning System (DEWS), and Air Battle control in the U.S. and other Far East locations.

The word *database* arose by the late 1960s. A database is a large store of data that conforms to predefined data structures that can be accessed through operations on those structures. It was common for both the data structures and the operations to match a clearly defined set of corporate policies. The operations that existed included, in today's terminology, column and table constraints, stored procedures, assertions, triggers, and a full complement of

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

referential integrity rules and actions. One DMS, now called database management systems (DBMS), TRW's GIM, included fully definable abstract data types, a form of distributed database and two phase commit. All these DMSs had schema manipulation languages and full security and privacy. The final component common to these DMSs was no access through third generation languages.

System 2000, the most successful of the DoD line of DBMS was very popular in the U.S. Federal Government. Most Federal agencies such as USDA, HUD, Commerce, the Army, Navy, and Air Force used System 2000 to implement large-scale management information systems.

The database designs presented in Figure 2.1 (2.1a through 2.1f) that are at the end of this chapter illustrate typical System 2000 databases. These design were created in 1971 for the Dallas, Texas Board of Education to manage its educational programs through a Planning and Management Information System (PMIS). These database designs provide the necessary information for education researchers and planners as they explore the changing demographics of communities and the various funding sources and use restrictions. Given today's terminology, these database designs were clearly wholesale data warehouses[11]. Dallas had original data collection databases that served the day to day operations of the school district. Lacking was the ability to perform longitudinal research into students, facilities, faculty, and educational programs. The PMIS databases served that need.

The PMIS databases (Students, Student Support (School Characteristics, Programs, Facilities, Staff), and School Year) were more than just historical. To illustrate the advance in data structures, each System 2000 database can now be defined within the data structure of a single SQL:1999 table.

The Student database, presented in Figure 2.1a contains seventy components (1* Student Number through 70* Grade/Achievement Level). A component is a column that either single value (e.g., 1* Student Number), or a repeating group (e.g., 6* Federal Property Residence). Components within the Federal Property Residence repeating group (7* Name of Parent through 14* Serial Number) represent multiple occurrences of Federal Property Residence information. In essence, Federal Property Residence is a nested table. As can be seen from components 53* Student Record, and 59* Courses/Activities/Studies, repeating groups can contain repeating groups. This same type of capability is available in SQL:1999.

---

[11] A wholesale data warehouse is one of five different architecture classes of databases. The first is original data capture and is commonly designed to optimize the capture and updating of original data. The scope of these databases is restricted to a single application. The second architecture class, transaction data staging area (TDSA) exists to meld the different semantics of various original data capture databases. TDSA databases are necessary because the original data capture databases can operate under different DBMSs, operating systems, and be from different application vendors such as PeopleSoft. TDSA data is commonly unindexed and remains on the TDSA servers for a week or less. The third class of database, Subject Area Databases (sometimes also called Operational Data Stores (ODS)) meld TDSA data from multiple application areas into a single subject area such as Human Resources or Finance. These database applications are commonly custom built. The fourth database architecture class are the data warehouses. There are two types: wholesale and retail (now called Data Marts). Both classes are read-only and may contain data from multiple subject area databases. The fifth data architecture class databases are those that supply commonly used reference data that provide the definitive meanings for various codes, site names, Email addresses, and the like.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The Student Support database contains four "tables." The School Characteristics table, presented in Figure 2.1b, contains components 100* School Characteristics through 128* Dropout Rate. This "table" represents characteristics for each school for different years. This "table" contains nested tables, 109* Community Programs, 112 * Major Disruptions, 115* Grades in Schools. The 115* Grades in Schools nested table also contains additional nested tables, 118* Ethnic Groups, 121* Target Groups within 118* Ethnic Groups.

The Programs "table," presented in Figure 2.1c, contains a number of nested table repeating groups that allow researchers to select and compare different programs by school, year, school grade participation, student characteristics within grades, and the like.

The Facilities "table," presented in Figure 2.1d, contains a number of nested tables that enable researchers to understand the types of buildings, owners, additions, functional areas, classrooms, utilizations, and types and sizes of rooms.

The Staff "table," presented in Figure 2.1e, contains an abbreviated personnel record for various staff within the school system. Nested tables address education, degrees, areas of study, certifications, and nested subtables for the employee record including annual report, assignments, and in-service education.

The final database, District Characteristics, presented in Figure 2.1f, contains information about the overall population of the Dallas Independent School district including nested tables that represent the characteristics of the population, residence types, and various types of revenue sources including fund use restrictions and the programs that may be funded.

The System 2000 query language was very rich and was able to select, sort and retrieve data in various formats. The DBMS performed "automatic" joins across all nested tables. Because the System 2000 data model is hierarchical, it was not able to join across hierarchies. Accomplishing that required data selection and dynamic new database creation according to newly created hierarchies.

What SQL:1999 provides is the ability to join across hierarchies of the nested tables. For example, SQL:1999 can both accomplish the System 2000 nested structures and can also perform joins such as:

```
JOIN 200* PROGRAM NUMBER IN PROGRAMS
        WITH
38* PROGRAM NUMBER IN DISTRICT CHARACTERISTICS
```

Note: this is not really SQL syntax, of course.

This join would make available program information for reporting about programs that are subject to fund use restrictions.

2.2 The Commercial Database Management Systems

By the end of the late 1960s, another class of DBMSs, started. These were created to preserve the millions of lines of COBOL code. These DBMS started with IBM's FFS, formatted file systems, and ended up with two systems, IMS from IBM, and the set of CODASYL (COmmittee on DAta SYstems and Languages) systems, headed by IDMS, a project of the B.F. Goodrich

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

company. Included also within the CODASYL set were IDS from Honeywell and DMS-1100 from Univac. Common to those systems were data structures (hierarchical and network), COBOL (Common Business Oriented Language) language access through a subschema facility, and record processing access strategies (that is, almost no indexes). CINCOM's Total was a simplified network data model DBMS designed to compete with IBM's IMS. Total was the first DBMS to run on almost every brand of hardware, a sort of de facto DBMS standard.

The CODASYL DBMSs have column and table constraints, a full complement of referential integrity actions and rules, and stored procedures that are activated as triggers.

In the 1970s, databases were still considered to be large complex sets of data structures that matched some coherent set of corporate policy. Database design was laborious and time consuming, not just because of the physical database reorganization requirements from computers that were not even capable of 1 transaction per second, but because the databases were to reflect corporate policy. Policy analysis is, and always has been difficult, time consuming, and something not done in an ad hoc, off-hand manner.

Database design was firmly and squarely based on business policy. During the 1970s, database designs were not considered as being database objects. In hindsight, they were (at least the data structures portion of a database object). There were databases for EMPLOYEEs, PROJECTs, TEACHERs, COURSEs, STUDENTs, FACILITIES, FINANCEs, etc. Each database was founded on a well known, coherent business concept. As each database object, that is, the employee, project, teacher, course, and student moved through well known states, certain data structures were valued, modified, and/or deleted. To accomplish these state transformations, whole information systems were created. These information systems either employed their own database transformation logic or used the DBMS self-contained database processes to accomplish the state transformations.

The ANSI NDL database structures allowed only one level of nesting. SQL:1999 can however contain multiple levels of nesting. ANSI/NDL has explicitly named and defined relationships that govern load/update based relationships. The actual addresses of the related records are computed and stored as the data records are loaded or updated. In addition to these explicitly defined relationships, sort orders are able to be specified so that the records that participate in the relationship are retrieved in the order specified.

SQL:1999 has richer data structures than ANSI/NDL. SQL:1999 can also dynamically specify relationships across data with common data types. To maintain semantic and business rule consistency database administrators establish views that join columns that make “business sense.” Users are then given access only to the SQL views.

2.3 Independent Logical File DBMS

A third set of DBMSs arose by the late 1960s. These DBMSs made traditional file access easier through the standard 3GL. Adabas from Germany and Datacom/DB from Dallas, Texas exemplified these systems. If an organization had data files, then after a quick and easy data loading operation it had database. The common expression was "Database in a Day!" These DBMSs, typified by Adabas, Datacom/DB Model 204, Focus, RAMIS, Nomad, and Inquire allow several levels of data structure nesting; the operations of select, project, divide, join, etc. RAMIS, FOCUS, and Nomad have a full complement of joins and unions.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

These systems adhere to the independent logical file (ILF) data model and were executing production databases well before 1970. ILF systems are characterized by shallow data structures and index based or sequential "file" access through 3GLs and their own self contained languages (today's 4GLs). These DBMSs however did not completely model database objects as completely as the previous DBMS classes did.

FOCUS from Information Builders in New York City is a very popular ILF data model DBMS. Figure 2.1g (also at the end of this chapter) contains the database structure for EMPLOYEE. Each SEGNAME and its contained FIELDNAMEs is equivalent to a table. In the SEGNAME line the subclause, PARENT identifies the segment that is the parent. For example, the parent of FUNDTRAN is EMPINFO. FUNDTRAN thus behaves like an SQL subtable. FOCUS can have multiple levels of nesting. The segment, DEDUCT has SALINFO as its parent. SALINFO's parent is EMPINFO.

Like SQL:1999 DBMSs, FOCUS and other ILF DBMSs can dynamically specify JOINs. In FOCUS, the clause is specified similar to the JOIN cited above.

2.4 Relational Database Management Systems

The first technical papers that define the foundation of relational database management systems were published in 1970. The foundation of relational database was similar to independent logical file DBMS except that relational database structures were restricted to just two dimensional data. The foundation documents also defined DBMS vendor independent specification for operations on these two dimensional data structures. The relational data model specifications did not contain:

- Formally defined data structures such as vectors, groups, and repeating groups
- DBMS invocable stored procedures
- Sophisticated integrity for columns and tables
- The ability to explicitly define common business relationships

Simply stated, by the time relational was discovered in 1970, every viable DBMS concept was already in some production class DBMS of one type or another. Regrettably, what relational did however, was to cause two very valuable concepts/activities: database design, and business policy based database objects to be abandoned.

During the 1970s and through the end of the 1980s, databases sprang up like weeds. The awards seemed to go to the one who could build the largest, most number of tables and rows database; who could push through the most number of rows/transactions in a second; who could have the most number of columns in a table, and the like. All the while, business policy analysis, coherent database design, etc, fell by the wayside.

2.5 SQL:1999 and SQL:2003 Data Models

Starting in 1992, the H2 committee began the development of dramatic extensions to the SQL:1992 standard. The greatest change in the standard is that it no longer adheres to the 1970

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

relational data model. The second biggest change is that the SQL:1999 language now consists of individual parts that comprise a foundation and then a series of independently specified packages. The remainder of this section provides an overview, in outline form, of the contents of the SQL:1999 standard.

2.5.1 Foundation Components

- Tables that have been enhanced to support new built-in data types (boolean, enumerated, extensions to character sets, translations, and collations)
- BLOB and CLOB data types
- Abstract Data Types (user defined data type with behavior, an encapsulated internal structure, and access characteristics of public, protected, or private)
  - strong typing
  - subtypes and inheritance
  - encapsulation
  - virtual attributes
  - substitutability
  - polymorphic routines
  - dynamic binding
  - compile time type checking
  - value ADTs
- Array
- Row Types (table person (SSN, name(first, middle, last), address(street, city, state, zip(four, five)))))
- User Defined Functions
- Predicate extensions (for all, for some, similar to, cursor extensions, null values, assertions, view updatability, joins)
- Triggers
  - Different triggering events, update, delete, and insert
  - Optional condition
  - Activation time: before and after
  - Multiple statement action
  - Several triggers per table
  - User-defined ordering
  - Condition and multiple statement action per each row or per statement

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Roles (enhancements to security), & Savepoints

- Recursion

2.5.2 Call Level Interface

The SQL Call Level Interface is the set of language specifications used by DBMS vendors to enable direct SQL engine access through completely specified call routines. Microsoft, for example has implemented SQL/CLI and calls it ODBC.

The CLI specification contains more than 50 different call specifications that address:

- Connection control to SQL servers
- Allocate and de-allocate resource
- Execute SQL statements
- Obtain diagnostic information
- Control transaction termination
- Obtain information about implementation

It also contains Resource Management Handle routines for:

- Environment
- Connection
- Statement
- Context

The CLI also contains a Descriptor Area that accommodates:

- Application parameter
- Application row
- Implementation parameter
- Implementation row

2.5.3 SQL/Multi Media (MM) Components

SQL/MM is itself a set of subparts that contain full specifications for a discrete set of data management functionality to address the data processing needs of:

- Full Text
- Spatial
- General Purpose
- Still Image

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.5.4 SQL Persistent Stored Module Language Components

SQL:1999 now has a complete embedded programming language to support the processing needs of its user defined data types, assertions, and triggers. The SQL/PSM language supports the following capabilities:

- Call
- Return
- Compound Statements (Begin ... End)
- If Statements
- Case Statements
- Loop
- Repeat
- While
- For
- Leave
- Assignment
- Signal and Resignal

2.5.5 SQL Transaction and Connection Management

Since the advent of distributed processing, client-server, and of course the Internet, the ability to manage transactions is critical. SQL:1999 now has facilities that address the following areas of transaction and connection management:

- Start transaction
- Set transaction
- Test completion
- Savepoint
- Release savepoint
- Commit
- Rollback
- Connect
- Set connection
- Disconnect statement

2.2.6 ANSI Standard SQL Data Model

SQL:1999 is not just a simple language for defining, accessing and managing tables consisting only of single valued columns of data. With respect to the basic data model capabilities, the SQL language more closely supports the independent logical file data model from the 1960s. It is therefore true to say that SQL standards, starting from SQL:1999 is more of an implementation

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

of the independent logical file data model (e.g., Adabas, Inquire, Datacom/DB, and Sybase) than of the 1970 relational data model.

SQL:1999 has, however, gone way beyond the capabilities of the independent logical file data model by incorporating facilities such as user-defined types, embedded programming language, and libraries of SQL:1999 defined routines for areas like full text management and spatial data. To say that these SQL:1999 extensions are mere “extended interpretations” of the relational data model is like saying that an intercontinental ballistic missile is merely an “extended interpretation” of a spear.

SQL:1999's impact on network and hierarchical data model DBMSs is significant. Network data model DBMSs have traditionally allowed complex table structures with arrays, groups, repeating groups and nested repeating groups. A very unique characteristic of the SQL:1999 data model is that it now allows arrays. In addition, the columns of the array are able to be outward references to other data. Since the order of the columns in an SQL:1999 array is maintained by the SQL:1999 DBMS, then the array, with its outward references, is essentially a CODASYL set. This is a dramatic departure from the relational data model.

The only remaining and viable network DBMSs are IDMS by Computer Associates and Oracle DBMS (formerly the VAX DBMS). Both have had an SQL language interface for about 10 years. How Computer Associates plans to take advantage of the existing IDMS facilities with SQL:1999 is not known. A significant customer of Oracle’s DBMS (formerly Vax DBMS from DEC) is Intel who uses the Consilium manufacturing package to manage computer chip manufacturing. How the Oracle Corporation plans to take advantage of the existing VAX DBMS facilities with SQL:1999 is also not known

The only two hierarchical DBMSs, System 2000 and IBM’s IMS will likely not be impacted at all. System 2000 is no longer being advanced by SAS, and IBM has a full implementation of DB/2 on many different operating systems.

SQL:1999's impact on independent logical file DBMSs, for example, Adabas, Focus, and Datacom/DB is significant. These DBMSs already support many of the SQL:1999 data model facilities. It would seem that these DBMSs could rapidly conform to the new SQL:1999 standard. If these vendors embrace the SQL:1999 model, then these DBMSs could claim conformance sooner than the existing set of relational DBMSs.

Simply stated, the SQL:1999 language defines a unique data model. It contains:

- The ability to model CODASYL sets,
- Many of the natural data clustering features of the hierarchical data model,
- Explicit many-to-many and inferential relationships like the independent logical file data model, and finally,
- The unique ability to directly model recursive relationships.

It therefore can only be said that the SQL:1999 data model is unique unto itself. Clearly, it is not the relational data model, CODASYL network, hierarchical, or independent logical file data models. Simply, SQL:1999 is a data model unto itself.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The figures that follow show the SQL:1999 and beyond (that is, the SQL:2003) data model in terms of its:

- Data Structures
- Relationships
- Operations

These are then compared to the facilities that exist in the network, hierarchical, and independent logical file data models.

2.2.6.1 Data Model: Data Structures

Column Data Type	Definition	SQL:1999
Single Value	Each component represents a single value such as Birthdate with the value 11/11/1987	✔
Multi-value	Each component represents multiple values such as Nicknames with values "Buddy, Guy, Mac"	✔-Note 1
Groups	Each component has subcomponents to represent single-set of values such as Address with Street-1, Street-2, City, State, Zip	✔-Note 2
Repeating Groups	Each component has subcomponents to represent multi-sets of values such as Dependents that contains subcomponents, Dependent Name, Dependent Birth date, Dependent SSN.	✔-Note 3
Nested Repeating Groups	Employee (Dependents (Hobbies))	✔-Note 4

Notes:

1 Arrays as a data type for a column
2 ROW data structure of a column
3 ROW data structure for a column wherein each Table structure field has the data type, ARRAY
4 ROW data structure for a column with contained ARRAYs where each ARRAY column is a ROW data structure with contained ARRAYs where each ARRAY column , etc....

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.2.6.2 Data Model: Relationships Background

SQL 1999 Relationship Types

Name	Example	SQL:1999
One-to-many	Employee to dependents	✔-Note 1
Owner-multiple-member	Territory contains salesmen and customers	✔-Note 2
Singular-one-member	Top performing employees	✔-Note 1
Singular-multiple-member	Top performing current, former, part-time, and retired employees	✔-Note 3
Recursive	Organization contains organization	✔-Note 4
Many-to-many	Automobiles and owners	✔-Note 5
One-to-one	Table and its primary key	✔-Note 6
Inferential	Many houses each with a location, and then buyer with desired location	✔-Note 7

Notes:

1 Traditional relational data model (SQL 1986, 1989, & 1992)

2 Implemented as a ROW(TerritoryId, Salesman REF (SalesmanId),

3 Developed using Subtables where Employees are partitioned off into their common columns (employee) and their unique columns (current, former, part-time, and retired)

4 Recursion operations built into the language (WITH RECURSION...)

5 Cross joins from within columns of ARRAYS contained as data types of column in different tables

6 Effectively as tables and subtables. Most directly with UNIQUE Fkey

7 A single valued non-primary key Location within House and same for Buyer

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.2.6.3 Data Model: Operations Background

SQL 1999 Operations

- **Row Operations –** traditional insert, delete, and modify of rows and columns within rows

- **Relationship Operations** – traditional operations that affect the relationships between rows of the same or different tables

Row Operations			
Operation	**Static**	**Dynamic**	**SQL:1999**
Find	SELECT According to STORED Order	SELECT and PUT into DML Specified Order	✔
Get	Obtain Row From Find	Ditto	✔
Add	Install a New Row Into Database	Ditto	✔
Delete	Remove an Existing Row From Database	Ditto	✔
Modify	Change Some Column Values in Existing Row	Ditto	✔

Relationship Operations			
Operation	**Static**	**Dynamic**	**SQL:1999**
Connect	Add to a Named RELATIONSHIP in Specific Order	N/A	✔-Note 1
Disconnect	Delete From RELATIONSHIP	N/A	✔-Note 2
Get Owner	Obtains The Parent of the Row That is Current	N/A	No
Get Member	Obtains the First Child of the Owner For the Named Relationship	N/A	✔-Note 3
Get next	Obtains the Next Row Within The Named Relationship	N/A	✔-Note 4

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Intersect	N/A	Find and Keep Only the Common	✔
Difference	N/A	Find and Keep Only the Not Common	✔
Join	N/A	"Append" Relations to Each Other	✔
Divide	N/A	Subset	✔
Product	N/A	Cross-Product	✔
Union	N/A	Merge and Drop Duplicates	✔

SQL:1999 Relationship Operations(cont.)

Notes:

1	Value an Column in an ARRAY. Data type of Array is REF type pointing elsewhere.
2	Set value of column in array to null
3	Get the first column of an array. Data type of Array is REF type pointing elsewhere
4	Get next column in array. Data type of Array is REF type pointing elsewhere

2.6 DBMS Evolution Summary

The very first DBMSs in the middle 1960s supported sophisticated data structures. The "relational revolution" that started in the middle 1970s changed all that. Relational DBMSs, however, only supported two dimensional structures. Responsibility for defining business relationships among the data and for specifying the most optimum access path for database processing was transferred from the data administrator and the database administrator to the end user. Early DBMSs had simple data integrity checks that prevented character data in numeric fields. Referential integrity was automatic because of predefined structures. The early SQL based DBMSs did not support referential integrity nor any integrity constraints beyond the minimum.

By 1992, ANSI SQL had developed a standard that had full referential integrity and a reasonable quantity of table based integrity constraints. Starting in 1992, the ANSI H2 committee began development of SQL:1999 which now supports a fully contained DBMS-based

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

programming language, and very rich integrity constraints. In addition to enriching the relational model, the relational model itself was abandoned. It was changed from simple two-dimensional data structures to nested structures of tables similar to those contained in System 2000 and in most ILF DBMSs. SQL:1999 was additionally enhanced to support user defined data types well beyond the fundamental types. Because of all these changes, SQL DBMSs can now support more business data relationships than previous.

2.6 Business Data Relationships

The eight types of relationships among business data are:

- One-to-many
- Singular-one-member
- Singular-multiple-member
- Recursive
- Many-to-many
- One-to-one
- Inferential

Figure 2.2 tabulates these eight relationships against the four popular data models. The cells indicate that the relationships are accomplished either *direct* or *indirect*, *static* or *dynamic*, and finally, *no*. A *direct* accomplishment relationship is one that is explicitly accomplished through the identified tables. An example of a direct relationship is the owner and 1 member. The only tables that have to be created are the owner table (for example, employee) and member table (for example dependent).

Relationship Type	**DATA MODEL**			
	Data Definition Language Declaration		**Data Manipulation Language Simulation**	
	ANSI Network	**Hierarchy**	**Independent Logical File**	**Relational**
Owner, 1 member	Direct & static	Direct & static	Direct & dynamic	Direct & dynamic
Owner, multiple members	Direct & static	No	No	No
No owner and 1 member	Direct & static	No	No	No
No owner, multi-members	Direct & static	No	No	No
Recursive	Direct & static	Indirect & dynamic	Direct & dynamic	Indirect & dynamic
Many to many	Indirect & static	Indirect & dynamic	Direct & dynamic	Indirect & dynamic

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Relationship Type	DATA MODEL			
	Data Definition Language Declaration		Data Manipulation Language Simulation	
	ANSI Network	Hierarchy	Independent Logical File	Relational
Inferential	Indirect & static	No	Direct & dynamic	Direct & dynamic
One to one	Direct & static	Indirect & dynamic	Direct & dynamic	Direct & dynamic

Figure 2.2. Naturally occurring relationships among database tables.

An *indirect* accomplishment mechanism means that the relationship can be accomplished through the creation of additional tables. For example, the many-to-many relationship can only be directly accomplished through the independent logical file data model. The network and relational data models require the creation of third table that is co-owned by the two owners of the many-to-many relationship.

A *no* accomplishment mechanism means that there is no practical way for the relationship to be accomplished by the DBMS's data definition language or natural data manipulation language. Accomplishment requires assistance from a 3GL, or through a vendor proprietary extension of the data model.

A relationship is *statically* accomplished when the identifier (pointer or primary key value) of the member of the relationship is stored either in a separate relationship mechanism or is stored in the owner table row of the related tables. This can only be accomplished through the process of load of update because the owner table in the relationship (or the separate relationship mechanism) has to be updated whenever a member of the relationship is stored. After the load of the member, the relationship is "known" or statically bound into the owner table row (or into the separate relationship mechanism). Thus, the relationship is identified as *static*.

A relationship is *dynamically* accomplished when the identifier (pointer or primary key value) of the owner row of the relationship is stored in the member table row of the related tables. This is accomplished when the member row is stored. The owner table row is accessed only to assess referential integrity, which is entirely optional. Despite the loading of the owner identifier into the target member row, the actual relationship between the owner and its members is not known until a request is made from the owner table row for all the member table rows, or from the member table row for the owner table row. In either case, the relationship, as evidenced by the retrieved rows is neither known nor materialized until retrieval time. Because this is accomplished at the time of each and every retrieval, the relationship is identified as *dynamic*.

The most common relationship is *one-to-many*. An example of this relationship type is: INCIDENT has zero, one, or many EMERGENCY MEDICAL RESPONSE INCIDENTS. An example of a one-to-many relationship is depicted in Figure 2.3. For each row from the INCIDENT table there are zero, one, or more rows from the Emergency Medical Response Incident table. In the example depicted in Figure 2.3, there is a bus crash that causes emergency medical response from three different advance life support units, one each from North East, Aberdeen, and Harve deGrace. A single query obtains, for a particular incident all the responding emergency medical response incidents. If there are many different incidents types

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

related to an overall incident then there has to be as many different queries as there are different one-to-many relationship types.

Within an database object INCIDENT however, there are multiple incident types such as:

- Emergency Medical Response Incidents
- Vehicle Checkpoint Incidents
- Vehicle Accident Incidents
- Emergency/shock Trauma Medical Event Incident
- Vehicle Stop Incident
- Vehicle Inspection Incident and Judicial Case Incidents

While an incident containing all of these different incident types is not nice to contemplate[12], an interstate highway rear end crash that started with a vehicle checkpoint incident by a state trooper stopping and checking an over-the-road truck for proper load tie-down was the instigator of another truck rear-ending a *rubber-necker*. As the truck jack-knifed, twenty additional cars, four more tractor trailers and an over the road bus all piled up.

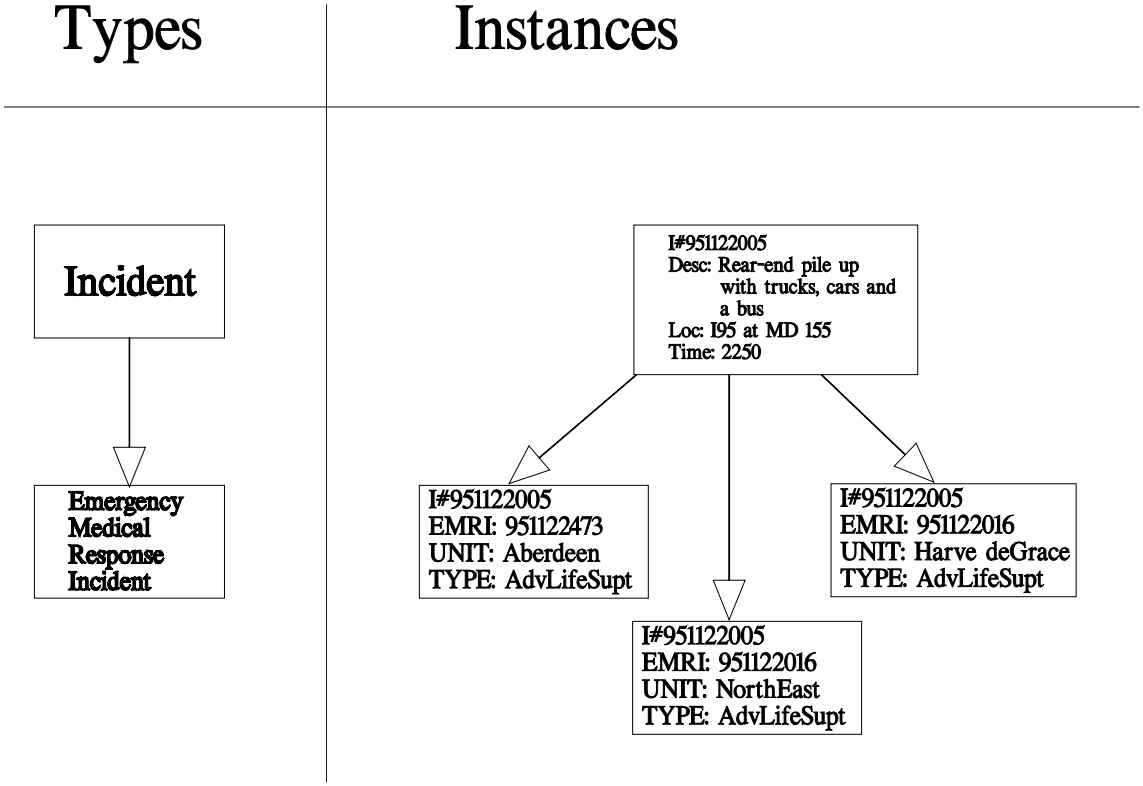


Figure 2.3. Data structure component of the database object: Incident.

[12] This example is clearly more realistic to the out dated and too simple, PARTS, SUPPLIERS, and SUPPLIER PARTS. After all, this is a book about database, not mathematical set operations.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

After all the people and property were transported and trucked away, there had to be an overall assessment of the entire incident. In this case, the ultimate goal was to have all related incidents accessible under one overall incident.

An *owner-multiple-member* relationship is a one-to-many relationship that has one owner table and multiple member-tables. This type of relationship is necessary when a single relationship is expressed across multiple types of diverse data. For example, in the crash described above, at least three different types of incidents were recorded:

- Shock trauma facility medical event incident to accomplish triage and to immediately treat the most critical survivors.
- State Police to direct traffic, block off Interstate 95 for the landing of the three Medivac helicopters, that ultimately went to Baltimore, Wilmington, and to the US Army hospital at the Aberdeen Proving Grounds.
- On-scene advance life support units to administer "golden hour" life support and to accomplish road transport of the non-life threatening injured.

Figure 2.4 illustrates use of this multiple-member relationship. One query can acquire all the different interrelated incident type instances.

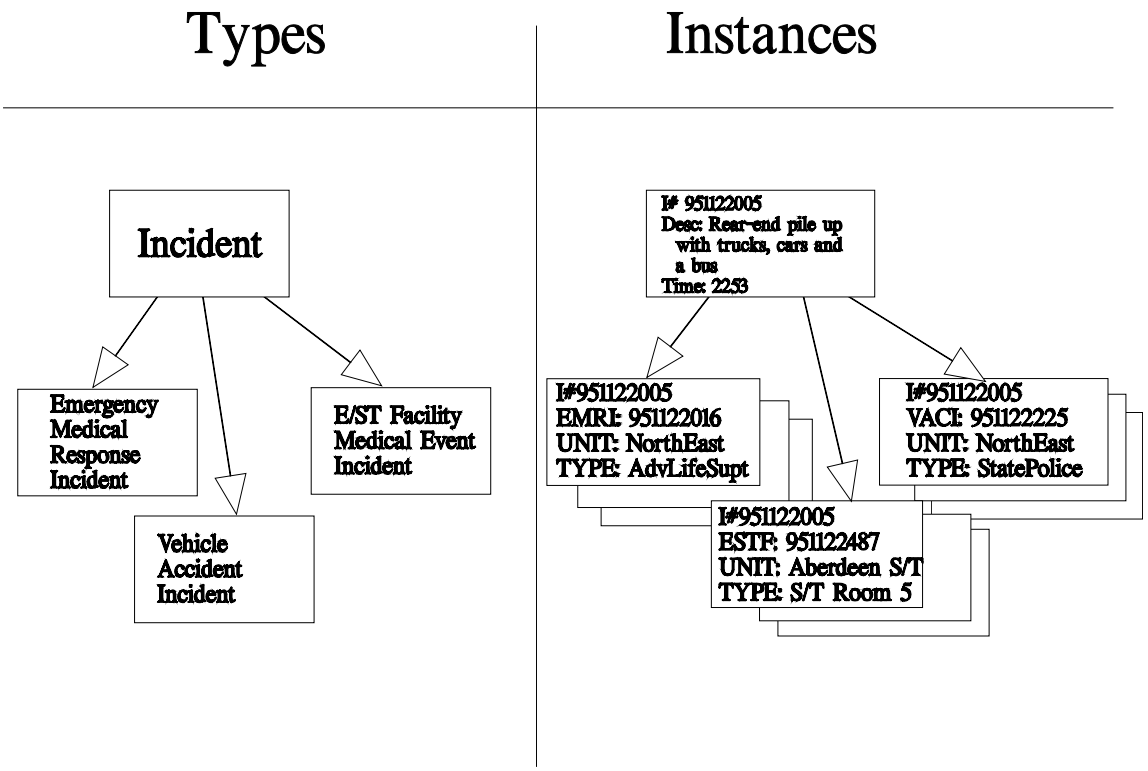


Figure 2.4. Multi-member relationship for database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

A *singular-single-member* relationship is so named because it contains only one table, the member. There is no owner table. The need for an ownerless relationship arises when it becomes necessary to relate rows from the same table without there being a naturally existing--different--owner table. In the crash incident above, all the emergency medical response incidents are classified with a code of red, yellow, or green. Figure 2.5 depicts 10 different emergency medical response incidents. Four are classified as red and these victims must be transported by Medivac if there is any hope of survival. Three are classified as yellow and these victims are the first transported by advanced life support road units. Three others are classified as green and these victims can be transported via basic life support road units. The relationship name is TRIAGE_CODE, and the value RED connects all those in the red status, YELLOW connects all those in the yellow status, and GREEN connects all those in the green status.

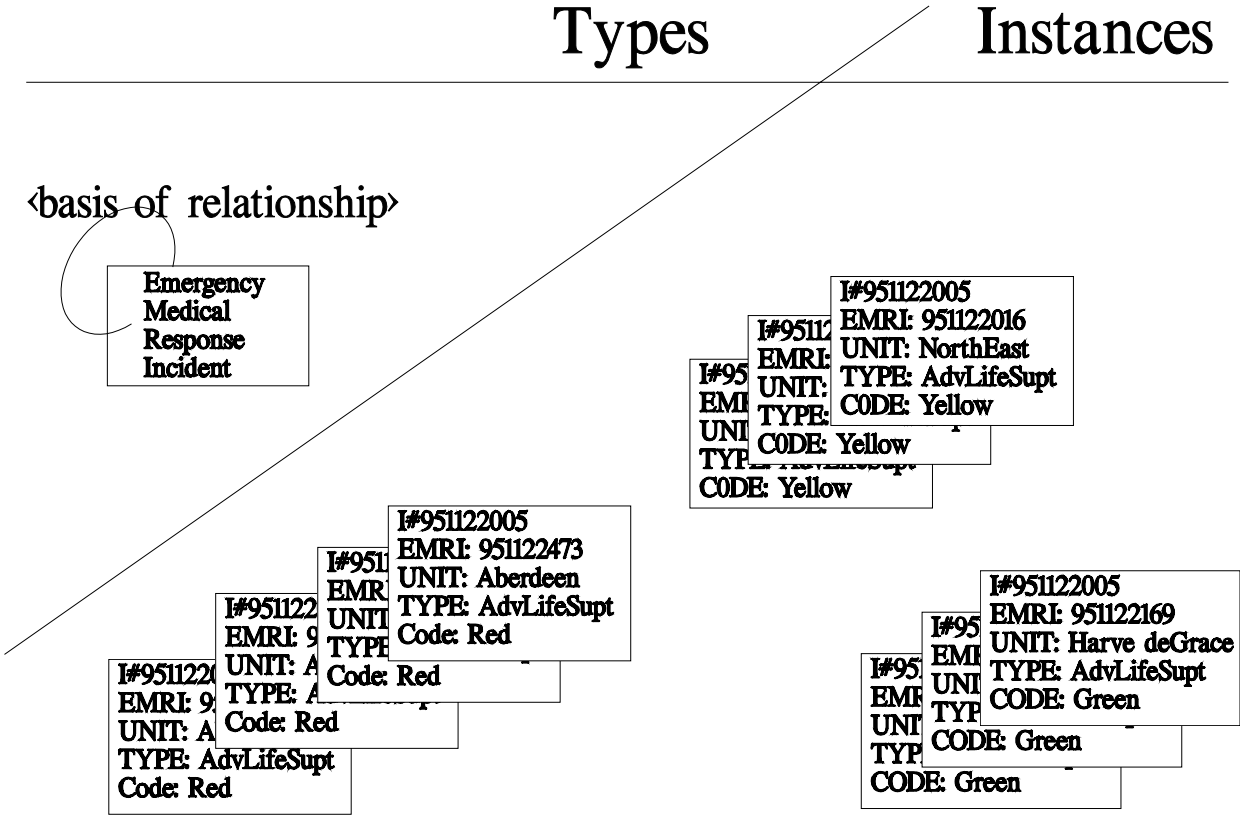


Figure 2.5. Singular-single member relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

A *singular-multiple-member* relationship has no owner table, but multiple member tables. There may be one or more member instances from each of the member tables for each relationship occurrence. In the multiple accident example, there is a responsibility indication for each of the incident types. Because of the accident's location, the Aberdeen Proving Grounds is close by and it has a set of U.S. Army advance life support units and Medivac helicopters. In addition, because the accident was so bad, a company of Military Police was dispatched to immediately secure the roadway perimeter while the state police performed tactical operations direction. The responsible organization table is able to own instances from each of the different incident types. Figure 2.6 illustrates this situation.

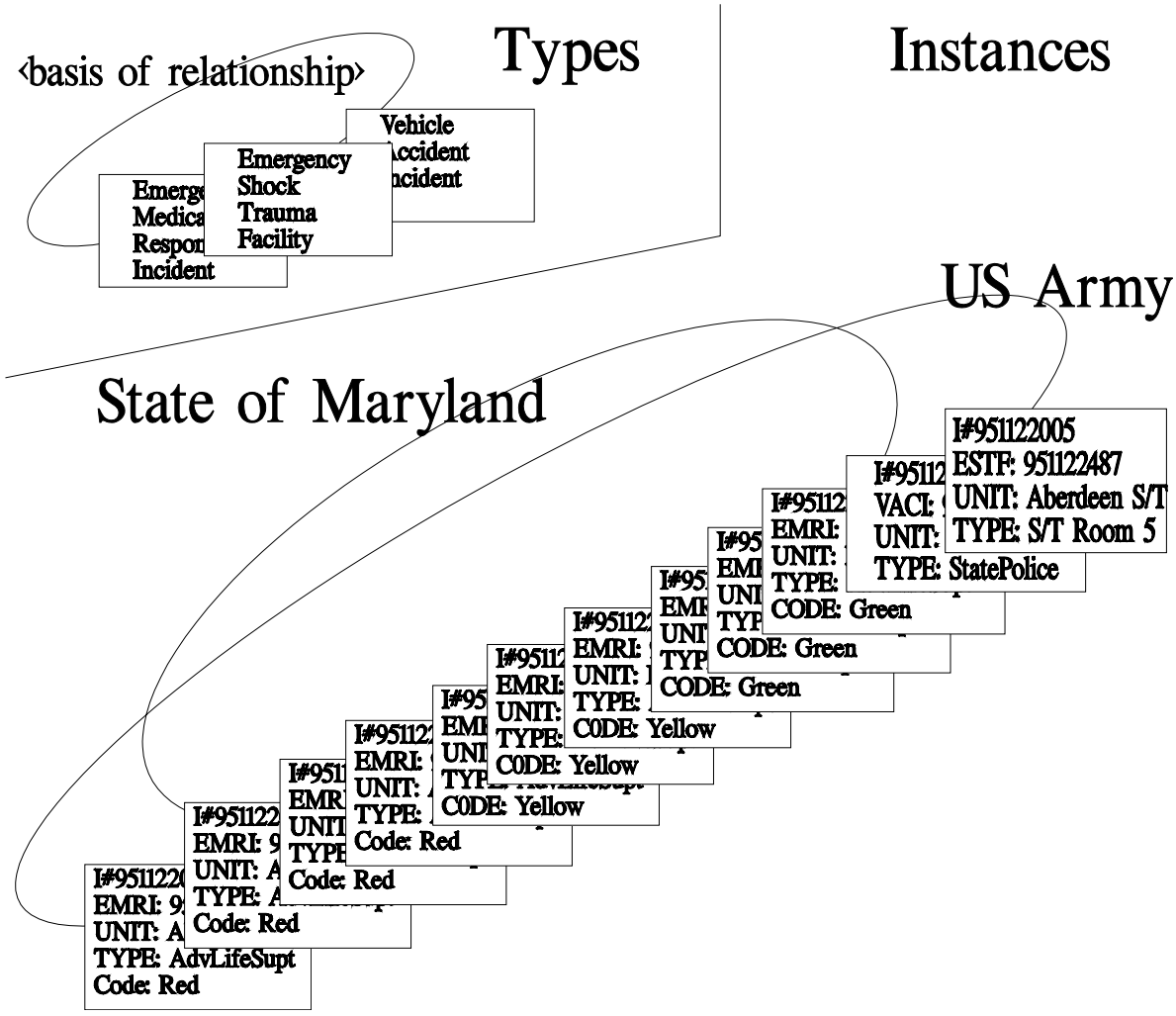


Figure 2.6. Types-based relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

One relationship instance that connects some of the instances from three different incident types identifies them as being the responsibility of the U.S. Army. The other relationship instance connects the remaining instances from the three different incident types as being the responsibility of the State of Maryland. The relationship type, singular multiple-member permits this type of association. A single query with the argument that provides the value for Responsible Organization selects all the instances from each of the different incident types.

Recursive relationships are a special class of owner-member relationship in which the owner table and the member table are the same. A recursive relationship exists between one instance of a table and other instances from the same table. Figure 2.7 illustrates a recursive relationship. This relationship is based on the harmful event aspect of the accident. The instances portray the hierarchy and the left-right sequence of events that took place.

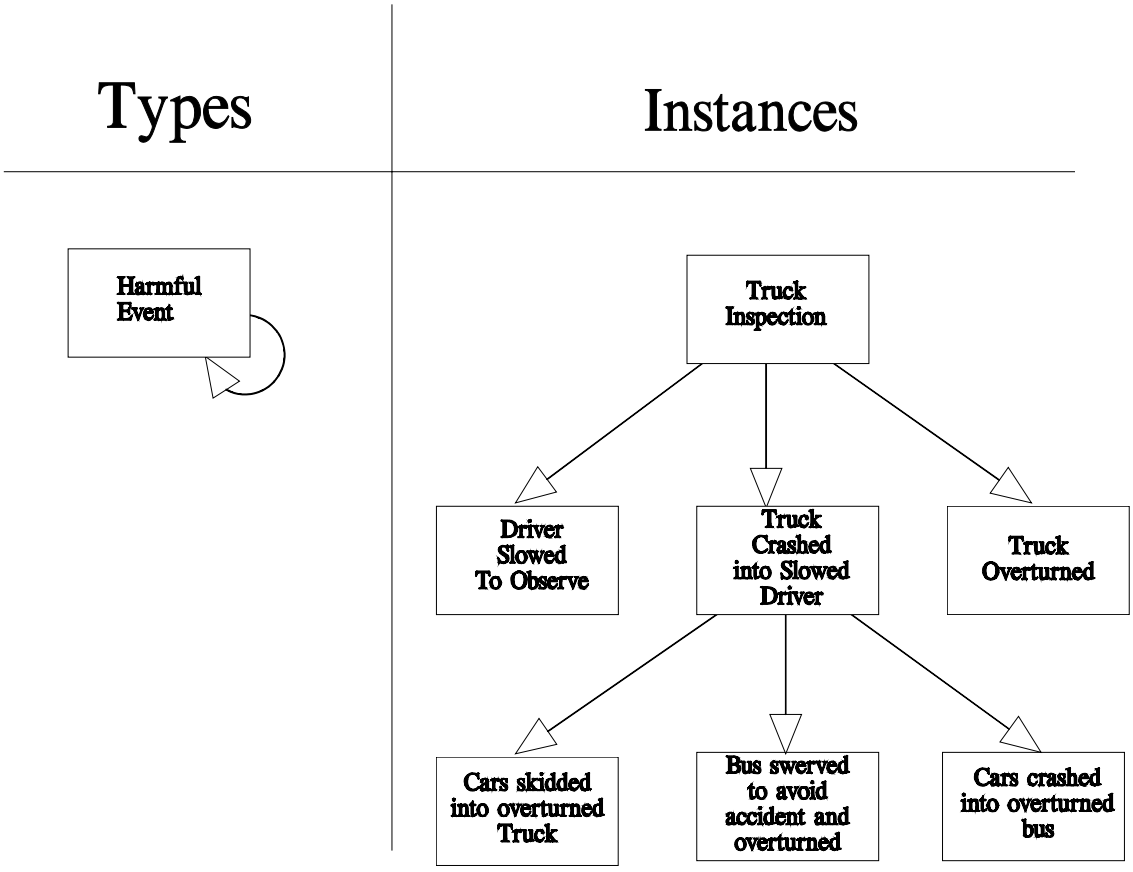


Figure 2.7. Recursive relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The *many-to-many* relationship, like the one-to-many relationship, involves two different rows. The relationship, however, is one-to-many in BOTH directions. The first of the tables is defined as the owner of the second, and the second of the tables is defined as the owner of the first. Figure 2.8 illustrates a many-to-many relationship. In this example, an Emergency Response Attendant (EMT, Paramedic, or an Ambulance crew) addresses the emergency medical needs of many different emergency medical response incident involved persons, and it is possible that multiple Emergency Response Attendants attend to the needs of one accident victim. Generally though, since information is usually recorded about the interaction of the EMR attendant and the victim, the many-to-many relationship really becomes two one-to-many relationships with the interaction information recorded in the association data structure.

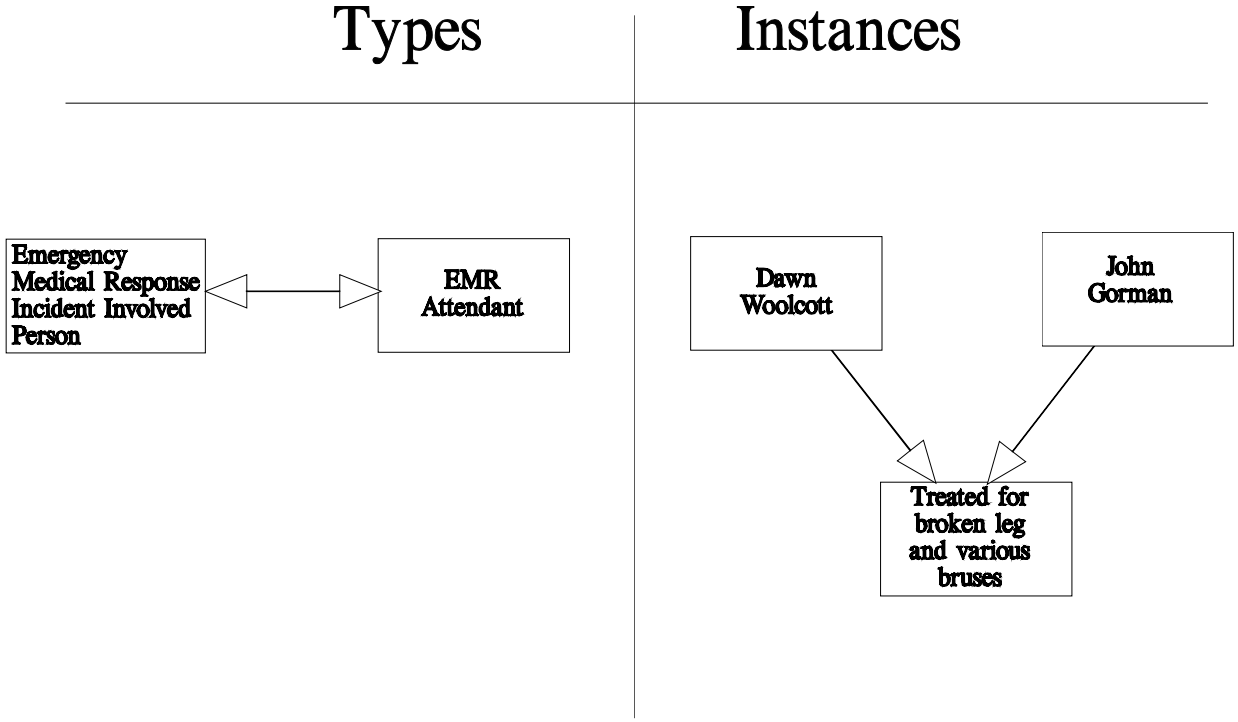


Figure 2.8. Many-to-Many relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The *one-to-one* relationship is not employed often. It binds a single row from one table to a single row of another table. The most common use of this relationship type is to segment the columns from one table into two or more tables. Suppose a table has four hundred columns, and that 90% of the time only 10% of the columns are accessed, and that the other 90% of the columns are accessed only 10% of the time. If a one-to-one relationship type is used to segment the table into two tables, 360 columns can be in the little accessed table and the other, often accessed, 40 columns can be in the other table. An example of the relationship is provided in Figure 2.9. In this figure, the quantity of data fields that need to be filled out by a Paramedic is significant. If the accident victim is in danger of immediate death, an extensive, time consuming medical exam may take too much of the "golden hour." The necessary information might only be that the victim needs immediate stabilization and Medivac transport to the Baltimore shock/trauma center.

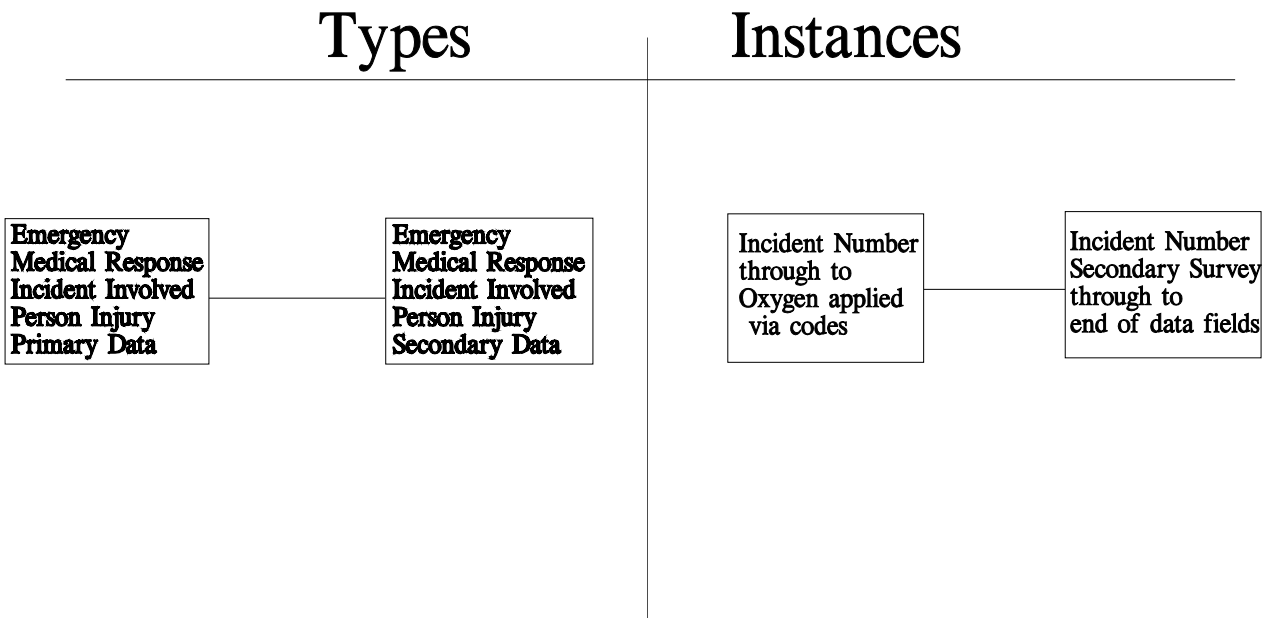


Figure 2.9. One-to-One relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The last relationship, *inferential*, relates rows from two tables. This relationship, however, does not involve the primary key of either table. Figure 2.10 illustrates an example of how this is used. An assessment is made of a victim and that an advanced life support vehicle is required. A list of the types of victim codes that are maximally allowed for a vehicle is produced. The relationship is then all vehicles that may be allowed the transport. That is, the inferential relationship between a type of EMR vehicle and the victim's condition. It would be true that a particular victim could have been transported by a particular vehicle based solely on the vehicle's rating and the victim's condition. That would not mean however that a particular victim was transported by a particular EMR vehicle. In this particular example, the victim Dawn Wolcott who received only a broken leg was classified as having "green" injuries, and EMR vehicle 17 is allowed to transport "green" injury victims.

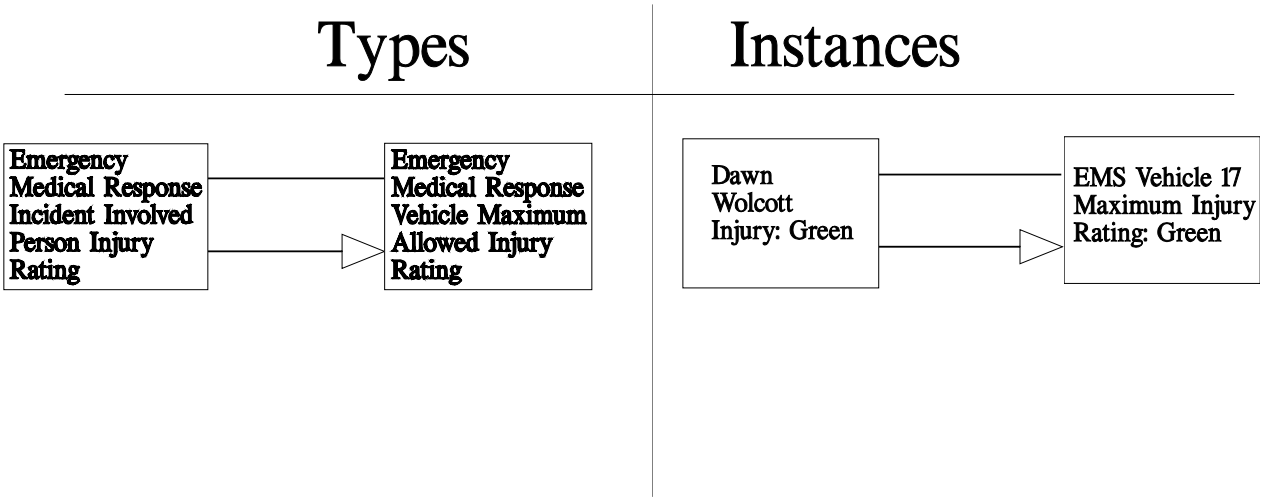


Figure 2.10. Inferential relationship for the database object class: incident.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The Transportation Public Safety database (see Chapter 4) contains about 25 distinct database objects. Within the 25 are about 150 discrete contained data structures. Every database object contains multiple one-to-many relationships between data structure components. About 10% of the data structures contain recursive relationships, and almost every database object has a need for a owner multiple-member relationship.

The sales and marketing database (see Chapter 5) includes customer, sales organizations, sales professionals, products and sales statistics and requires over 50 distinct data structures within the database objects. Among these tables are a good number of many-to-many, recursive, and one-to-many relationships. Some relationships bind closely related tables into unified data structure groups. For example, the customer classification database object requires three tables: customer classification, customer classification structure, and customer classification structure type. Customer assignments to customer classifications require both customer and the customer classification assignment table. In short, 5 tables to represent the customer and the customer classification database object. Multiple collections of these table sets are then bound into an overall sales and marketing data structure. Across the 50+ table structure there are only eight database objects, that is, sales persons, sales territories, customers, products, orders and shipments, and invoices and payments.

The courts database (see Chapter 6) addresses the needs of court processing. This database was designed to address justice matters once they arrive at the court, are processed through the court and that are completed by the court. The database objects are calendar, case, court, court transaction, issue, and participant. Across these database objects are about 80 distinct data structures.

All the tables within and between database objects require referential integrity. Referential integrity is a defined set of rules and actions that apply to identified relationships. The rules relate to the type of relationships that must be maintained, and the actions are the identified activities that must take place to enforce a rule or respond to a change in the database in order to maintain the database's referential integrity.

2.7 Data Models

Every DBMS's logical database is controlled by a set of rules. The combination of the rules that define tables, declare relationships, and that cause table (that is, row) and relationship (that is, join) operations is called the DBMS's data model.

Because data models have different table structures and relationship binding methods, each data model also permits specific operations to manipulate these structures. A data model's definition is thus in three parts:

- The formal definition of the permitted table structure, *AND*
- The formal definition of the number, types and kind of relationships that can be defined between rows of the same and different types, *AND*
- The formal definition of the operations that can take place on the resultant data structure of row and relationship instances

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

There are four[13] data models, network, hierarchy, independent logical file, and relational. Figure 2.11 depicts the components of data models. Static data models are built on relationships that are embedded in the rows. Static relationships are fully specified in the data definition language. Dynamic data models, in contrast, are based on retrieval time discovery of relationships among rows. Dynamic relationships are manifest as values shared between a columns in the related tables. Because of this critical difference, static data models contain relationship definitions while dynamic data models have data manipulation language relationship expressions.

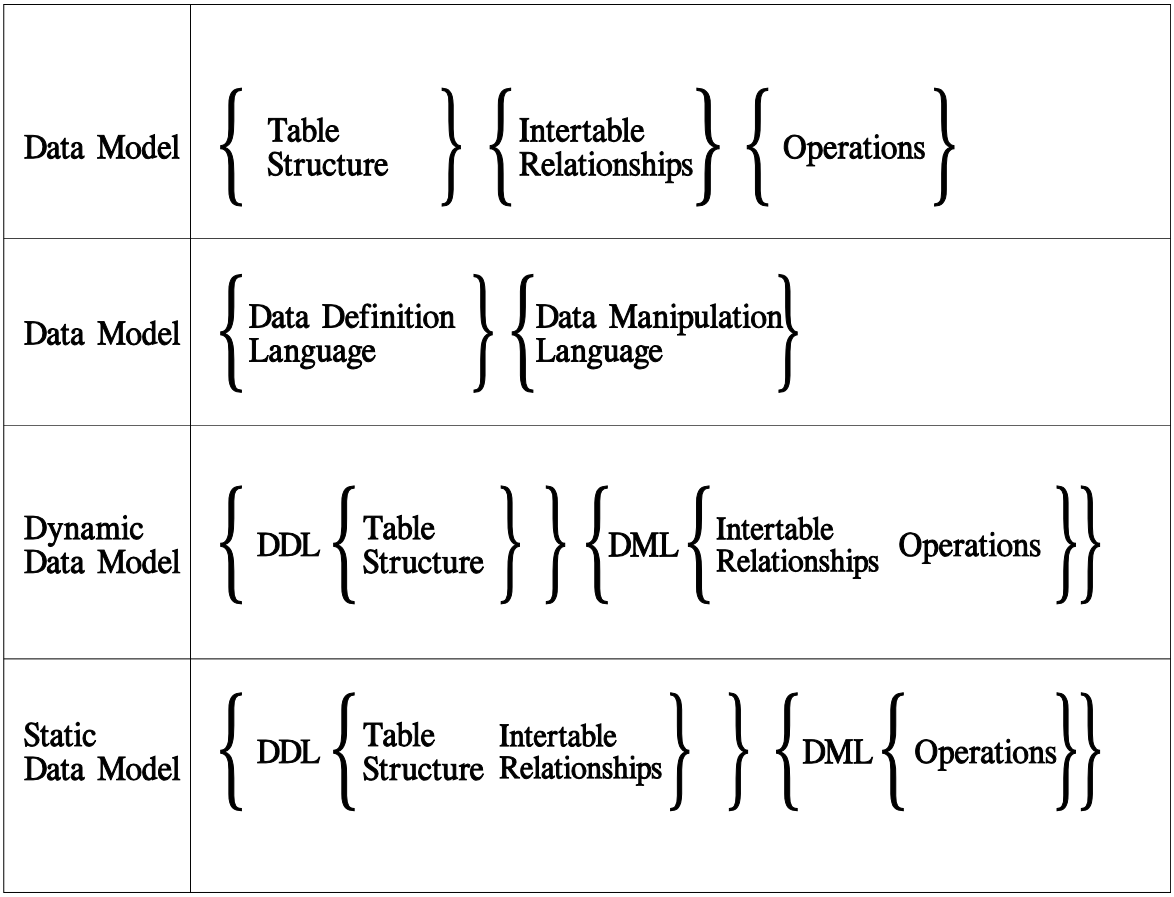


Figure 2.11. Data models components for DBMSs.

[13] Most database books recognize only three data models. To recognize only three eliminates recognition of about 75% of the 1960s through 1990s DBMS. Thus, any assertion of only three data models shows a lack of understanding of DBMS history

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Figure 2.12 depicts the four data models that existed from the 1960s through the late 1990s. SQL:1999 is none of these models. Rather, it is the union of them all. This figure illustrates data models by contrasting data structures, relationships, operations, data definition language components, and data manipulation language facilities.

Characteristics		Data Models			
		Static		Dynamic	
		Network	Hierarchical	Independent Logical File	Relational
Column Types in Table	Single valued	yes	yes	yes	yes
	Multi valued	yes	no	yes	no
	Multiple dimensions	yes	no	no	no
	Group	yes	no	no	no
	Repeating Group	yes	no	yes	no
Relationships		Table structures with predefined owners and members that create networks and hierarchies	Two dimensional table structures with predefined member table structures that create hierarchies	Multiple independent table structures. Relationships discovered through DML	Multiple, independent two dimensional tables. Relationships discovered through DML
Oper-ations	Row	Add, Delete, Find, Modify	Add, Delete, Find, Modify	Add, Delete, Find, Modify	Add, Delete, Find, Modify
	Relationship	Connect, Disconnect, Get owner Get member Get next	Get owner Get member Get next	Join, Intersect, Division, Union, Difference	Join, Intersect, Division, Union, Project
Data Definition Language		Table Structure Relationships	Table Structure Relationships	Table Structure	Table Structure
Data Manipulation Language		Operations	Operations	RelationshipsOperations	Relationships Operations

Figure 2.12. Components of DBMS data models from 1960 through the late 1990s.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Figure 2.13 show a very critical difference among data models: relationship types. Identified are the different types of relationships that can be explicitly defined in the four data models. A relationship that exists between an owner and one or more members is implemented either statically or dynamically. A DBMS embraces static relationships whenever the member's addresses or identifiers are embedded in the owner row or are stored in a separate relationship structure. A DBMS embraces dynamic relationships whenever the owner's address or identifier is stored as a value of a specific column of the member table. These relationships are accomplished either directly because there is no need to define additional tables, or is accomplished indirectly because additional tables have to be defined. A table is considered to be *additional* whenever the DBMS cannot execute the relationship directly without the database's designer first defining another table.

Relationship Type	**DATA MODEL**			
	Data Definition Language Declaration		**Data Manipulation Language Simulation**	
	ANSI Network	**Hierarchy**	**Independent Logical File**	**Relational**
Owner, 1 member	Direct & static	Direct & static	Direct & dynamic	Direct & dynamic
Owner, multiple members	Direct & static	No	No	No
No owner and 1 member	Direct & static	No	No	No
No owner, multi-members	Direct & static	No	No	No
Recursive	Direct & static	Indirect & dynamic	Direct & dynamic	Indirect & dynamic
Many to many	Indirect & static	Indirect & dynamic	Direct & dynamic	Indirect & dynamic
Inferential	Indirect & static	No	Direct & dynamic	Direct & dynamic
One to one	Direct & static	Indirect & dynamic	Direct & dynamic	Direct & dynamic

Figure 2.13. Critical difference among DBMSs: Relationship specification and binding.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

To accomplish an indirect relationship type requires the definition of additional tables. Thus, the relationship type is not really accomplished; rather it is transformed into a different relationship type, and then executed through custom programming in the various application programs. Finally, Figure 2.13 declares a relationship-data model intersection as a *no* whenever there is no practical method of accomplishing the relationship.

From Figure 2.13, it can be seen that the data models that require the least quantity of tables and relationships for typical business problems are the network (ANSI NDL) and the independent logical file data models. Because of this, it is not surprising that both the hierarchical and strictly relational DBMSs have fallen by the wayside.

The ANSI/NDL data model can directly model the most relationships, all eight of the relationships can be accomplished. Six directly, and two indirectly. The hierarchical data model can only implement one relationship directly, it cannot implement four, and three indirectly. The independent logical file data model can implement five relationships directly, but can not implement three others. The final data model, relational, can implement only three relationship types directly, two indirectly, and three others no. From this tally the DBMS most suitable for business applications is ANSI/ NDL. The second most suitable is ILF. Third place goes to relational, and fourth hierarchical. The table below summarizes this conclusion.

ANSI SQL from the 1986 standard through the 1992 standard used to only model relational data structures. SQL:1999 now models the union of all four data models. Hence the SQL:1999 (and also SQL:2003) data model is a data model unto itself. It is the inevitable response to the need to handle real business databases.

Network and hierarchical data models and their traditional implementations through network and hierarchical database management systems enabled business analysts to consciously define, and then data processing professionals to install, load, update, retrieve, and delete whole business relevant data structures. A data structure exists in two forms: data structure definition and data structure instances. Data structure definitions are often called metadata, and are communicated to a DBMS via a data definition languages (DDL). A data structure is for example all the data related to an entire employee, customer, and the like. Examples of business data structures are provided in Chapters 5 through 7. Relationally, the metadata for an employee data structure may be seen to contains upwards to 50 to 75 tables. Also relationally, the data for a single “logical” employee instance from these 50 to 75 tables could embrace several thousand rows. Hereafter such very sophisticated data structures are called business records.

Because both the network and hierarchical data models have the concept of a schema, record types within the schema, and explicitly declared relationships among the record types allow the explicit definition of business records.

The relational data model in contrast only has the concept of schema and table. Because there are no explicitly declared relationships, collections of tables cannot be seen, loaded, updated, retrieved, and deleted in terms of whole business records. To attempt to solve this problem, the relational data model was expanded to include referential integrity to preserve the integrity of references between tables.

Notwithstanding the inclusion of referential integrity, relational tables must undergo very elaborate, sophisticated, and very resource intensive data processing procedures to re-materialize business records. Consequently, SQL:1986 through SQL:1992 cannot explicitly define business records.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

To assist in shielding each and every user from having to create these data processing procedures in each and every computer program, the relational data model included the view concept. The data processing procedures are defined within the view construct. When the data processing routines automatically execute to retrieve a view record, the various selects and joins materialize the business records. While a view record might be perceived as a business record, there is nothing to prevent a different user from defining a different business record view. Finally, because a view does not have persistent and "firewall" characteristics, it is not a reliable substitute for the business record concept present in the network and hierarchical data models.

Finally, because users have access to all rows of a table, the ability to limit access to specific business records, that is, a fire-walled collection of rows from a large quantity of different tables, is very difficult. It is very difficult therefore to have business record encapsulation within a strictly relational database/DBMS.

The ANSI SQL standard (1986) language initially modeled itself after the relational model. Hence, ANSI SQL was not able to adequately handle sophisticated business records without sophisticated and resource intensive data processing programs. The initial ANSI SQL language standard also included views. In 1989 included simple referential was added. Full referential integrity was added by 1992.

There are two major characteristics of the relational model that are problematic: dynamic relationships and full table access. First, dynamic relationships means that all tables can be interrelated based solely on matching the value sets from any columns in the tables to be joined. The problem with this *capability* is not that it is present, but that its very presence causes severe performance implications. Further, it allows users to express queries that are semantically outside those intended by the business analysts. For example, a join could be created to FIND THE TEACHERS WHO HAVE AN MA AND THEIR STUDENTS WHO ARE BORN IN MA (Massachusetts). While this could be expressed in a completely valid SQL query, it clearly makes no business sense. It's not the possibility that's important, its the validity.

The second relational problem, full table access, means that all rows from any given table are available for retrieval and/or update regardless of the business record to which they belong. The most common implementation technique for rows within tables is to store them adjacent to each other in the order of the values of its primary key regardless of their business record membership.

The first relational problem, dynamic relationships, is seldom needed when the business records have been properly thought out and well designed. This is especially true for certain data architecture classes more than others. The three stable data architecture classes (see Chapter 3) are: reference data, original data capture, and operational data store. The dynamic data classes are the warehouse databases, both wholesale and retail. Transaction data staging area databases are generally not relevant to this issue.

The second relational problem, full table access, is similarly seldom needed because a majority of business reports need discrete business records. Traditionally implemented relational databases force the end-user to pay a high price to reconstruct business records for these very common reports. Relational DBMS vendors have responded to the need for efficient business records by allowing database administrators to precisely tune the physical placement of data. Smart relational DBMSs sense that a retrieved data block contains rows of data from multiple business records and do not repeat I/Os.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.8 ANSI Database Management System Specifications

The American National Standards Institute, ANSI, chartered H2 as the database languages committee in 1978. The first meeting was in Washington, DC. There have been an average of 6 meetings per year. The membership has ranged from the initial 15 to a maximum of about 60. The current 2006 membership is 14. Only two original members remain, Don Deutsch (the chairman), and this author (the secretary). Over the years there have been over 5000 technical papers. A number of papers are clearly the best from the brightest.

2.8.1 The Network Data Language (NDL) Project

The first H2 project standardized the CODASYL network data model. H2 named this language NDL, the network-data language. CODASYL, the COmmittee on Data Systems and Languages, through its data definition languages subcommittee had developed a number of journals of development (JOD) that specified a data definition language (DDL) for syntax and semantics to support the definition and implementation of network data structures. This DDL was completed by the end of 1980. To complete the specification, the DDL was complemented with a subschema/view facility and a data manipulation language. The subschema was based on the CODASYL COBOL's subschema language. The data manipulation language (DML) work was a complete new invention, by Phil Shaw[14] of Sybase (then of IBM). The NDL was completed by 1983. The NDL standard was delayed several years so the SQL standard could catch up. NDL and SQL both became ANSI database language standards in 1986.

The NDL is characterized by tables that contain vectors and groups. Tables can be formed into structures through a variety of relationship types. The language also contains a formally defined data manipulation language that can be employed as a semantics base by 3GL (COBOL, Fortran, C, etc.) to write explicit data access.

Some vendors today, for example, Oracle and Computer Associates implement database managements systems based on ANSI NDL.

2.8.2 The SQL Projects

SQL is a public domain three letter, three syllable “word.” It formerly stood for structured query language. The SEQUEL language, the IBM name for SQL was invented by IBM for accessing two dimensional data structures. The two dimensional structures and their operations was

[14] Phil Shaw was the IBM representative to X3H2 since the first meeting in 1978. Phil provided the vast majority of the technical papers for the network data language standard, NDL, and because of Phil, NDL is a very high quality specification. While much is said in the press and on the lecture and seminar circuit, of the work of the originators of the SQL language, they are its fathers in name only, because it was Phil Shaw, over the years, without notice or attention, who has toiled days, weeks, and years drafting, redrafting, and word smithing the ANSI SQL:1986 standard. And, once the American NDL and SQL:1986 standards were finished, it was Phil who was the key American representative to the International Standards Organization (ISO), enabling both NDL and SQL:1986 to become international standards.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

described by E.F. Codd in 1970. IBM named and then trademarked the structured query language SEQUEL. SQL is often mistakenly pronounced the same as SEQUEL. Not only are the words spelled and pronounced differently, the languages have become very different. The ANSI SQL projects embraces a number of standards including:

- The main SQL for data definition, data loading, update, manipulation, and protection of data according to the SQL data model.
- The specification of language interfaces to 3GLs such as COBOL, FORTRAN, and C.
- The specification of a schema based stored procedure language.
- The specification of a call-level interface to between a language agent and an SQL engine.
- A subfamily of projects called SQL/MM that currently embraces temporal based data, full-text retrieval, and families of abstract data type definitions.

In brief, the SQL standard, published in 1986, included some integrity constraints, views, flat table structures, and access through 3GLS. It was about 100 pages. The first SQL standard enhancement, SQL:1989, published in 1989 added a restricted set of referential integrity. For example, there was no cascade delete. The SQL:1989 standard was about 125 pages.

The SQL standard published in 1992 added full referential integrity, a schema manipulation language, integrity constraints, and multiple character set (different languages) support. The SQL:1992 standard was about 500 pages.

SQL:1999 has made fundamental changes in architecture, data models, and has greatly expanded functionality. With respect to architecture, SQL:1999 embraces two phase commit and asynchronous transaction processing. With respect to data model, SQL:1999 is no longer relational, it is firmly camped within the data model richness of the independent logical file by adding subtables, abstract data types, sets, lists, and multi-sets. As to features, SQL:1999 has added a full self-contained data manipulation language, assertions, triggers, and stored procedures

2.9 The Effect of ANSI Database Specifications on Database Objects

The only ANSI standard data model being popularly implemented is SQL. The recent advances in the ANSI SQL specification are progressing the language and its implied facilities towards the requirements of database objects.

The most significant difference between SQL:1986 through SQL:1992 and the non-relational SQL standards: SQL:1999 and SQL:2003 is that through SQL:1992, the standard efforts were focused on making a relational-like language sufficiently complete to accomplish standard, non-object oriented business applications possible. SQL:1986 and SQL:1989

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

established the SQL language as the standard language for all persistent data definition and manipulation whenever the persistent data is to be accessed by multiple 3GLs.

SQL:1992, as the first significant SQL extension made SQL practical to employ for robust commercial applications. Not withstanding these advances, SQL:1992 requires the full use of 3GLs and/or 4GLs to build and maintain database objects because 3GLs and 4GLs cannot, by definition be encapsulated within the total control structure of a DBMS, that is, shielded from end-user change and/or manipulation, SQL:1992 cannot be employed to design, develop, deploy, and maintain database objects. Simply, without SQL:1999, database objects are not possible.

1* Student Number (Integer 9(6))
2* Student Name (Non-key Name X(15))
3* Birthdate (Non-key Date)
4* Ethnic Code (Non-key Name X)
5* Sex (Non-key Name X)
6* Federal Property Residence (RG)
    7* Name of Parent (Non-key Name X(15) in 6)
    8* Military Address (Non-key Name X(25) in 6)
    9* Fed Property Code (Key Name X in 6)
    10* Branch/agency/department (Non-key Name X(4) in 6)
    11* Rate/rank (Non-key Name X(5) in 6)
    12* Present Station (Non-key Name X(10) in 6)
    13* Unit (Non-key Name X(10) in 6)
    14* Serial Number (Non-key Name X(10) in 6)
15* Post HS Aspiration (Non-key Name XXx)
16* Post HS Aspiration Date (Non-key Date)
17* Head of Household (Non-key Name X)
18* Hoh SSN (Non-key Integer 9(9))
19* Education of HOH (Non-key Integer 99)
20* Primary Language At-home (Non-key Integer 99)
21* Socio-economic Status (Non-key Integer 99)
22* Family Size (Non-key Integer 99)
23* Migration Codes (RG)
    24* M-code (Key Integer 99 in 23)
    25* M-date (Non-key Date in 23)
26* Migration Codes (RG)
    27 M-code (Key Integer 99 in 23)
    28 M-date (Non-key Date in 23)
29 Handicap Status (RG)
    30 H-code (Key Integer 99 in 23)
    31 H-date (Non-key Date in 23)
32 Disadvantaged Status (RG)
    33* Relation to HOH (Key Integer 99 in 32)
    34* O-code (Key Integer 99 in 23)
    35* O-date (Non-key Date in 23)
36 Dropout Proneness (RG)
    24* D-code (Key Integer 99 in 23)
    25* D-out-date (Non-key Date in 23)
39 Public Assistance Information (RG)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

```
        24*     Public Aid Code (Key Integer 99 in 23)
        25*     Public Aid Date (Non-key Date in 23)
42*     Standard Test Results (RG)
        43*     Test Name (Key Integer 99 in 42)
        44*     Test Date (Non-key Date in 42)
        45*     Test Scores (RG in 42)
                46*     Subtest (Key Name XX in 45)
                47*     Subtest Score (Non-key Integer 9999 in 45)
50*     Student Services Used (RG)
        51*     Student Service (Key Integer 99 in 50)
        52*     Service Date (Non-key Date in 50)
53*     Student Record (RG)
        54*     Date of Record (Key Name X(5) in 53)
        55*     Grade/module (Non-key Integer 99 in 53)
        56*     School Code (Non-key Integer 999) in 53)
        57*     Cum GPA (Non-key Decimal 9.999 in 53)
        58*     Percent Absent (Non-key Decimal 99.99 in 53)
        59*     Courses/activities/studies (RG in 53)
                60*     Course Number (Key Integer 999 in 59)
                61*     Section Number (Non-key Integer 9(5) in 59)
                62*     Program Number (Non-key Integer 9(5) in 59)
                63*     Instructor Number (Non-key Integer 9(9) in 59)
                64*     Grade Points (Non-key Decimal 9.999 in 59)
                65*     Credits (Non-key Integer 99  in 59)
                66*     Percent of Time (Non-key Decimal 9.99  in 59)
                67*     Work/study Occupation (Non-key Integer 99  in 59)
                68*     Employer (Non-key Name X(15)  in 59)
                69*     Passing/failing (Non-key Integer 9  in 59)
                70*     Grade/Achievement Level (Non-key Integer 99  in 59)
```

Figure 2.1a. System 2000 data definition language for a Students database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

100* School Characteristics (RG)
- 101* School Year (Key Integer 9999 in 100)
- 102* School Name (Key Name X(20) in 100)
- 103* School Number (Key Integer 999 in 100)
- 104* School Type (Key Name X in 100)
- 105* Follow-up Fulfilling Post Hs Expectation (Key Decimal 99.99 in 100)
- 106* Follow up in College (Key Decimal 99.99 in 100)
- 107* Follow-up Full Time Employed (Key Decimal 99.99 in 100)
- 108* Follow-up Other (Key Decimal 99.99 in 100)
- 109* Community Programs (RG in 100)
  - 110* Community Program Name (Key Name X(20) in 109)
  - 111* Community Program Type (Key Integer 99 in 109)
- 112* Major Disruptions (RG in 100)
  - 113* Disruption Type (Key Integer 99 in 112)
  - 114* Disruption Comment (Non-key Name X(20) in 112)
- 115* Grades in School (RG in 100)
  - 116* Grade/module (Key Integer 99 in 115)
  - 117* Grade/module Student Count (Key Integer 9(5) in 115)
  - 118* Ethnic Groups in Grade (RG in 115)
    - 119* Ethnic Group (Key Integer 99 in 118)
    - 120* Ethnic Group Student Count (Key Integer 9(5) in 118)
    - 121* Target Groups (RG in 118)
      - 123* Target Group Type (Key Integer 99 in 121)
      - 124* Target Group Student Count 99 in 121)
      - 125* Percent Reading Acceptable (Key Decimal 99.99 in 121)
      - 126* Percent with Vocational Skill (Key Decimal 99.99 in 121)
      - 127* Percent with Educational Audit Acceptable (Key Decimal 99.99 in 121)
      - 128* Dropout Rate (Key Decimal 99.99 in 121)

Figure 2.1b. System 2000 data definition language for a School Characteristics Tree within the database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

```
200*  Programs (RG)
      201*  Program Name (Key Name X(20) in 200)
      202*  Program Number (Key Integer 9(5) in 200)
      203*  Annual Reports (RG in 200)
            204*  Report Year (Key Integer 9999 in 203)
            205*  Program School Number (Key Integer 999 in 203)
            206*  Total Program Cost (Key Integer 9(8) in 203)
            207*  Total Salaries Cost (Key Integer 9(8) in 203)
            208*  Total Capital Cost (Key Integer 9(8) in 203)
            209*  Total Other Costs (Key Integer 9(8) in 203)
            210*  Vertical File Location (Key Name X(6) in 203)
            211*  Tuition Income (Key Integer (6) in 203)
            212*  Institutional Methods (RG in 203)
                  213*  Instructional Method (Key Name X(10) in 212)
            214*  Grades/courses (RG in 203)
                  215*  Grade/achievement Level (Key Name X(5) in 214)
                  216*  Program Course Number (Key Integer 999 in 214)
                  217*  Program Section (Key Integer 99 in 214)
                  218*  Program Building Number (Key Integer 9999 in 214)
                  219*  Program Room Number (Key Integer 999 in 214)
                  220*  Program Period (Key Integer 99 in 214)
                  221*  Meetings per Week (Key Integer 9 in 214)
                  222*  Student Characteristics Distribution (RG in 214)
                        223*  Ethnic Type (Key Integer 99 in 222)
                        224*  Sex (Key Integer 9 in 222)
                        225*  Age Group (Key Integer 9 in 222)
                        226*  Target Group (Key Integer 9 in 222)
                        227*  Group Count (Key Integer 99 in 222)
                  231*  Student Grouping Method (RG in 214)
                        232*  Grouping Method (Key Name X in 231)
                        233*  Assignment Rationale (Key Name X in 231)
            240*  Relationship to Indicators (RG in 200)
                  241*  Indicator (Key Name XX in 240)
                  242*  Relationship to Program (Key Decimal .99999 in 240)
                  243*  Indicator Date (Key Date in 240)
            250*  Agency Sponsorship (RG in 200)
                  251*  Agency Name (Key Name X(10) in 250)
                  252*  Percent Sponsorship (Key Decimal 99.99 in 250)
```

Figure 2.1c. System 2000 data definition language for a Programs Tree within the database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

300* Facilities (RG)
- 301* Status of Building (Key Name X in 300)
- 302* Owner (Key Name X(10) in 300)
- 303* Building Name (Key X(15) in 300)
- 304* Building Number (Key Integer 9999 in 300)
- 305* Building Function (Key Integer 99 in 300)
- 306* Year Completed (Key Integer 9999 in 300)
- 307* Conform to Building Codes (Key Name X in 300)
- 308* Sqft under Roof (Key Integer 9(6) in 300)
- 309* Sqft Outside (Key Integer 9(6) in 300)
- 310* Sqft Potential under Roof (Key Integer 9(6) in 300)
- 311* Twelve Month Maintenance Cost (Key Integer 9(7) in 300)
- 312* Building Address (Non-key Name X(20) in 300)
- 313* Building County (Key Name X(10) in 300)
- 314* Additions (RG in 300)
  - 315* Addition Project (Key Name X(15) in 314)
  - 316* Year Addition Completed (Key Integer 9999 in 314)
- 317* Functional Areas (RG in 300)
  - 318* Functional Area Type (Key Name X in 317)
  - 319* Function Square Foot Area (Key Integer 9(6) in 317)
- 320* Classroom Types (RG in 300)
  - 321* Classroom Type (Key Integer 99 in 320)
  - 322* Classroom Year Available (Key Integer 9999 in 320)
  - 323* General Utilization Ratio (Key Decimal 99.99 in 320)
  - 324* Room Numbers (RG in 320)
    - 325* Classroom Number (Key Integer 999 in 324)
    - 326* Maximum Student Stations (Key Integer 999 in 324)

Figure 2.1d. System 2000 data definition language for a Facilities Tree within the database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

400* Staff (RG)
- 401* Employee Name (Key Name X(15) in 400)
- 402* Social Security Number (Key Integer 9(9) in 400)
- 403* Ethnic Type (Key Integer 9 in 400)
- 404* Sex (Key Name X in 400)
- 405* Date of Birth (Key Date in 400)
- 406* Birth State (Key Name XX in 400)
- 407* Last Salary Action Date (Key Date in 400)
- 408* Local Salary Date (Key Date in 400)
- 409* Latest Hire Date (Key Date in 400)
- 410* Professional Status (Key Integer 99 in 400)
- 411* Marital Status (Key Name X in 400)
- 412* Employment Status (Key Name X in 400)
- 413* Termination Date (Key Name X in 400)
- 414* National Teacher Exam Year (Key Integer 9999 in 400)
- 415* National Teacher Exam Score (Key Integer 999 in 400)
- 416* Citizenship Code (Key Name X in 400)
- 417* Filing Number for Papers (Non-key Name X(5) in 400)
- 418* Higher Education (RG in 400)
  - 419* Degree (Key Integer 99 in 418)
  - 420* Year Achieved (Key Integer 999 in 418)
  - 421* College Code (Key Integer 9999 in 418)
  - 422* Areas of Study (RG in 418)
    - 423* Scholastic Level (Key Name X in 422)
    - 424* Study Area (Key Integer 9999 in 422)
    - 425* Semester Hours (Key Integer 99 in 4222)
- 426* Certification (RG in 400)
  - 427* Type Certificate (Key Name X in 426)
  - 428* Year Awarded (Key Integer 9999 in 426)
  - 429* Expiration Date (Key Date in 426)
  - 430* Issuing State (Key Name XX in 426)
  - 431* Competence Codes (RG in 400)
    - 432* Competency Code (Key Integer 9999 in 431)
- 440* Employee History (RG in 400)
  - 441* Year of Record (Key Integer 9999 in 440)
  - 442* School District (Key Name XXX in 440)
  - 443* Months on Job (Key Decimal 999.9 in 440)
  - 444* Part/full Time (Key Name X in 440)
  - 445* Leave of Absence Type (Key Name X in 440)
  - 447* Job Code (Key Integer 9(4) in 440)
  - 448* Major Duty (Key Integer 99 in 440)
  - 449* Principal Evaluation (Key Name X in 440)
  - 450* Pay Grade (Key Integer 99 in 440)
  - 451* Salary Step (Key Name X in 440)
  - 452* Annual Salary (Key Decimal 9(5).99 in 440)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

453* Contract Status Code (Key Integer 99 in 440)
454* Bi-lingual Instructional Competence (Key Name X in 440)
455* Assignments (RG in 440)
    456* Program Number Assignment (Key Integer 9(5) in 455)
    457* Course Assignment (Key Integer 999 in 455)
    458* Section Assignment (Key Integer 99 in 455)
    459* Room Assignment (Key Integer 999 in 455)
    460* Assignment Code Code (Key Integer 9999 in 455)
    461* Percent of Total Time (Key Decimal 99.99 in 455)
470* in Service Education (RG in 440)
    471* Start Date (Key Date in 470)
    472* Length in Hours (Key Integer 99 in 470)
    473* Type of in Service Education (Key Name X(5) in 470)

Figure 2.1e. System 2000 data definition language for a Staff Tree within the database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1* School Year (Key Integer 9999)
2* Total Forecast Revenue (Key Integer 9(9))
3* Total Expenditures (Key Integer 9(9))
4* Total Area Population (Key Integer 9(7))
5* Unemployment Rate (Key Decimal .99)
6* Ethnic Group Characteristics (RG)
- 7* Ethnic Code (Key Name X in 6)
- 8* Population (Key Integer 9(6) in 6)
- 9* Unemployment Rate (Key Decimal .99 in 6)
- 10* Birthrate per Thousand (Key Integer 9(5) in 6)
- 11* Median Family Income (Key Integer 9(5) in 6)
- 12* Pct Receiving Public Assistance (Key Decimal .99 in 6)
- 13* Tax Obligation (Key Integer 9(6) in 6)
- 14* Children 0-1 (Key Integer 9(5) in 6)
- 15* Children 1-2 (Key Integer 9(5) in 6)
- 16* Children 2-3 (Key Integer 9(5) in 6)
- 17* Children 3-4 (Key Integer 9(5) in 6)
- 18* Children 4-5 (Key Integer 9(5) in 6)
- 20* Residence Types (RG in 6)
  - 21* Residence Type Code (Key Name XX in 20)
  - 22* Residence Count (Key Integer 9999 in 20)
  - 23* Pct of Total Residences (Key Decimal .99 in 20)
  - 24* School Age Children per Residence (Key Decimal 9.9 in 20)

25* Revenue Sources (RG)
- 26* Source (Key Name X(6) in 25)
- 27* Dollar Receipts (Key Decimal 9(7).99 in 25)
- 28* Authorization Agency (Key Name X(10) in 25)
- 29* Revenue Source Formula (Key Name X(6) in 25)
- 30* Tax Base (Key Decimal 9(6).99 in 25)
- 31* Assessment Fraction (Key Decimal .99 in 25)
- 32* Tax Rate (Key Decimal 99.99 in 25)
- 34* Collection Rate (Key Decimal .99 in 25)
- 35* Fund Use Restrictions (RG in 25)
  - 36* Restriction Code (Key Integer 9(5) in 37)
  - 37* Programs Authorized (RG in 35)
    - 38* Program Number (Key Integer 9(5) in 38)
    - 39* Program Name (Key Name X(15) in 37)
    - 40* Amount from Source (Key Integer 9(6) in 37)
    - 41* Pct of Source (Key Decimal .99 in 37)
    - 42* Federal Sources Not Paid (RG in 37)
      - 43* Service Type (Key Name XX in 42)
      - 44* Service Value (Key Decimal 9(5).99 in 42)

Figure 2.1f. System 2000 data definition language for a School Year database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

```
FILENAME=employee ,suffix=foc

SEGNAME=empinfo, segtype=s1
FIELDNAME=emp_id      ,alias=eid    ,format=a9     ,$
FIELDNAME=last_name   ,alias=ln     ,format=a15    ,$
FIELDNAME=first_name  ,alias=fn     ,format=a10    ,$
FIELDNAME=hire_date   ,alias=hdt    ,format=i6ymd  ,$
FIELDNAME=department  ,alias=dpt    ,format=a10    ,$
FIELDNAME=curr_sal    ,alias=csal   ,format=d12.2m ,$
FIELDNAME=curr_jobcode,alias=cjc    ,format=a3     ,$
FIELDNAME=ed_hrs      ,alias=ojt    ,format=f6.2   ,$

SEGNAME=fundtran, segtype=u, parent=empinfo
FIELDNAME=bank_name   ,alias=bn     ,format=a20    ,$
FIELDNAME=bank_code   ,alias=bc     ,format=i6s    ,$
FIELDNAME=bank_acct   ,alias=ba     ,format=i9s    ,$
FIELDNAME=effect_date ,alias=edate  ,format=i6ymd  ,$

SEGNAME=payinfo,segtype=sh1,parent=empinfo
FIELDNAME=dat_inc     ,alias=di     ,format=i6ymd  ,$
FIELDNAME=pct_inc     ,alias=pi     ,format=f6.2   ,$
FIELDNAME=salary      ,alias=sal    ,format=d12.2m ,$
FIELDNAME=jobcode     ,alias=jbc    ,format=a3     ,$

SEGNAME=address,segtype=s1,parent=empinfo
FIELDNAME=type        ,alias=at     ,format=a4     ,$
FIELDNAME=address_ln1 ,alias=ln1    ,format=a20    ,$
FIELDNAME=address_ln2 ,alias=ln2    ,format=a20    ,$
FIELDNAME=address_ln3 ,alias=ln3    ,format=a20    ,$
FIELDNAME=acctnumber  ,alias=ano    ,format=i9l    ,$

SEGNAME=salinfo,segtype=sh1,parent=empinfo
FIELDNAME=pay_date    ,alias=pd     ,format=i6ymd  ,$
FIELDNAME=gross       ,             ,format=d12.2m ,$

SEGNAME=deduct,segtype=s1,parent=salinfo
FIELDNAME=ded_code    ,alias=dc     ,format=a4     ,$
FIELDNAME=ded_amt     ,alias=da     ,format=d12.2m ,$

SEGNAME=jobseg  ,crfile=jobfile,crkey=jobcode,parent=payinfo ,segtype=ku ,$
SEGNAME=secseg  ,crfile=jobfile,              parent=jobseg  ,segtype=klu,$
SEGNAME=skillseg,crfile=jobfile,              parent=jobseg  ,segtype=kl ,$
SEGNAME=attndseg,crfile=educfile,crkey=emp_id,parent=empinfo ,segtype=km ,$
SEGNAME=courseg ,crfile=educfile,             parent=attndseg,segtype=klu,$
```

Figure 2.1g. FOCUS data definition language for an Employee database.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

3

SUCCESSFUL COMPUTING ENVIRONMENTS

The goals of a successful computing environment in enterprise-wide and world-wide computing are:

- Develop and maintain sophisticated and complex systems that access data and perform appropriate processes
- Accomplish information system development and maintenance *within* the life span of a modern business problem
- Develop a greatly increased quantity of information system solutions to business problems at a lower per unit cost
- Create applications that are both relevant to the individual and applicable to the enterprise as a whole

The evolution of computing environments can be seen through the evolution of: computer languages, information systems, hardware and operating systems, and the staff required to bring all the components together.

3.1 Computer Language Evolution

Computer languages were first formalized in the late Fifties. From the first generation of computer languages until now, language development has been evolving towards two development goals:

- Making languages independent of the host computer and operating systems
- Accomplishing more computing activities with less human effort.

While not yet at the thought-based code generator, languages have evolved greatly from their machine language ancestors.

In general, there have been four generations of computer languages. First generation languages were machine languages that were tightly bound to the specific computer models, operated without operating systems, and accomplished single data processing activities. Second generation languages operated through operating systems and accomplished multiple data processing activities. Third generation languages were drafted by committees (usually CODASYL (Committee on Data Systems and Languages)) and standardized by ANSI, and are thus independent of operating system and computer hardware manufacturers. Fourth generation languages, while more independent of computers and operating system are either dependent on DBMSs, or are owned exclusively by the language's vendor. Under either case, the result is the same: proprietary, vendor dependent environments.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Third generation languages (3GL) contained formal data definition facilities. These facilities defined data structures within the context of a record. The most sophisticated data definition facility for business applications exists within the language COBOL. A COBOL record definition can contain substructures down to nine levels. In addition, COBOL has the ability to re-define structures that cause the data to be interpreted differently depending on the value of a "higher" field.

COBOL did not however have the ability to define relationships between different record types. Finally, the data structure's definition had to be included in the computer program at compile time. That made production COBOL programs 100% sensitive to data structure changes. If a system of programs existed that required a data structure definition change to one program, then all the other programs that included antiquated data structure, were "broken." They all had to be changed and recompiled to "know" of the new data structure. In short, while there were persistent data stores, there were no persistent data store definitions. COBOL has never been considered as a database management system because a DBMS, by definition, requires persistent data definitions that are independent of the programs that access the data.

Fourth generation languages (4GL) are database access languages. There are two varieties:

- DBMS vendor proprietary and
- Independent vendor proprietary.

The DBMS Vendor's 4GLs. automatically "know" about the database to which they are linked. This knowledge brings automatic variable definition, data typing, and any automatic editing and validation that is defined within the database's data structure definition (schema). While automatic knowledge of database structures is a considerable "plus," a very significant negative is that the fourth generation languages are bound to the DBMS vendor's DBMS. There is no ANSI standard activity that standardizes database access languages independent of a DBMS vendor. The current ANSI SQL activity for persistent stored modules (ANSI SQL PSM) language statements can only access one DBMS's database at a time, and only operates within the context of either a 3GL or implicitly through a DBMS vendor's 4GL. The ANSI SQL PSM specification has remained largely unchanged since it was first standardized in the late 1990s.

The independent vendor 4GLs, like Power Soft (prior to being owned by Sybase), Borland's Delphi, and Clarion for Windows are end-user facility rich languages the interact with ANSI SQL DBMSs either through vendor proprietary interfaces or through de facto standard interfaces like ODBC. ANSI's H2 just finished standardization of the SQL DBMS access subset of calls within ODBC. The ANSI standard is called SQL/CLI. Microsoft the primary developer of ODBC conforms to the SQL/CLI subset in its ODBC version 3.

Organizations intending to develop DBMS vendor independent systems or systems that access multiple databases under the control of different DBMS vendors must use ANSI standard 3GLs like COBOL, FORTRAN, or C.

An interesting side effect occurred on the language progression road: the ratio of the time to learn and to accomplish the application dramatically changed. The time to learn the application, given traditional methods and a complete lack of the subject matter knowledge, has essentially remained constant. Application accomplishment, database design and database

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

application development through 4GLs has dramatically shrunk. Because of these changes, the ratio between learning and doing, which used to be a fraction of the total systems development time, has become a multiplier. To counter this ratio reversal, either subject matter experts must be charged with application development, or subject matter templates must be stored in a repository[15] and be available for use. Complicating the changed ratio even further, is the simple fact that information systems have changed from single user, application and site to multiple user, application, and site (even world-wide).

A significant enhancement in language development has been code generators. Code generators interact with users and then produce completely operational computer programs. Code generators exist for 3GLs and for 4GLs. The benefit of using 3GL code generators is that they produce ANSI standard process logic code, and also produce data access logic in ANSI standard SQL. Code generators perturb the ultimate learn-to-do ratio to the maximum. Code generators store the output of systems analysis and design in a repository and generate computer source code.

This author, for example, used one code generators, Information Builder's PC/FOCUS from the early 1980s through the mid 1990s to produce working repository systems. Since the mid 1990s, SoftVelocity's Clarion for Windows has greatly exceeded the capabilities of PC/FOCUS in this regard. The amount of time on a PC to create a 500 line computer program that adds, deletes, and maintains an entire structure of hierarchically related tables is about 5 seconds. Given that the time to create the design of the data structure is two months, then the ratio (times faster to code than to analyze and design) is so large that it becomes meaningless (345,600 = 60 days *8 work hours *60 minutes *60 seconds /5 seconds). The real value under this scenario is the artifacts of analysis and design. The generated code artifacts are almost valueless as they can be re-generated at almost no cost.

Given the obscene learn-to-build ratio, it is even more obscene not to retain the systems analysis and design products in a re-usable repository so the contained metadata can be used many times. If the metadata is used 10 more times for developing different aspects of the same overall application then while the ratio doesn't improve to a respectable number (down to only 31,418), the amount of systems analysis and design time saved approximates two staff years, or over $350,000. In 2000- 2001 Whitemarsh built a 6600 function point system for $350,000. Through traditional methods it would have cost $2.6 million. The difference was not a dramatic decrease in the analysis and design time, it was an almost zero amount of time devoted to "detailed design, code, and unit test."

Code generators, which are the current plateau in data processing language development really pays off when the employed metadata is stored in a repository. While any metadata is an improvement over no metadata, the form and structure of metadata is essential to determining its reusability.

[15] To be truly effective, the repository must be present both as an enterprise wide-resource and as persistent tables within the ANSI SQL schema information tables. Under the first form enterprise wide consistency and re-use is operative. Under the second form, as ANSI SQL schema information tables, the repository's metadata is available to the DBMS for automatic and pervasive use and control over all definition, use, manipulation, and elimination of database objects through all end-user languages (3GL or 4GL).

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

3.2 Information Systems Evolution

Information systems evolution seem to have gone full cycle. And with the Internet, maybe another half-cycle. From one system per application to few for the whole organization back to one for each application, and now, via the Internet, a moderate quantity for the whole organization. Each half-cycle was generally caused by either an un-binding of a technology, or the emergence of computing power for the end-users.

In the Sixties through the early Seventies, there were as many different data processing systems as there were applications. With the advent of DBMS in the middle Seventies, the quantity of data processing systems greatly reduced. Large scale databases were designed to contain hundreds of record types or tables that would serve hundreds to thousands of users. But because the quantity of time to develop and maintain these applications grew faster than the productivity improvements in systems development, the cost of applications was perceived as being out of control. Complicating this out-of-control perception even further was the increasing cost of the hardware (mainframe and telecommunications). Then, in the early Eighties, the PC arrived. Closely behind the PC was the development of word processors, spread-sheets, and PC based DBMSs. Users discovered that they no longer needed centralized data processing. While the quantity of PCs grew from a few in the corporation to at least one on ever desk, so also increased the disintegration of corporate wide data semantics.

Under the mainframe systems environment the answers to the questions in Figures 1.1 and 1.2 were clearly in column 1. There was both centralized semantics and centralized storage for data, and centralized development control and centralized execution control for processes. Under this paradigm there was never the opportunity for more than one version of the truth.

Under the PC-anarchy systems environment the answers to these same questions swung completely to the right. There was decentralized semantics and decentralized storage control for data, and decentralized development control and decentralized execution control for program development and execution. Under this paradigm there could never be the opportunity for one version of the truth.

Corporate reporting of business statistics, which were initially in total disarray during the pre-DBMS days because of no centralization and computing were again in total disarray because there were DBMSs and development environments on every desktop. What went around came around.

Client/server was proposed to be the “silver bullet.” Client/server is however more complicated than what it ultimately replaced--mainframe computers. It is more complicated because clients and servers can be from different hardware manufacturers, run by different operating systems, and be operating under DBMSs from different vendors. All that required interaction of computers. To keep peace on the network requires sophisticated coordination that can only be brought about by a single vendor, or through ANSI standards. There are however, no ANSI standards in this area.

Client/server comes in four flavors:

- IQ = 0 clients (formerly known as time-sharing)
- Uploading then server processing

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Uploading data and interactive server based processing, and down loading with client processing

- Cooperative processing between clients and servers

Given the lack of ANSI standards, for many of the client/server components, the fourth level of client/server can not be achieved unless all the clients and servers are operating through the same vendor's DBMS, that is, for example, all Oracle, Microsoft, Sybase, or Informix. The best that can be achieved through multiple vendors is the third level. Most organizations in the middle 1990s strived for the third level. Even with the third level of client/server, there is a great likely hood that there will be different brands and models of server, operating system, DBMS vendor, and application development language. Because of all these differences, the need for vendor (hardware or software) independent application specifications and adherence to ANSI standards is essential for survival.

Achieving the third level of client server computing requires either one server for all applications (essentially a mainframe) or one server for each application or application class, or, if the application is too large for a single server then multiple servers for each application or application class. This later case either means distributed database or distributed processing. In both cases, complete coordination requires centralized application development, or coordinated application development through a centralized repository of business semantics. As with the section above on languages, the form and structure of metadata is essential to determining its value in the development of different applications on the same server or of the development of the same application on different servers.

The fourth level started to appear in the late 1990s. Client systems collected information, performed local edits either through locally existing edit and validation tables, and then more intensive edits through access to main-frame or server tables, and then finally transaction permanent storage on the main-frame or server. Under this scenario the client machine was said to be "fat." That is, it contained the vast majority of the heavy lifting logic. The server system and the DBMS was "thin" in that it was merely employed to receive, edit, and store transactions, or conversely, to retrieve and ship transaction data to the client.

Today, there is an increased quantity of "thin" clients. That is because the server contains a more significant quantity of the application programming logic. When this is done the software maintenance on the client is much less. In order to have a thin client the server's DBMS has to be more robust with a greater quantity of centralized logic. An extension of this thin-client is the use of the Internet to capture, ship, and display the results of Internet-based transactions.

3.3 Hardware and Operating Systems Evolution

Today, single user, personal computer (PC) based systems are outdated. The requirement to share data is impossible to ignore. Shared data can occur if and only if there exists formally defined databases, commercial off the shelf (COTS) software to control it, and LAN/WAN, Intra- and Inter-nets = to transmit it.. The COTS software, in this case, are database management systems (DBMS). Robust, sophisticated DBMSs did not exist on single-user PCS or local area network (LAN) based PCs till the early 1960S. Thus, small and medium sized enterprises are

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

now ready for a new generation of DBMS-based data sharing, cooperative work computing technology. This technology is based on a client/server paradigm and is serviced through local and wide area networks, or through intra- and inter-nets. Because the architecture of this systems environment is so different from the single-user PC environment, the only thing salvageable is data; all else has to be redone. Figure 3.1 illustrates a LAN with a series of clients, PCs that are IBM compatible, such as COMPACT, and a server that is supporting several databases.

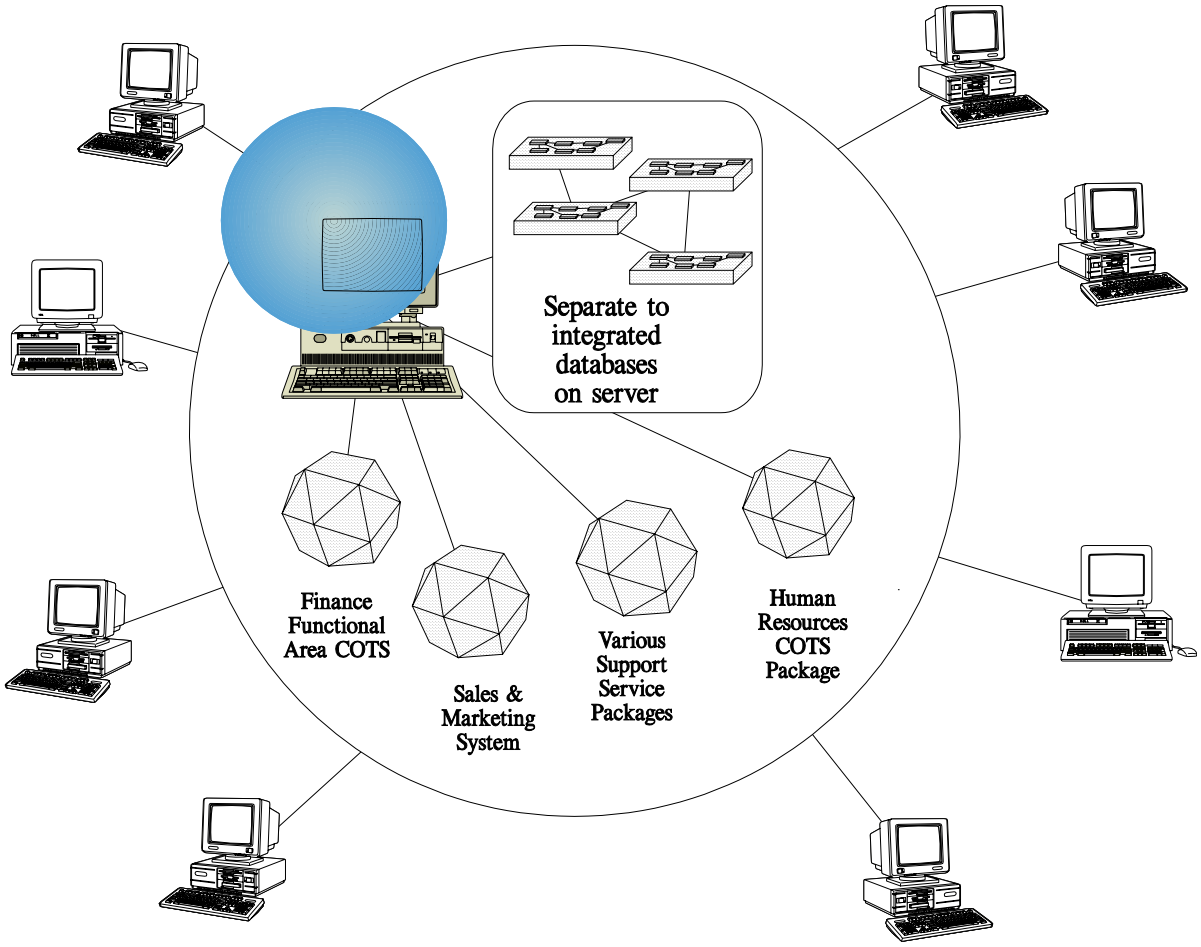


Figure 3.1 Client/Server Computing Environment.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Today, as well, multi-user, main-frame based systems are becoming outdated. The need to down-size (literally become smaller), and to provide local development autonomy is impossible to ignore. While shared data is the norm for main-frame based systems, the existing set of systems development tools, end-user interfaces, and the never ending backlog of system enhancement requests have caused the main-frame environment to be seen as antiquated, and unable to keep up with the dynamic, changing needs. Thus, the large systems are ready for a new generation of systems. The characteristics of these new systems are:

- Client/server based (through LAN/WAN, or through Intra- and Inter-nets),
- The data is maintained by DBMSs, and
- End-user terminals, which are invariably PCs, being used for localized data entry and validation. Some hand-held and Internet connected PDAs (personal digital assistants) are now able to run full O/Ss and applications.

Again, because the architecture of this new systems environment is so different from the main-frame architectures, the only thing salvageable is data; all else has to be redone.

In summary, all ranges of systems, from small to large are need to change. Small and medium systems are changing towards larger *client/server systems supported by networks (LANs (local area networks), WANs* (wide area network), *intranets and internets* to better communicate and share data. Large systems are changing towards smaller *client/server systems supported by these same classes of networks* to develop end-user systems more quickly and that serve special end-user needs. Both environments, small to medium and medium to large have arrived at the same technology answer, but for different reasons.

Figure 3.2 illustrates the medium to large environment in which there is a network to handle the traffic to one or more servers as well as integration with legacy/new applications that operate under mainframes. There could be one server for all database applications, that is, one for human resources, finance, and support services, or there could be one server for each application type. The decision depends on a number of issues such as required size, the need for data and process integration, the availability of discrete COTS from different vendors for each application or a large application suite from a single that addresses all applications.

Computing environments of different sizes traditionally have very different hardware and software architectures. The mere thought of developing a PC based system and scaling it up to a main-frame system surely demonstrates one's naivety. Conversely, developing a main-frame based system and scaling it downwards towards the single-user PC is like trying to run a Greyhound bus go-cart track.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

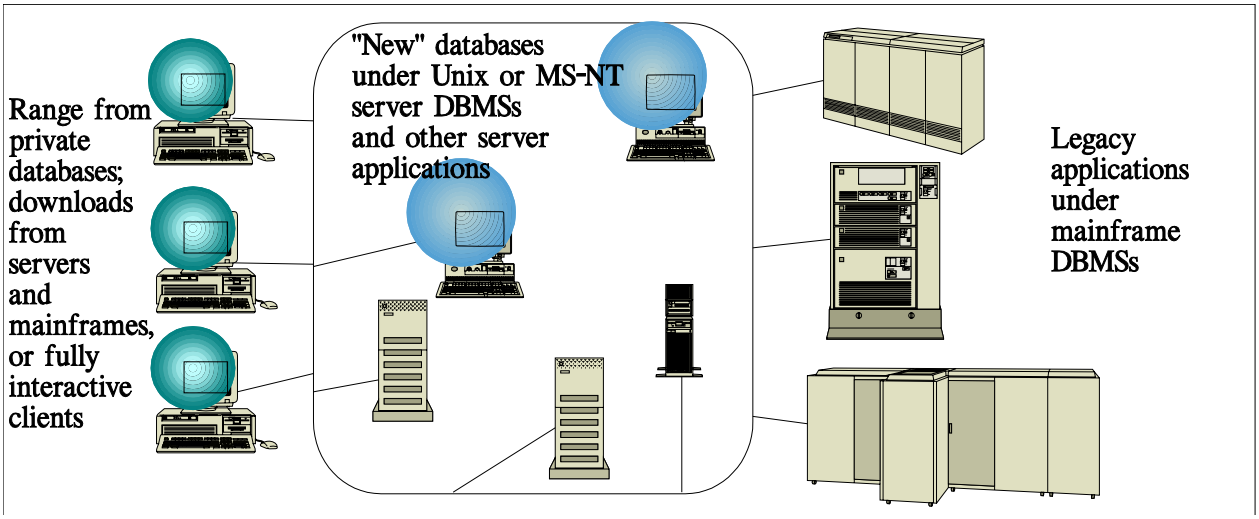


Figure 3.2. Medium to large computing environment.

Because PC and mainframe software and hardware architectures are so different, the system development life cycles have been quite different. PC system life cycles are characterized as "hack-away" a month or two until a program (usually 1 per system) emerges, with usually one or just few staff members. Detailed design and implementation documentation--what's that? User manuals are generally well done, however.

Mainframe system life cycles are characterized by very careful planning, multiple-year, multiple-team (4-8 members each) that have detailed work plans, sophisticated products, intensive, thorough testing, and volumes and volumes of documentation.

Today however, small to medium and medium to large hardware and software architectures are merging towards client/server systems. The only critical difference is one of size and power. Common are:

- Database management systems for data definition, sharing, storage, updating and reporting all using the ANSI standard database language, SQL for their definition, updating, and reporting operations.

- Intelligent terminals, usually PCS that are self-contained, Windows/GUI (graphic user interface) displays, that contain central processing units and multiple gigabyte storage for data entry, editing, and local data staging/storage.

- Data access and storage platforms that house DBMSs and can process shared data requests in small configurations (25 simultaneous users, one 100 MIPS (million instructions per second) processor, 100 megabytes (million) memory, and 6-10 gigabytes storage) to large configurations (1000s of simultaneous users, 16-64 100 MIPS processors, 500-1,000 megabytes of shared memory, and 60-100 gigabytes storage)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Local area networks serving 10-50 users to combinations of wide-area networks (WANs) and LANS to serve thousands of users to use of Intra-nets and Internets to serve an almost unlimited quantity of users.

3.4 Peopleware Evolution

A full complement of I-CASE (integrated computer aided software engineering) tools with repository and code generators costs less than $125,000 (hardware, software, and maintenance) for 10 to 15 systems analysts and programmers. Assuming a life span of five years, then the cost per user hour is about $4. A billable staff hour is about $100 Without this tool suite, the productivity decreases by 40%. That increases the effective staff hour cost to $140. Simply, an investment of $4 saves about $40. Or, put another way, with CASE, repository, and code generators the same work can be accomplished with 40% less staff.

Given the dramatically increased velocity by which applications and systems can be developed staff must either be eminently qualified in application areas, or they must have access to ready-to-use database object templates from mission description areas that can be combined into systems and then fed to code generators.

Systems development must become similar to the personal computer (PC) building process. A PC consists of less than 20 discrete component assemblies that can be combined into a working computer in a matter of hours. Businesses that do not have this type of systems development environment will probably fail because information compares to money as the most important corporate asset.

3.5 Outsourcing: Scourge or Salvation

Organizations, attempting to either cut costs or to streamline their information systems development have been outsourcing. That is, contracting out the organization's project leaders, systems' analysts, programmers, and data processing operations. This however, is like "getting rid of centuries accumulated know-how[16]." This quote from Strassman really gets to the core of the problem with outsourcing. Accumulated business wisdom is being tossed out the window.

Database is policy organization and management. DBMS is the technology implementation of database. Given that data is executed policy, then data's specification is the technology-based specification of the corporation's policy. Databases are the corporations long-term policy-based memory. How can businesses out-source the definition of its corporate memory to individuals who have no business-wisdom? That is business suicide.

If outsourcing is inevitable then businesses must demand that every data processing task be accomplished such that the delivered system be free from all outsourcing vendor's proprietary CASE, that the computer languages be ANSI standard, that the DBMS be ANSI standard SQL, and that all documentation, from initial business policy and procedure down through detailed

[16] This quote was taken from an article by Paul Strassman (former CIO of major U.S. corporations and the U.S. DoD) entitled *Outsourcing a Game For Losers*, August 21, 1995 in Computer World.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

system documentation be delivered in a repository that allows full inspection, quality control, configuration management, and outsourcer independent evolution and maintenance.

Stories abound about companies who were so locked into the outsourcer that the business was completely unable to evolve because it had trashed all its professional staff and no longer had any data processing or systems documentation. Stories also circulate about companies who have outsourced overseas. While their costs were immediately reduced, when it came time to upgrade they couldn't. Nobody knew the what nor the how!

Outsourcing firms make their money NOT on problems avoided, but on problems solved. How can you bill for time not expended? Outsourcers, to keep their costs low, only provide the least qualified staff they can get-away with. That is, minimum business knowledge, minimum prior project experience, etc. In addition to assigning the least costly staff, outsourcing firms generally do not create the outsourcer-independent critical meta-data that fully frame the business policy and procedure from which databases and systems are built. That's because the outsourcing firm is being paid solely on the quantity of systems produced, enhancements built, and/or problems repaired. "Damn the analysis and design, we're only being paid to code," one outsourcer was heard to say!

If outsourcing firms were true engineers, they would be earning their keep on the basis of problems avoided, solutions that have flexibility, and systems that are maintainable for a fraction of their cost rather than multipliers. But, if an outsourcing firm did the latter rather than the former, they would not be able to keep THEIR costs low. AND, their billing would fall off because they would not be needed. Clearly such behavior would not be in the interest of the outsourcer's owners/stockholders!

What is the benefit to outsourcing? Clearly, one main reason is to reduce the life-time cost of an employee. Here's an example in 1995 dollars. To wit:

> If average salary of an data processing professional is $60,000 and if G&A and overhead multiplier is 2.5, then the per year staff cost is $150,000. From 22 to 55 the total cost of the data processing professional is $4,950,000. If the staffer then draws a retirement of $40,000 and lives 30 more years, the retirement cost is 1,200,000. The total staff cost is thus, $6,150,000.
>
> Alternatively, if a data processing professional is contracted through an outsourcer at $100/hour then the annual cost is $184,000. Over 30 years the total cost is $5,520,000. The possible savings approximate $600,000 per person. In addition, business' can trade in the outsourcing staffer every year for better and newer models.

Saving money is however is a subterfuge because with outsourcing, there is no real retention of corporate wisdom through the building of a corporate metadata repository. The value of this however, is both difficult to quantify and at times, completely unvalued because there is no acceptance that the residual products of a data processing professional is an asset. Sadly, the only identified and valued asset is the system/program produced. Businesses have no understanding of value derived from leveraging long-term corporate knowledge into building more and better systems/programs at both an increased quality and faster rate.

To make the outsourcer's work valuable, a meta-data repository must be fully defined and put into place as a minimally acceptable work environment for the outsourcer's effort. The

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Army in the mid Eighties contracted for the development of a world wide system to collect maintenance data on targeted sets of material to predict reliability, availability, and maintainability. Ten systems had to be built in less time than the previous two. The design and build was accomplished on-time and for less funds than were predicted simply because of the repository which stored the reusable metadata. Instead of costing 10 * 100%, or $5 million, the systems cost 64% less, or $1.8 million. A savings of $3.2 million. The reasons were simple:

- A well defined database oriented methodology that clearly set down procedures, specified products, and quality reviews.

- A repository that enabled the continued use and reuse of metadata to accelerate development and maintenance.

The Army had outsourced the entire data collection, analysis, and reporting effort to PECO, an Iowa contractor specializing in logistics analysis and planning. PECO, in turn, contracted out all data processing system analysis, design, implementation to other contractors. PECO operated the logistics system itself. PECO also kept the repository "inside" their domain. This enabled PECO to be independent of contractor vagaries. PECO exerted positive control over all planning, quality control, all phases of testing, and complete design through user documentation.

Within a year PECO no longer needed its contractors because all the systems' metadata was designed and implemented through CASE, repositories, and code generators that were owned by PECO. The Army, on the other hand was not as good a buyer as PECO. When PECO's contract ended, PECO, not the Army owned the repository. The Army had contracted for the delivery of the software systems and all pertinent documentation. PECO complied. A small "dump truck" of documents were delivered along with all source and executable forms of software. Not delivered were the two repository-based factories: analysis and design, and software implementation and maintenance. While the source and executable software and documentation were the "golden eggs," the two repository-based factories were the "goose."

The Army did not comprehend the value and the critical role of the repository. Without the repository the new contractor could not create new systems nor evolve existing systems. Slowly, but surely the entire systems environment crumbled and was discontinued. Without the CASE, repository, and code generator environment there were no more golden eggs; the goose was gone. Scourge or salvation? It depends significantly on who defines, owns, and controls the assets. That is, the CASE, repository, and code generator environment. To PECO, outsourcing was salvation. To the Army it was a scourge.

3.6 World-wide Enterprises

Enterprises are commonly world wide with heterogeneous hardware, operating systems, DBMSs, and application software packages. When the company was small it contained its own data processing organization. The data processing organization supported business' five critical functions: human resource management, sales and marketing, manufacturing, logistics, and customer management.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

As new corporate units were added to make a country wide organization, either each new unit added their own data processing organization to accomplish their own style of the five business functions or they employed remote computing services from an increasingly larger centralized data processing organization. Regardless of choice, they accomplished both their own internal needs and also supplied data to the multiple business unit management organization, which needed information about each business unit and also needed to create consolidated reports and conduct their own special studies.

Finally, when corporations became international the data processing organizations not only expanded beyond country borders they also had to deal with these five fundamental business functions from within different cultures. Each culture has its own style for names, the important data to collect and maintain, and the ways and methods of evaluating profit and progress. Regardless of these significant cultural differences, there is still the need to generate data for world wide corporate management reporting and special studies.

From the 1970s through the late 1980s, the choice of sophisticated computing technology for accomplishing business applications was simple: the mainframe. But, with the advent of the personal computer, which grew into larger and larger capacity and performance servers, the whole nature of data processing changed from very large main computers all executing the same set of programs to a very large set of small computers all executing different programs. In the mainframe days, it was never a matter of dealing with more than one set of semantics because there could never be more than one. Now, because the mainframe has largely disappeared and because of the rise of the individual's ability to easily create whole complex business systems, the very ability to have a single set of consistent semantics has largely disappeared. Businesses have now recognized this terrible situation and are desperately striving to return to a single set of business semantics. database objects is a route to that objective.

3.7 Enterprise Data Architectures Evolution

There are five distinct data architectures that must be implemented within a world-wide enterprise. These are:

- Reference data databases
- Original data collection databases
- Transaction data staging area databases
- Integrated subject area databases
- Warehouse databases

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Figure 3.3 presents all five of these database classes[17] within the context of an overall enterprise data architecture. At the bottom of each of the second to the fifth data class are its most significant characteristics.

3.7.1 Reference Data

Reference data is that set of data that is commonly employed across the other different classes of data. Examples of reference data include sets of data like:

- Gender codes and names
- State codes and names
- Country codes and names
- Units of measure and names
- Currency codes, denominations, and names
- Standard address parts such as street (ST), road (RD), etc.
- Manufacturing parts, names, and descriptions

Reference data has distinct characteristics such as:

- Persistence
- Employed in longitudinal policy formulation
- Impacts multiple policy areas
- Enables fact discrimination and/or selection
- Break points for fact aggregation and statistical analysis
- Requires higher level of management approval
- Causes multiple negative impacts when changed
- Is not easily melded with other strategic data (zip | telephone = ???)
- Gives meaning to quantities (physical dimension, time dimensions)

[17] Three of these five classes are a reformulation of the four classes set forth in Enterprise Database in a Client Server Environment (John Wiley & Sons, 1994). In that book, the four classes are: data collection, operational, warehouse, and specialized study. These four are reclassified in this book as: original data collection, integrated, and the two types of warehouse, wholesale and retail warehouse, respectively. The transaction data staging area database class is a database class that preserves the semantic independence of both the original data collection database applications and the integrated subject area database applications. Reference data, sometimes called strategic data is a new formulation brought about by factoring the common, critically persistent data from the other four data classes. The five database classes in this book can be inferred from Building a Data Warehouse by Vidette Poe (Prentice Hall, 1996).

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

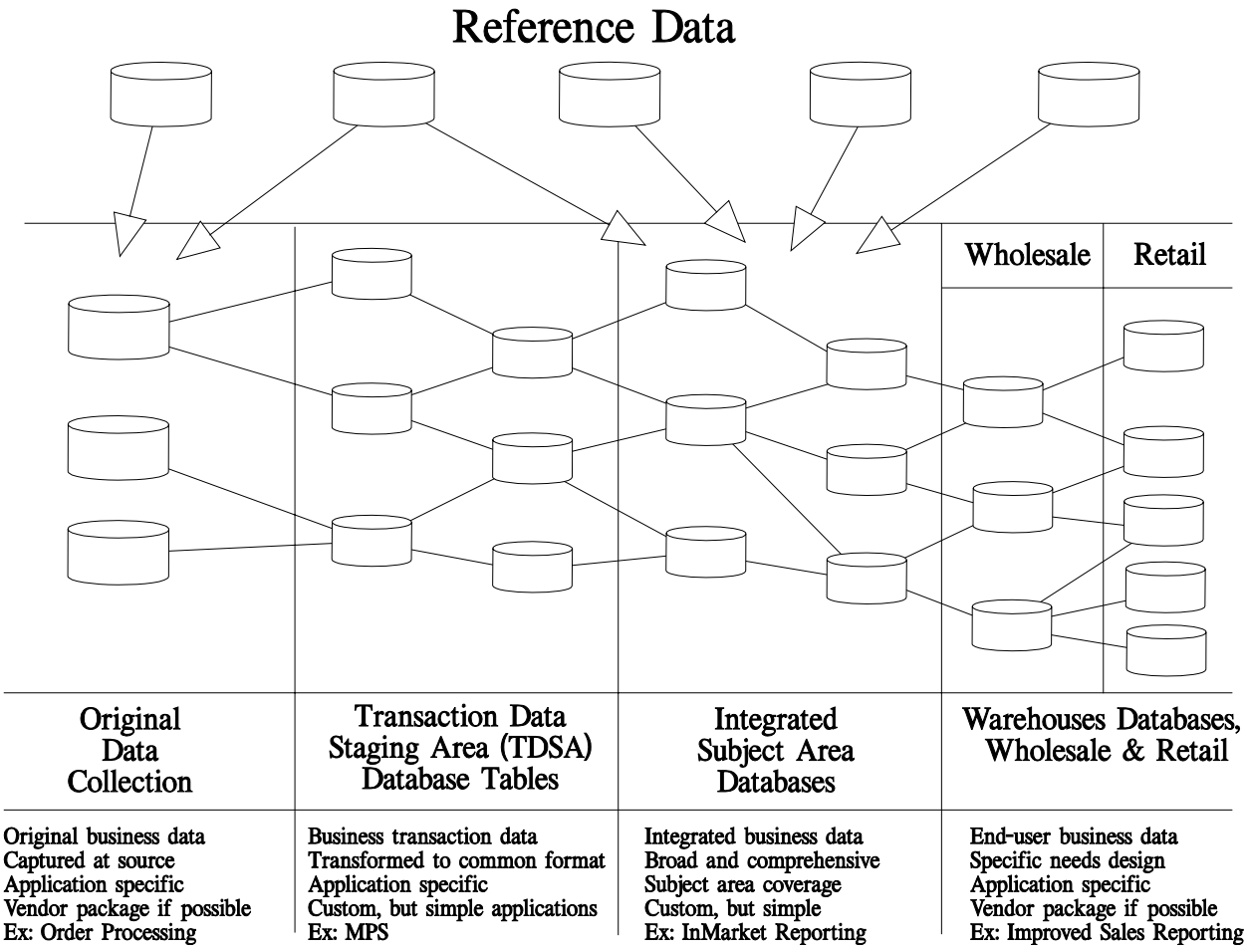


Figure 3.3. Data architecture classes.

Reference data is established through formally created projects that cause the reference data to be created, stored, reported and maintained. Reference data projects are critical to the business because business' fact data that is stored in the operational data collection databases is categorized, summarized, validated, and reported by means of the reference data.

Supporting each type of reference data are full policy definitions and descriptions. Because of enforced reference data, a corporation can enjoy enterprise-wide semantics. The reference data must also support the full set of valid and invalid values including alternatives for different countries and languages. The key issue in representing multiple language reference data is whether all the short values, for example Female or Hembra is encoded as an F, or whether all encoded values are represented as numbers, for example, "1".

A critical characteristic of reference data is that it must support evolving data history. Reference data supported by dates tracks data value changes. Business data can thus be interpreted under either the current set of meanings or their original meanings.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Reference data is commonly seen as only business data that is used for references (gender codes and names). Reference data also embraces metadata, that is, the standard names and definitions for business data, including for example the definition of Country Code, its length, and the ISO standard from which its name, definition, and value set is derived. Metadata embraces not just data fields but also database tables, databases, subject areas, programs, systems, etc. All metadata is stored and maintained in a repository and is the subject of Chapter 6 of this book.

3.7.2 Original Data Collection Databases

Original data collection databases are those that contain all business facts. Prior to being stored in an original data collection database, the business data is manual, that is, customer orders, employment applications, hours worked but not yet recorded, and the like. The characteristics of operational data are that it's detailed atomic data, not summarized data. The data is accurate as of the last update. The databases that contain the databases are stable and long lasting with normalized data structures. These databases make heavy use of reference data to ensure that it is understood across all users. Because of reference data and because of very extensive integrity checking no invalid data should ever be allowed.

Operational data collection databases are generally tuned for the transaction capture, storage and update of data. Correctness and integrity is of paramount importance. Performance is clearly secondary. The databases are oriented towards specific applications such as order capture, order processing, staff time recording and reporting, and payroll. Because these applications are very narrow in scope, they are commonly acquired as commercial off the shelf (COTS) packages.

Operational data collection databases must have 24 hours and 7 days availability as they are the source of the company's business data. Simply put, these databases support the company's day-to-day operations.

3.7.3 Transaction Data Staging Area Databases

Transaction data staging area databases are the beginning of integrated data semantics for the entire enterprise. While individual original data collection database applications may have both specialized and localized semantics. These localized semantics may exist throughout the enterprise because of different languages, laws, and existing sets of application packages that existed prior to migrating to an enterprise wide data architecture.

To handle all these situations, data translators are created that take localized data values and meanings and translate them to enterprise-wide values and meanings. The transaction data, now understood through enterprise-wide semantics is placed in a staging area for a specific period of time. At the end of the specified time period, the data is dropped from the staging area. Applications requiring data from the staging area only have to know where it is and the set of semantics that govern it.

The fundamental role of data within the transaction data staging area is to relieve the individual original data collection databases from having to contain enterprise wide semantics

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

and to relieve the integrated subject area databases from having to know the location of the individual original data collection database applications and from having to know the values and semantics for each original data collection database.

While the transaction data staging area database applications represent another class of applications, the cost of its design, implementation and maintenance is more than offset by the reduced cost of having separately developed and maintained value and semantic translators in each of the original data collection database applications that would "push" data to the integrated subject database application, or an analogous class of software in the integrated subject area database applications to "pull" data from the original data collection database applications. Additionally, this intermediate layer of value and semantic translators enable the organization to change the original data collection database applications on an as needed basis to accommodate new DBMS, new operating system, and different classes of hardware that may be required to support organizational units of different sizes and locations.

3.7.4 Integrated Subject Area Databases

An integrated subject area database, sometimes called the operational data store (ODS)[18] embraces a wide area of a business such as all finance, all human resources, all sales, orders and marketing. The data for these integrated databases is taken from the various transaction data staging area databases and is combined into one overall database. Figure 3.4 illustrates the point. The database in the center is an integrated database about the subject area: sales and customer management. This database is a amalgamation of data from the databases:

- Sales organization management database that defines and manages all sales regions, districts and territories
- Human resources database that provides the proper information about each salespersons who is assigned to a territory
- Customer management database who ensures that all the customers are properly identified, classified, and managed
- Order processing database which receives, tracks, and performs all order management through to actual shipment.
- Product inventory management database which receives product into product warehouses, fills orders, and begins the process of product distribution
- Manufacturing database which transforms raw products into finished goods

[18] Bill Inmon's book, *Building the Operational Data Store* (John Wiley & Son, 1996) clearly defines this database type and places it within the overall data architecture necessary for organizations attempting world-wide, heterogeneous database environments.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Product distribution database which receives filled orders from inventory and delivers them to the customers.

- Accounts payable and receivable databases which financially accounts for orders and invoicing

All these original data collection systems provide transaction data to the enterprise transaction data staging areas which are represented in Figure 3.4 through the "lines" that connect the databases. The data loading and update systems that feed the integrated databases take data from the transaction data staging areas and load and/or update the integrated databases.

The names of these eight systems might imply a corporate organization structure. That would be an unfortunate conclusion. The systems that could logically belong in one enterprise organization, Sales and Marketing would be:

- Sales organization management
- Customer management
- Order processing

Another organization, manufacturing, inventory and distribution could undertake all three of those applications. Human resources is likely its own organization, and finally, accounts payable and receivable would be part of the finance department.

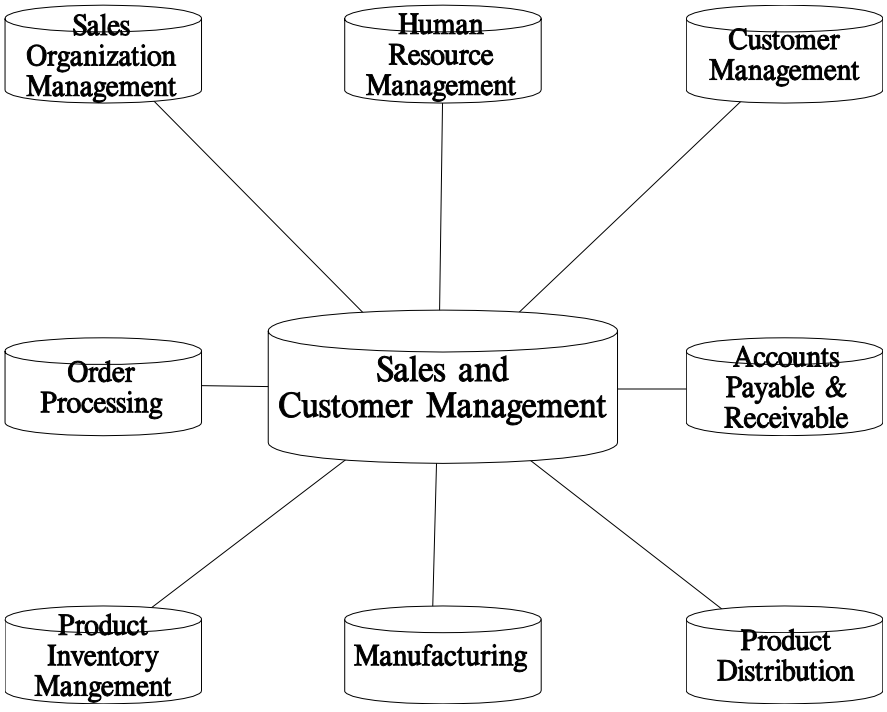


Figure 3.4. Collection of functionally specific databases feeding a data warehouse.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

In addition to the data integration route, some data from one database may certainly be required in others. For example, sales organization management assigns customers and sales persons to specific sales organizations such as regions, districts and territories. The data from these "cousin" applications should be available in a read-only manner. Whether the references from one the sales organization management database that lists the current district manager as Bruce Baine does so through a "hot-link" or through a once a week integrity check routine is a matter of technology, not of fundamental design.

The key data characteristics of integrated subject area database are that its data is at the detail level. The data may be lightly summarized by storing invoice totals, customer sales-to-date information and the like. The data, because of the transaction data staging areas is either current or nearly current. The database supports either endless histories or rolling histories. A rolling history might be the annual sales for prior years 5 and 4, quarterly sales for year 3, monthly sales for prior year 2, and daily order, shipment, and invoice line items for the current year.

The key uses of the integrated subject area databases are detailed level analysis of current and historical trends, up to the minute decisions that require a complete picture. For example, a sales call might require:

- Customer orders,
- Sales organization assignments,
- Payables and receivables history,
- Recent history of sales,
- Deliveries and returns.

Integrated subject area databases may be interconnected. That is, one integrated subject area database may be the source of data for other integrated subject area databases. For example, if there was a integrated subject area database for world wide finances, summarized data from the various sales and customer management databases might be selected and forwarded onto the world-wide finance database.

Integrated subject area databases are typically very large but must provide a fast response time to ad hoc queries. Because of these characteristics, a common approach for implementing these databases is to use very large and fast servers or mainframes.

Integrated subject area databases are not however warehouse databases because of two major reasons:

- Updating other than summarization may occur. For example, a characterization of a customer's importance to a company may be stored, or a program might run through all a customer's records and assign an Key Customer Ranking code.

- All retained data is commonly at the transaction level such as order, shipment, invoice, payables, and receivables line items.

While the ideal enterprise wide data architecture is based entirely on database objects, from original data collection databases through to retail data warehouse, organizations that depend on COTS vendors for their original data collection applications cannot achieve this ideal. The transaction data staging area database systems enable organizations to dampen out the semantic

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

differences that are unavoidable through packages. Once the data is semantically homogeneous, it can be loaded into integrated subject area databases which are designed through the use of database objects.

3.7.5 Warehouse Databases

A warehouse database is a special class of databases for two reasons:

- No updating other than summarization occurs
- All data has its units and granularities synchronized

Simply put, data, as a warehouse item represents a finished good. No more "manufacturing" or "finishing" is going on. In the case of data that means no more updating. The data (warehouse item) may be selected according to a specific order and may even be palletized with other items of the same or different types and sent to a more localized distribution point. Because of these similarities with product warehouse the data warehouse concept arose.

A very early proponent of data warehouses, Bill Inmon (Prism Solutions of Littleton, Colorado) has written a number of books on data warehouses[19] defines it to be a collection of integrated, subject-oriented databases designed to support the decision support function, where each unit of data is relevant to some moment in time. The data warehouse thus contains atomic and lightly summarized data. Ron Ross, another data warehouse proponent defines a data warehouse to be a structured database that stores consolidated data for decision support from production-level files or databases. A more recent definition for warehouse comes from Vidette Poe who defines a data warehouse as the [data] foundation of a decision support system (Building a Data Warehouse, Prentice Hall, 1996).

Data warehouses come in two flavors: wholesale and retail. The critical differences between the two is in the area of size and scope. A retail data warehouse is clearly smaller because it contains less data. Less both in terms of rows and also scope of tables. For example, there may be a customer profile retail data warehouse that contains all relevant customer information for the Yellow Knife sales territory. The wholesale data warehouse might be the one that contains all the customer sales information for all the Northwest Territories of Canada.

A retail data warehouse also differs in scope in that its focus may be narrower than that of its source wholesale warehouse. For example, all the customer profile information for hospital customers for a particular segment of a product line versus all the customers for all the product lines.

Because warehouse databases address well bounded subject areas as do integrated subject area databases they are commonly know by the data they keep, e.g., Human Resources, Sales and Customer Management, Inventory Management, Finance, and Real Property Management.

A common technique employed in the development of retail data warehouses is to perform extensive summarizations that are related to less extensive (more detail) summaries that

[19] Bills books in this area include: *Building the Data Warehouse* (Wiley & QED, 1993) and *Using the Data Warehouse* (John Wiley & Sons, 1994)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

are in turn finally related to data that is almost transaction level data. Specialized query and reporting tools then enable the end user to begin at the highest summary level and to "drill-down" to the ultimate level of detail. This technique is sometimes called "data mining."

The software systems that perform this type of analysis and reporting are called on-line analytic processors (OLAP). Traditional relational DBMS vendors such as Oracle have developed specialized report writers that also perform on-line analytic processing. The ANSI database languages committee, H2, has also developed a whole library of SQL syntax that supports data warehouse data structure design and also data processing. These facilities thus off-load these activities from every data warehouse database application to the centralized DBMS.

Warehouse databases often derive their data from multiple integrated subject area databases. A given data item, for example, a travel expense voucher might have its data end up in multiple data warehouse databases dealing with finance, projects, airlines or lodging. Because of this redundancy, it is impossible to perform correct updates. If warehouse data is observed to be wrong, then the original source data must be found and updated. Once the warehouse is refreshed or regenerated then it will be correct.

The principle reason for warehouse databases is to support business data analyses such as trends and forecasting. These analyses are possible because the data warehouse's database can include data from multiple integrated subject area databases. As historical data grows it may be turned into high level summaries similar to the five years of history data described above. The main set of data warehouse users are the managerial community who both set broad direction and positioning, and formulate and assess long term decisions. Because of this type of use, data warehouse databases are generally not on a business' critical path, thus, while the need for data selection and processing power is extensive, response time is clearly not required to be "subsecond" as the decisions the data supports may take months to formulate, adjudicate, and promulgate.

3.8 Data Architecture Example

Figure 3.5 presents a set of data processing applications and two sets of data warehouses. Each business function application (e.g., sales and marketing, or human resources) might be operated through a separate application package that was procured from a different vendor. In order to merge and handle these semantically disparate sets of data, each application package needs to create data extracts that flow into the organization's data warehouses. In the example in Figure 3.5, there are three: personnel deployment, organizational effectiveness, and product profitability. In this situation, since the individual application packages have their own semantics, the only location for specifying database objects is in the wholesale warehouses. Collectively these three wholesale warehouses have to contain the full specifications of all the business' database objects.

Figure 3.6 presents the data architecture of a country wide organization that has multiple business units. Each business unit (left column of three blocks) contains their own set of data processing activities (represented by a graphically reduced version of Figure 3.5). To accomplish multiple business unit processing, these three sets of data processing capabilities produce data that is stored in the middle box of the business' five basic functions. Each middle-box database's design contains the business' database objects. These five databases then support the generation

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

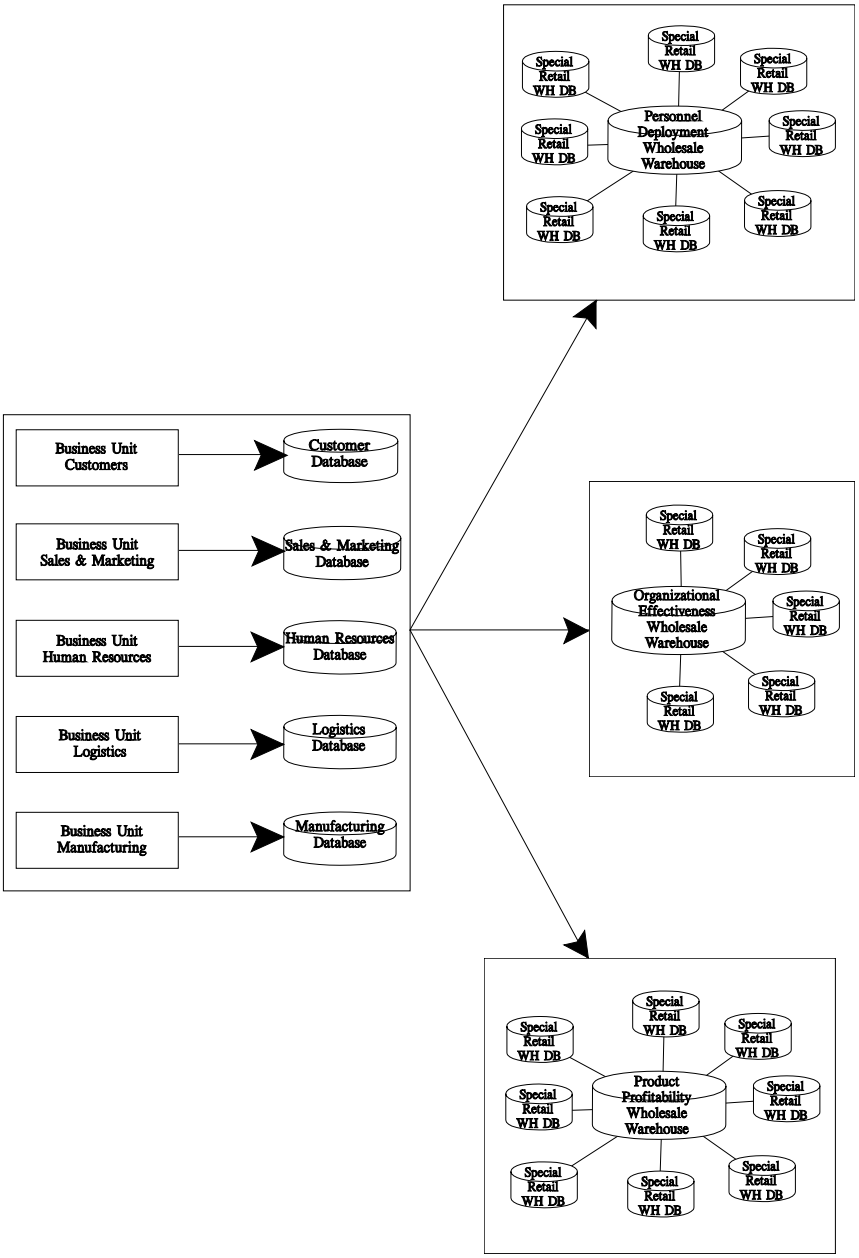


Figure 3.5. Data Warehouses supporting business-unit database collections.

of data for the three inter-business unit wholesale warehouses for personnel deployment, organization effectiveness, and product profitability. Because of database objects, business characteristics, trends, and plans from all the business units can be consolidated and shared. Each business unit is still however free to develop their own intra business unit application systems. Apart from all the other reasons for database objects, that is, ease of business specification, data processing system accomplishment evolution and maintenance, database objects are only absolutely necessary to support data sharing among the business units.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Figure 3.7 presents the data architecture for a world wide organization. The rationale for this architecture is merely an extension of the country-wide data architecture. Each country's own multiple business unit data processing data architecture is represented by a graphically reduced version of Figure 3.6 that is stored in each of the boxes on the left side of the diagram. The middle set of databases represent a world-wide set of data about the five critical business functions. Finally, Figure 3.7 contains the three wholesale warehouses, personnel deployment, organization effectiveness, and product profitability so that world-wide reports can be obtained, and so that country wide comparisons can be performed.

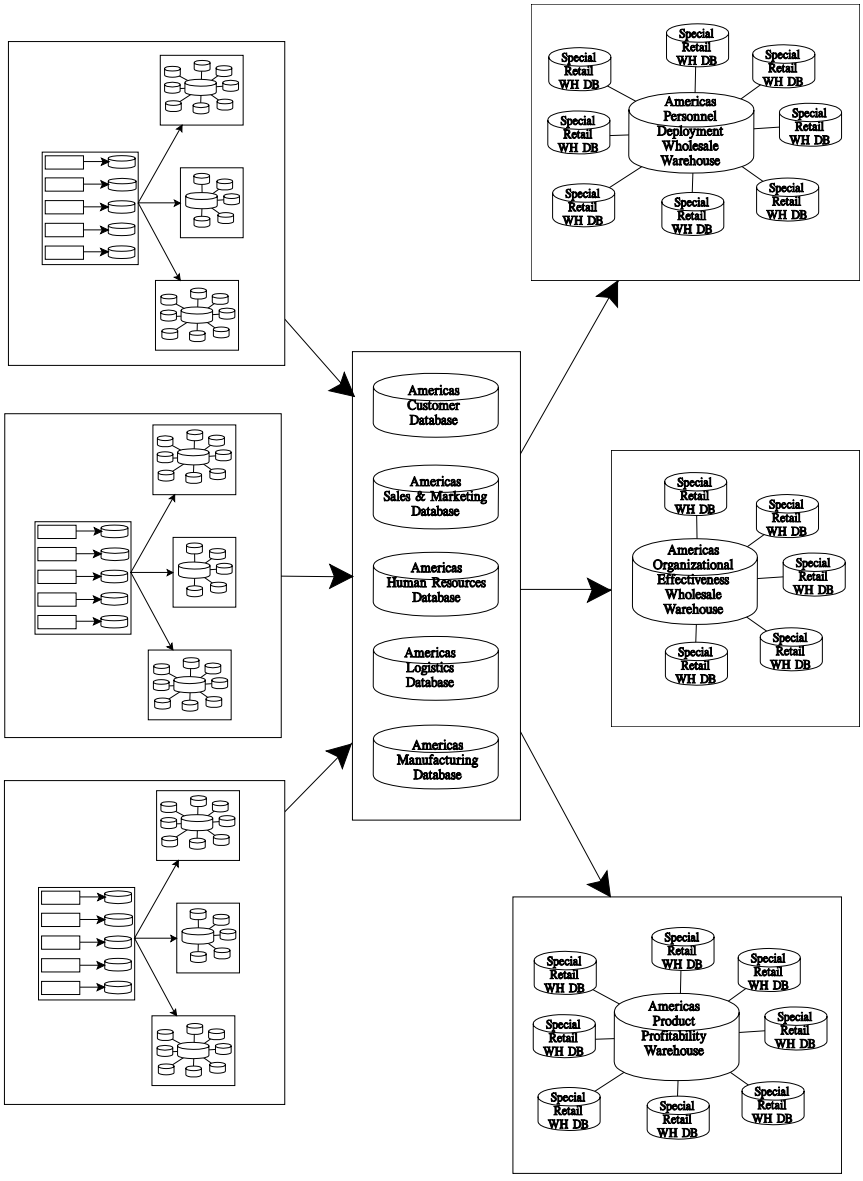


Figure 3.6. Country-wide database collection.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

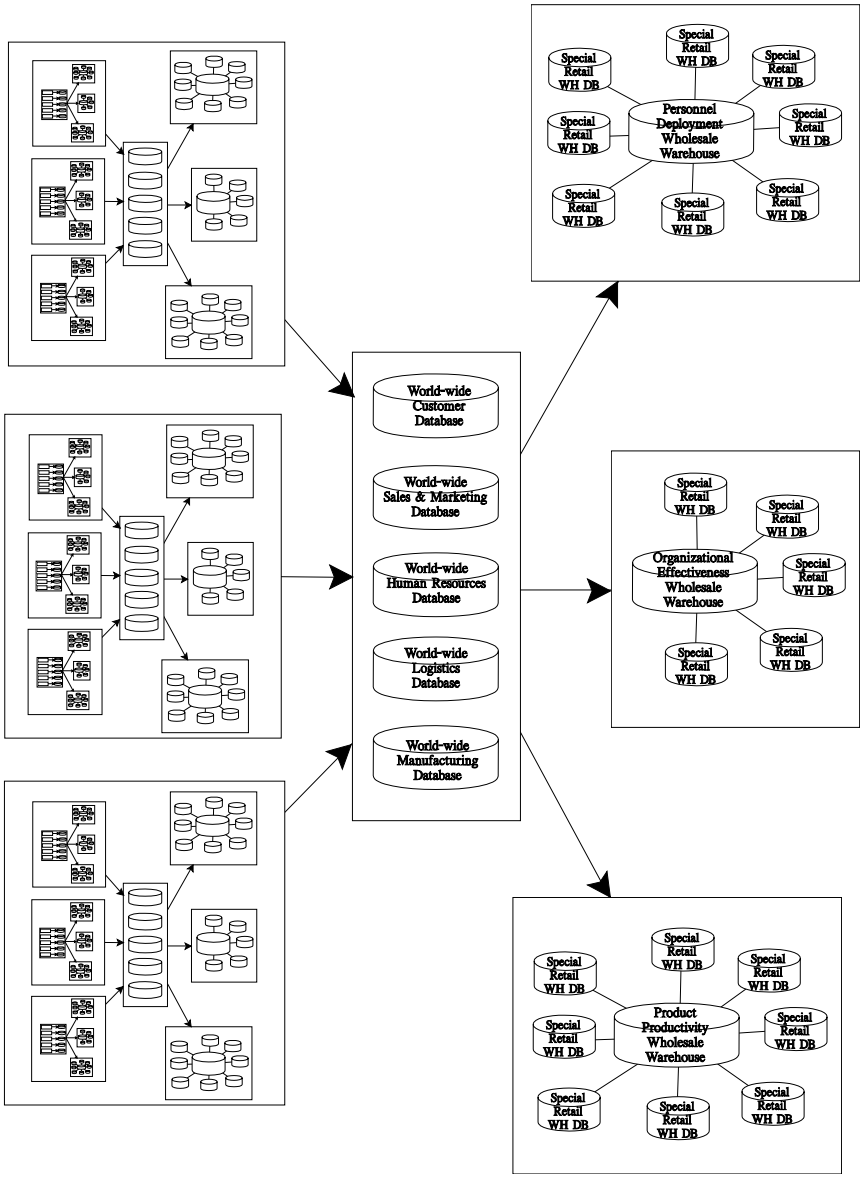


Figure 3.7. World-wide database architecture.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

With this architecture of building database objects, world-wide trends can be identified and traced down into a specific country, and then down into a specific business unit, and then if necessary, the specific operation data can be explored through the use of the individual organizations application packages. With this approach, corporations that range from multiple business units to world wide can have standard semantics represented within database objects while at the same time preserve the required individuality of business unit specific data processing needs.

Application packages to serve the needs of a business's main five functions should ideally purchased in the form of metadata that is transformed into working applications through CASE, repository, and code generators. The Oracle Corporation's suite of application packages are intended to be custom fitted to organizations and then generated. The delivered metadata is contained in the Oracle CASE dictionary is modified to accommodate the site and then fed to the Oracle CASE database and application generators. The result is working applications.

Other software corporations are also creating templates that are then used via generators to produce software package environments. The SoftVelocity Corporation of Pompano Beach, Florida has the Clarion For Windows Clarion software development environment that automatically develops fully operational Windows application packages in hours that used to take from weeks to months. Clarion For Windows is also engineered to accommodate third party corporation templates and to completely created metadata based application packages that have both the required database design (data structure and integrity constraints) and complete application logic (data entry and modification screens and reports). These metadata based application packages for example human resources, inventory, finance (GL, AP, AR, and payroll) enable its users to create the changes needed for the specific corporation and to then generate the production versions. The changes can add different fields to screens, the database design itself by adding new database object data structures, new database object modification routines, and reports.

Before corporations decide to procure completely developed application packages that fit in the short run but run out of capability in six months or a year, the alternative of buying metadata based packages that can be changed and re-changed through time to meet the ongoing needs of the business should be seriously considered. Not only are these application development environments (e.g., Oracle and Clarion) inexpensive, the metabase based application packages are competitively priced with completely developed (but unmodifyable) production application packages.

Database objects exist in two forms: the actual instances such as the employee Matt Flavin, and the metadata specification of the database objects. In general, the former are called objects and the later are called object classes. Both forms exist and both forms are real. The database object metadata is the "goose" and the database object instances are the "golden eggs." The critical aspect of database objects that must exist throughout worldwide enterprises to permit world wide exchange of database object instances is the database object metadata (specifications of data structure, database processes, database object information systems, and database object states). The critical aspects of Figures 3.5 through 3.7 that must be present prior to the sharing of database object instances is the database object metadata.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

3.9 Successful Computing Environment Summary

The main points to the preceding sections are that:

- The computer language evolution has brought us computer languages that are clearly more powerful to the extent that high level specifications can be created quickly out of reusable templates that, in turn, can cause the complete creation of whole business information systems.

- The information systems evolution has taken us one full circle and another half circle from main-frame dominated to PCs on the desk top back to robust servers, and even main-frames that are executing may virtual servers.

- The hardware and operating systems evolution has taken us from a ratio where the hardware was many times the cost of the staff to where the staff is many times the cost of the hardware.

- The people-ware evolution has taken us from where staff had many, many months to understand a business problem before attempting its automation to the point where solutions have to be learned, design, built, and deployed in no more than a few months. While this is caused mainly by the increased velocity of business changes, it is exacerbated by th high-level staff hour cost.

- The outsourcing scourge or salvation issue is one that seems to have come full circle. From centralized staff to outsourced staff or contractors to then outsourced staff located at off-shore organizations. Many organizations are now finding that such arrangements are not as business wise as first thought because of all the time and cost overhead associated with management, review, and administration.

- Finally, as the world becomes more “flat” in that everyone can be in contact with everyone else through the Internet, that the natural time-delays of data collection, storage and processing and then transmission to a “higher” level of database is becoming almost instantaneous. Thus, from a communications and interchange point of view, the world is slowly shrinking to just a “point.”

Simply stated, all these changes have caused us to need tools and environments that operate on a higher level of abstraction so that specifications and/or designs can be accomplished once and use many times. And, with the dramatic increase in the cost of staff, these tools and environments have to be easier to use, more forgiving of common mistakes, and once systems and applications are built these tools and environments must support evolution and change. And complicating the cost of staff, the increased requirement to respond to business environment changes means that whole systems have to be designed, built, tested, deployed, and maintained very quickly.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

4

DATABASE OBJECTS

Much has been written about objects. As usual, however, data-biased persons see the world inside-out from those who are process-biased. To the process-extremist, an object is a process with encased data structure components. To the data-fanatic, an object is a data structure with contained process components. Can they be reconciled? Aren't they just inside-out's of each other? No. It's the same kind of difference as exists between data flow diagrams and entity-relationship diagrams. They are NOT inside-out's of each other. They have a fundamentally different orientation.

The data driven and process driven orientations don't have to come together. Many articles about object-oriented analysis and design highlight techniques that were very popular during the days of highly structured Fortran programming; where complex systems were put together out of commonly used and reused pieces (and subroutines) of code from libraries. While those code objects are important and useful, they are not database! Nor, are they what database was EVER intended to be.

Database was always intended to be representations of coherent structures of data that matched well defined areas of policy and the supporting behavior to match well known business event states.

> *A database object is an instance of a data structure that proceeds through predefined states according to embedded process transformations.*

The database object is defined in its four database object parts: data structure, process, information system and states.

- Database Object Data Structure: the set of data structures[20] that map onto the different value sets for real world database objects such as an auto accident, vehicle and emergency medicine incident.

- Database Object Process: the set of processes that enforce the integrity of data structure fields, references between database objects and actions among contained database object tables, the proper computer-based rules governing database object table insertion, modification, and deletion. For example, the proper and complete storage of an auto accident.

- Database Object Information System: the set of specifications that control, sequence, and iterate the execution of various database object processes that cause

[20] A database object data structure has an identifier to isolate its instances from all others and fields which represent single values, multiple values (i.e., vectors), groups, repeating groups, and nested repeating groups. When a data structure only contains single valued fields it is termed a simple data structure database object. If the data structure contains multi-valued fields though nested repeating groups then it is termed a complex data structure database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

changes in database object states to achieve specific value-based states in conformance to the requirements of business policies. For example, the reception and database posting of data from business information system activities (screens, data edits, storage, interim reports, etc.) that accomplish entry of the auto accident information.

- Database Object State: The value states that represent the after-state of the successful accomplishment of one or more recognizable business events. Examples of business events are auto accident initiation, involved vehicle entry, involved person entry, and auto accident DUI involvement. Database object state changes are initiated through named business events that are referenced from within business functions. The business function, auto accident investigation includes the business event, auto-accident-incident initiation, which in turn causes the incident initiation database object information system to execute, which in turn causes several database object processes to cause the auto accident incident to be materialized in the database.

A database object may contain database objects. A database object is specified to the database object management system (DBMS) through a database object definition language (DDL). When database objects are stored, the result is called an database object database, or simply, a database. All four components of a database object operate within the "firewall" of the ANSI SQL DBMS. This ensures that database objects are protected from improper access or manipulation by 3GLs. An ANSI SQL:1992 DBMS that only defines, instantiates, and manipulates two dimensional data structures is merely a simplified functional subset of the ANSI SQL:1999 DBMS that defines, instantiates, and manipulates database objects.

4.1 Database Object Environment

Figure 4.1 depicts a high level diagram of the models that are involved in a database object environment. There are six distinct technology independent models. The technology independent models represent the specification of a database object environment.

The specification models in the database object environment include:

- Mission
- Database object
- Business information systems
- Business events
- Business function
- Business Organization

The mission and the database object model are presented in this chapter. The remaining database object environment models are presented in Chapter 8.

The mission model provides a description of the ultimate aims, goals, or objectives of the enterprise within which the database objects exist. The database objects are independent of the

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

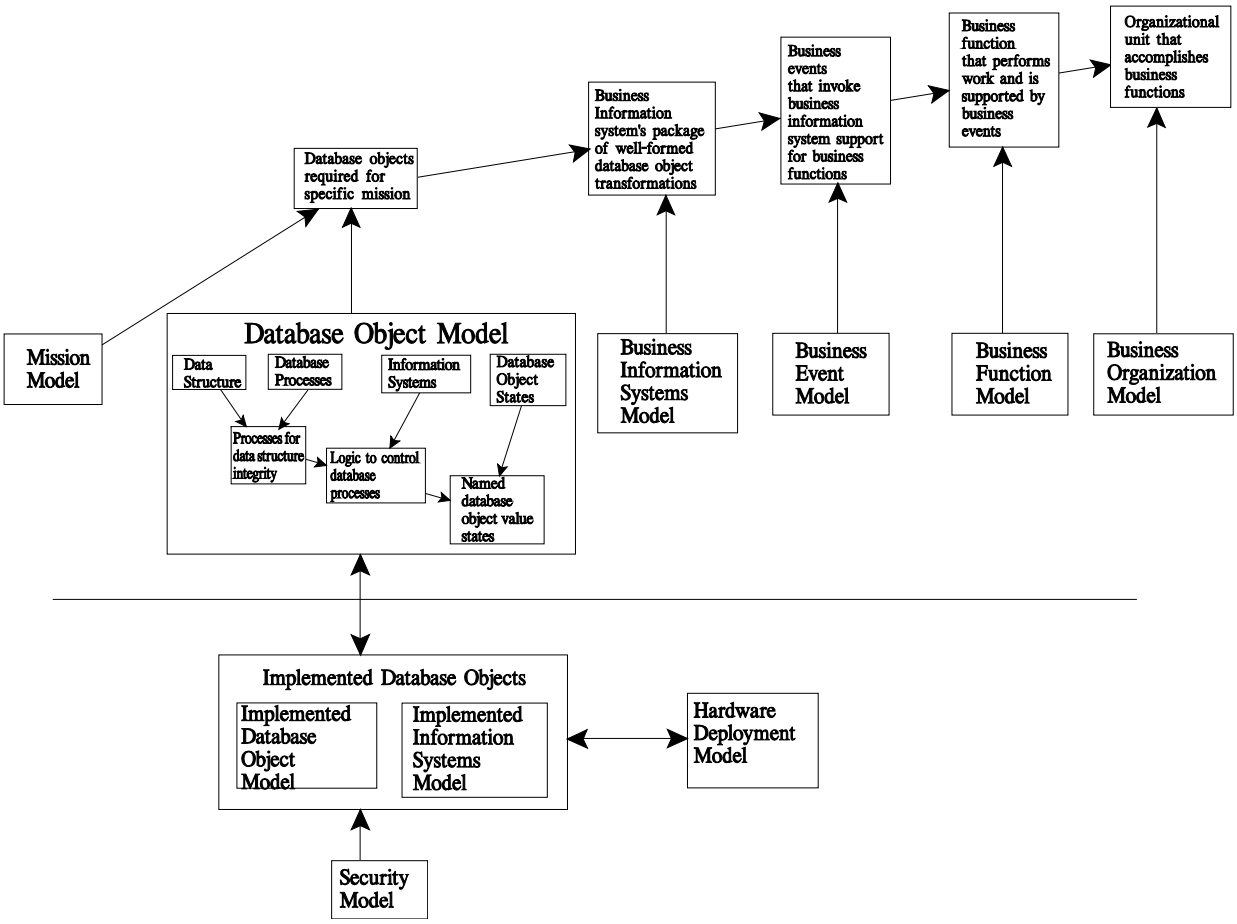


Figure 4.1. Architecture of Knowledge Worker framework models.

business information systems that employ them. The business function triggers the execution of business information systems via business events. Business functions are accomplished by business organizations. The intersections between all the models are many-to-many. That means for example, that one or more database objects serve the needs of one or more missions. Similarly, the intersection of database objects and missions, that is, mission based database objects are what is accomplished by one or more business information systems. Many to many relationships are employed to ensure that each model can be employed one or more times. This overall design characteristic flows naturally from the our minimally essential environment characteristics set out in the previous chapter.

Figure 4.2 depicts the database object composition paradigm. Depicted in this figure is the database object instance at a moment in time. This figure represents a trivial case of a database object. There is only one state, one database object information system, one set of database object processes that transform different database object database object tables.

Each white rectangle in Figure 4.2 represents a segment in the database object's data structure. Segments are either NULL or valued. In this figure, there is a root segment, two descendent segments, and for one segment, there are two additional descendent segments. When the database object structure (5 segments) is in the NULL state only the structure exists. A

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

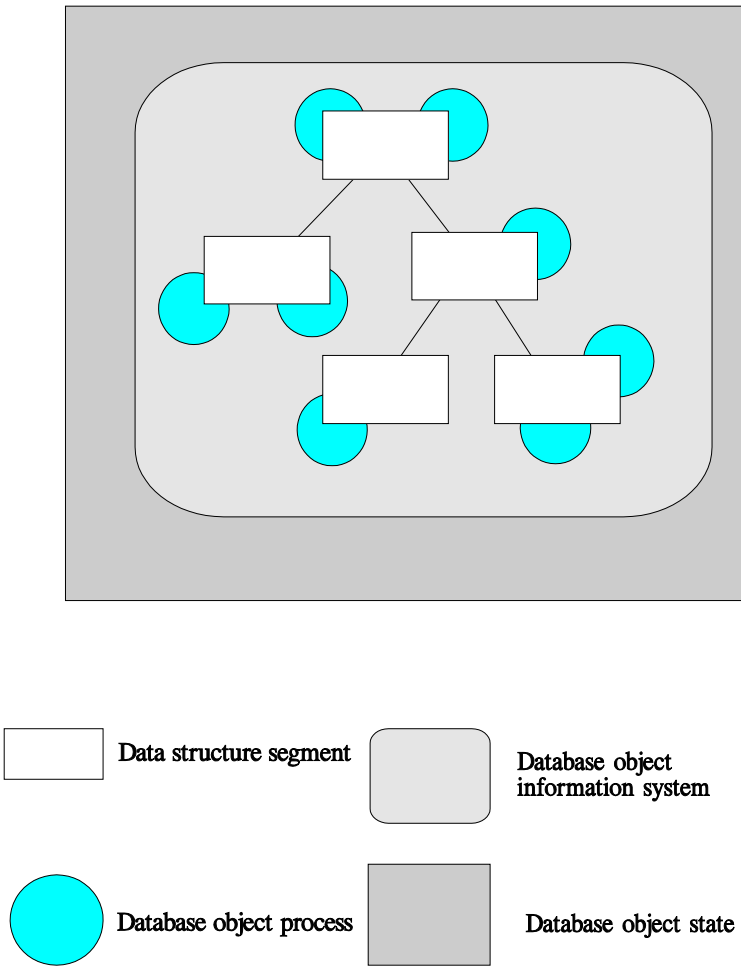


Figure 4.2. Database object composition paradigm.

database object in the null state is equivalent to a tree without leaves. As the database object progresses through states, different database object tables become valued. Some segments only contain one instance while others can contain multiple instances.

Database management systems, regardless of data model (network, hierarchical, independent logical file, and relational) exhibit the NULL and valued concept. After the data definition language is read and compiled by the DBMS, the database's state can be queried. In the case of a NULL database, the row counts of all tables is zero.

Each database object database object table is equivalent to an ANSI SQL:1999 table. The minimum essential field types are: single valued, multiple valued, group, repeating group and nested repeating group. In the EMPLOYEE example, a single value field is SOCIAL SECURITY NUMBER or BIRTH DATE. A multiple-value field is TELEPHONE NUMBERS.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

A group field is ADDRESS with its contained fields of STREET, CITY, STATE, and ZIP. A repeating group is a group field with multiple instances, for example, DEPENDENTS with the contained fields SSN, NAME, BIRTHDATE, and SEX. A nested repeating group is a repeating group that allows a contained repeating group. For example, DEPENDENTS containing HOBBIES.

Intersecting each database segment, represented in Figure 4.2 through partially hidden circles, is a series (from none to many) of database object processes. Each database object process executes when some action (e.g., add, delete, or modify) is taken. The actions can be the changing of a value (NULL to non-null or reverse) of a database object table field, the insertion of a database object data structure instance, or the modification of a database object data structure instance. The three types of database object processes are field, action, and references. A field level database object process are those that protect the allowed set of values for a database object. For example, there is probably a business rule that states that the value of SEX for an EMPLOYEE must be either MALE or FEMALE. This rule would therefore prevent either NULL (that is, unknown) or some other value such as YES.

The action type of database object process includes BEFORE, AFTER, and then verb types such as INSERT, DELETE, MODIFY. The action type and verb type enable six different actions that to occur for each database object table change. The process logic contained in each database object process can be to validate or compute. For example, the business might want to know the age of an EMPLOYEE candidate upon application. This could be computed when the EMPLOYEE BIOGRAPHIC database object table instance was installed through an AFTER INSERT action that would compute the age and then perform a database object table modification to install the age value.

The references action ensures that a particular database object table instance can neither be inserted, modified, or deleted without checking the existence of another database object table instance. In the example for EMPLOYEE, the third type of database object process can enforce the business rule that no EMPLOYEE BIOGRAPHIC database object table instance is allowed to be installed without there being one or more SKILL instances.

Surrounding the entire database object in Figure 4.2 is an outer database object information system. In the example of Figure 4.2 only one database object information system is shown. In reality, many database object information systems may cause database object transformations from one value state to another. For example, the database object information system that creates the employee requisition state would certainly be different from the database object information system that transforms an employee requisition to an employee candidate.

In the EMPLOYEE database object example, the null employee is transformed to the representation of an EMPLOYEE REQUISITION by valuing some one or more fields from one or more database object table instances. In this example, there may merely be a single database object table called EMPLOYEE REQUISITION with the fields: REQUISITION NUMBER, EMPLOYEE CLASS, EMPLOYEE SALARY RANGE, and then one or more instances of REQUIRED EMPLOYEE SKILLs that have been valued. Given that this information system's actions are accomplished, then the employee requisition exists. Upon query, 10 employee requisitions may be "open" and eligible to be "filled."

The next state for the database object EMPLOYEE might be EMPLOYEE CANDIDATE. That state has the EMPLOYEE REQUISITION state as a prerequisite. Another database object information system executes, finding a targeted database object that is in

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

EMPLOYEE REQUISITION stage, and adds to it a list of persons who represent EMPLOYEE candidates. In this situation, the database object table EMPLOYEE BIOGRAPHIC would be valued with an employee candidate's name, address, phone number, current employer, and available skills. When this state, EMPLOYEE CANDIDATE is achieved, it too can be queried so that employee candidate screening and interviewing can begin. During the execution of the EMPLOYEE CANDIDATE database object information system, certain database object processes such as VALIDATE TELEPHONE AREA CODE, VALIDATE ZIP CODE, and VALIDATE SKILLs would execute, thus enabling only valid data to be entered into the database. The remaining series of database object states are achieved through this process of proceeding from a prior state via a database object information system to the next state and at the same time protecting the database object's value instantiation by means of database object processes.

The outer most rectangle in Figure 4.2 represents the named state that in which the database object currently resides. For the database object EMPLOYEE, the value state might be EMPLOYEE REQUISITION. This represents some well defined state in the business' life cycle for an employee. The database object's state is achieved through the execution of one or more database object information systems. The state's achievement is binary. It is either achieved, or the effects of the database object information system that attempts to achieve it is rolled back. A database object's state is determined through one or more query statements. The easiest form of a state would be a column in the database object's root table that would contain the current value. That value would be set initially to null and then to alternate value states after one or more database object information system executions (that in turn cause database object process updates).

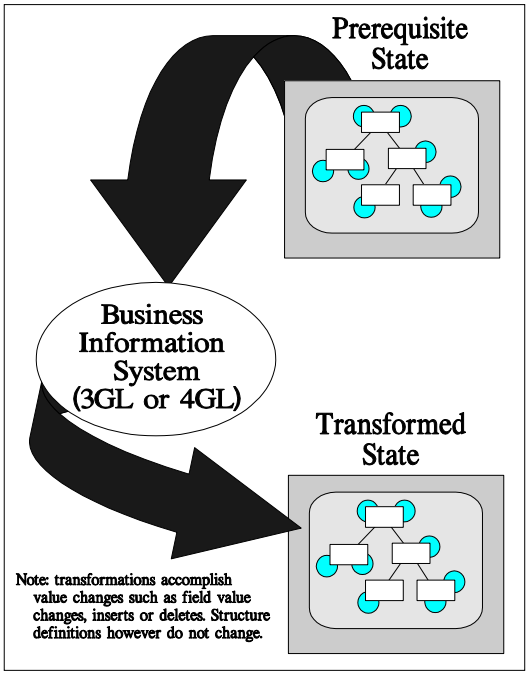
The states and the explicit conditions under which a database object's state is determined is established during analysis and design. More work? No. It's the same work that every quality database application designer and builder must perform. The only real difference is that here it is done once for the database, and thus, for all applications. Not only is it not more work, it is actually less work. Certainly anarchy takes less time than consensus. But the sum of application design and building anarchy and then subsequent conflict resolution and consensus building after all the database applications attempt to employ the same database will certainly take may times longer than had consensus been accomplished in the first place. The state of the database object can be queried. That is, the set of all database objects in the EMPLOYEE REQUISITION state can be found and then traversed.

4.2 Why ANSI SQL for Database Objects

The execution paradigm is shown in Figure 4.3 which shows the overall process by which a database object changes states. A database object must contain a prerequisite state so that it can be transformed to its next state. Database objects are selected for transformation in response both to the prerequisite state and because of some contained field's values. For example, the database object selection criteria might be to obtain all the EMPLOYEE REQUISITIONS where FULFILLMENT MONTH equals February 1996. Requested values from the database object are provided to the calling agent, that is, a 3GL or 4GL program. The program then performs its own logic and provides changed values back to the DBMS so that the database object's state can

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Execution Paradigm



Execution Sequence (effectively)

1. Data is obtained through 3GL and 4GL
2. Database object requested state is invoked through Business Information System. (Outside Square)
3. Data is passed to the database object information system (Rounded Edge Square)
4. Database object processes are passed data and they perform their actions (Circles to rear of white squares
5. Database object data structure is modified through inserts, changes, and deletes (connected collection of white squares)
6. If success, then commit, otherwise rollback.

Figure 4.3. Database object execution paradigm.

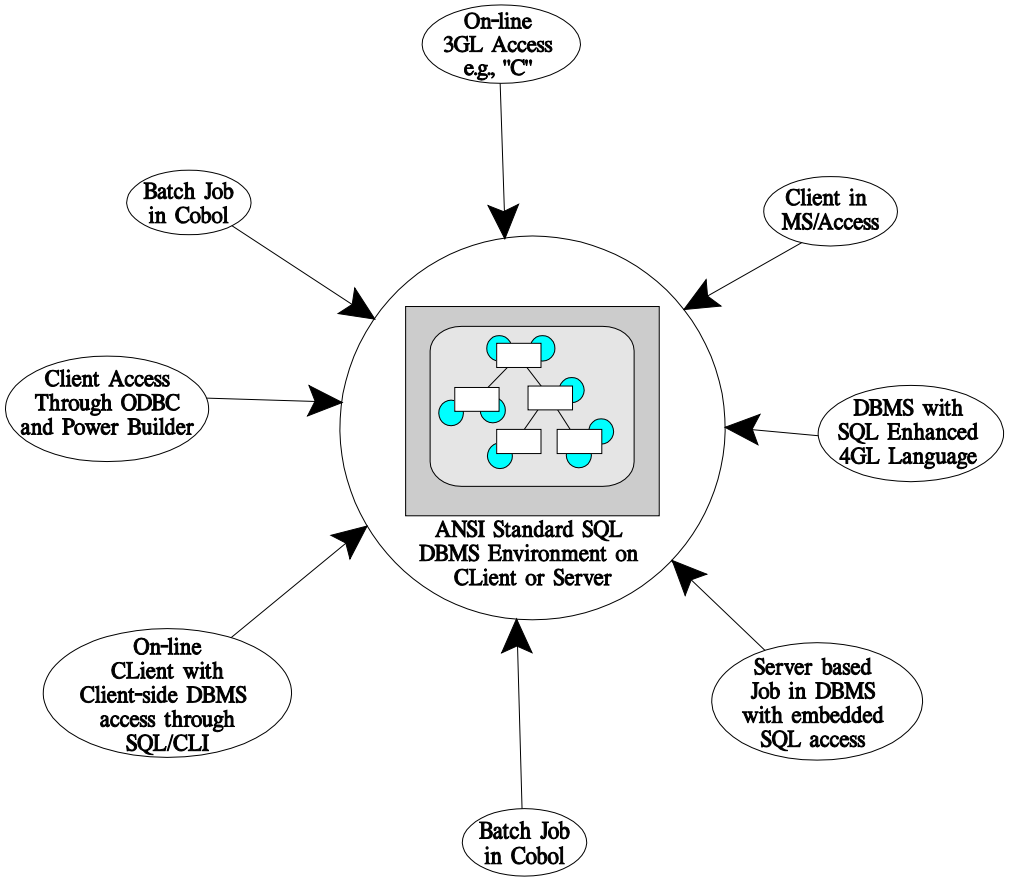
change. In this case, the EMPLOYEE REQUISITION's employee counselor might be assigned to a particular human resources staff person for action. Assuming that the change is accepted, then the database object's state might be called ASSIGNED EMPLOYEE REQUISITION. Other state changes might be to transform an EMPLOYEE REQUISITION to one that has EMPLOYEE candidates.

The states of the database object are those that are identified through database object analysis. The names of the states are predefined and to each state is assigned all the appropriate database object information systems. Assigned to the database object information systems are the database object processes, which in turn act on the database object tables.

While the proceeding example set appears to be hierarchical, a database object can be modified through many different database object information systems and their contained database object processes. The critical database object component that keeps the database object progressing through the correct value state sequence is the fact that a database object cannot be transformed from one state to another without first being in the prerequisite state. This prevents for example, a CANDIDATE EMPLOYEE being installed without there first being an EMPLOYEE REQUISITION.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The reason database objects are critical to client/server, heterogeneous, multiple-DBMS, world wide environments is depicted in Figure 4.4 which enumerates all the different types of change agents that surround the database objects. If the critical semantics of a business is stored both in its data and its computer programs, AND if there are many different types of language (C, COBOL, 4GL, MS/Access, Sybase's Power Builder, etc.), then the probability of having consistent semantics across this programming language environment is ZERO. Either a business



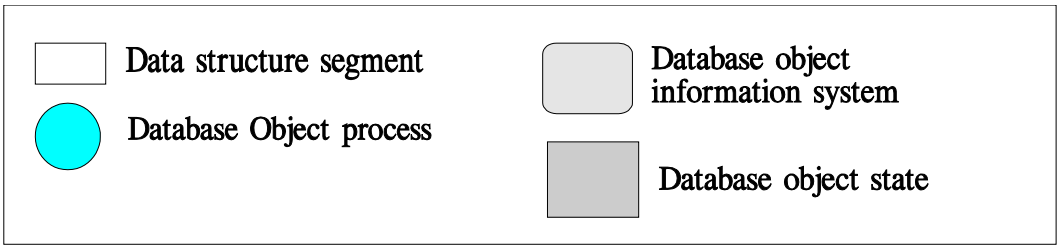


Figure 4.4. Database objects within a heterogeneous, multi-DBMS environment.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

must decide on one and only one DBMS, one and only one operating system, and one and only one 4GL programming language, or it must move the semantics of its essential business policies and procedures inside the "firewall" of the DBMS. But since it ranges from impractical to impossible to demand that a business operate only one brand of DBMS, these critical database object must be moved inside the ANSI SQL language. It is only through the ANSI SQL language that business can ensure that database object semantics are protected.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

5

TRANSPORTATION PUBLIC SAFETY DATABASE OBJECTS

The requirement to develop transportation public safety database objects resulted from a consulting engagement with a state government. The effort focused on an assessment of existing transportation public safety systems throughout the state such as the department of motor vehicles, department of transportation, state police, and various emergency medicine organizations. The first step in the process of developing an integrated database design consisted of developing the public safety mission description:

> The mission of transportation public safety is the development, implementation, monitoring, and maintaining transportation public safety traffic programs throughout the state so that citizens can transit within the state in a safe manner. Transportation public safety programs involve roads and all vehicle types there on, traffic accident investigations, vehicle stops of various types, on and off-road emergency incidents, and follow up investigations. Transportation public safety supports the creation of operational, management and long range analyses, reports, statistics, and predictions of out year requirements for enhancements to transportation public safety facilities and infrastructures.

In support of this mission, the following database object classes were needed:

- Vehicles
- Persons
- Highways
- Incidents
- Facilities and/or Organizations
- Property

An analysis of the database object classes produced the complete set of database objects listed in Figure 5.1. A diagram of the relationships among the database objects within the incident database object class is depicted in Figure 5.2. The incident database object contains seven subordinate database objects. Each of these database objects either contains database objects or only contains a database object structure.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Database Object Class	Database Object
Facility or organization	Facility
	Transport Organization
	Government Agency
Highway	Highway
	Traffic Device
Incident	Incident
	Vehicle Checkpoint Incident
	Vehicle Accident Incident
	Emergency Shock Trauma Facility Medical Event Incident
	Emergency Medical Response Incident
	Vehicle Stop Incident
	Vehicle Inspection Incident
	Judical Case Incident
Person	Government Person
	Emergency Shock Trauma Treatment
	Insurance Policy
	Person
	Judical case
	Violation
	Interview
	Hazardous Material or Object
	Property
	(drug or alcohol) Medical Test
Vehicle	Drivers License
	Vehicle

Figure 5.1. Database object classes and database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

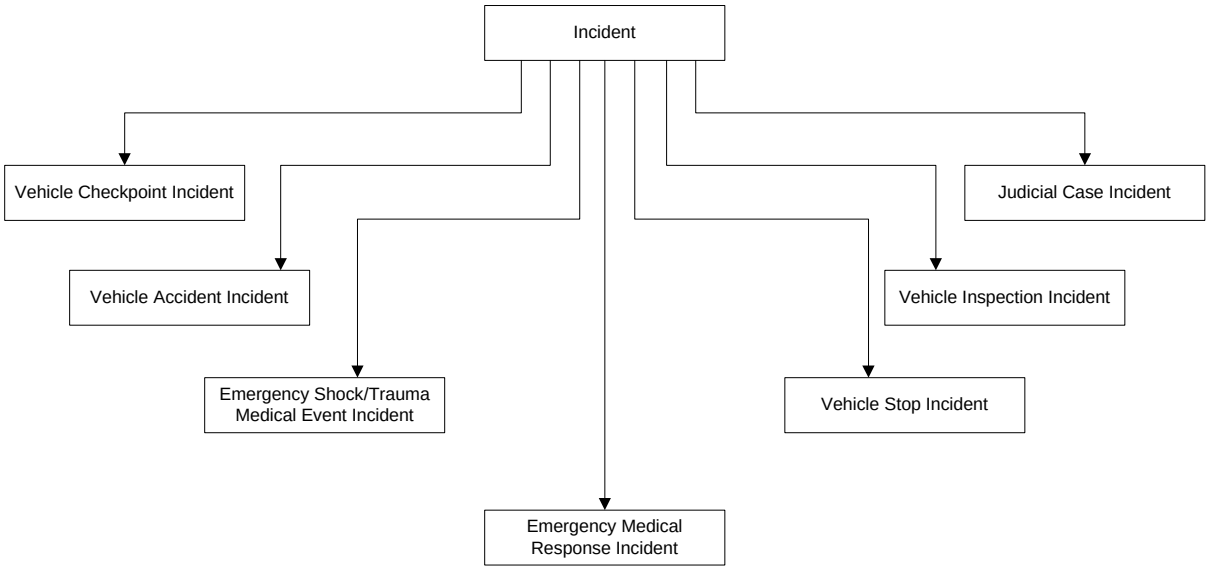


Figure 5.2. Emergency Medicine Incident.

5.1 Data Structure

Each database object class was analyzed to discover the hierarchy of contained data. For example, the incident database object class contains the following database objects:

- Incident
- Vehicle Checkpoint Incident
- Vehicle Accident Incident
- Emergency Shock Trauma Facility Medical Event Incident
- Emergency Medical Response Incident
- Vehicle Stop Incident
- Vehicle Inspection Incident
- Judicial Case Incident

5.1.l Incident

The incident database object contains only one simple database object data structure. It is presented in Figure 5.3. All the data structures within the Emergency Medical Response Incident database object database object are either simple or complex. Figure 5.4 enumerates its fields, each of which only represents a single value. In this database object example its data structure is equivalent to a traditional SQL two dimensional table.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Incident Identifier Incident Date Incident Time Incident Call-In Site Identifier Incident Initiator Identifier

Figure 5.3. Incident data structure.

5.1.2 Emergency Medicine Response Incident Database Object Tables

The Emergency Medicine Response Incident database object is depicted in Figure 5.4. It contain seven distinct database object tables, as follows:

- Emergency Medical Response Incident (see Figure 5.5)
- Emergency Medical Response (see Figure 5.6)
- Emergency Medical Response Harmful Event (see Figure 5.7)
- Emergency Medical Response Involved Person (see Figure 5.8)
- Emergency Medical Response Involved Person Injury (see Figure 5.9)
- Emergency Enroute Transport (see Figure 5.10)
- EMT/Paramedic/Ambulance Report (see Figure 5.11)

This figure also referentially contains data from three other database object data structures, that is,

- Emergency Shock Trauma Facility (see Figure 5.12)
- EMR Attendant (see Figure 5.13)
- Emergency Medical Response Organization (see Figure 5.14)

The database object tables are generally represented hierarchically except for Emergency Response Involved Person which has two owners, Emergency Medical Response and Emergency Response Harmful Event. Note also that Emergency Response Harmful Event is recursive, that is, there may be any number of levels of contained Emergency Response Harmful Events.

In Figure 5.4, the data structure Emergency Medical Response Involved Person is co-owned by the Emergency Medical Response database object table and a database object table within the Person database object (not shown). All the data structures contained in the Emergency Medical Response database object contain a multiple part identifier including the Emergency Medical Response Incident identifier and the incident identifier.

Figure 5.5, Emergency Medical Response Incident, contains a single database object table that contains the critical incident identifying information. Included in this is the various time, date, call-in site, and person identifiers, the geographic coordinates of the emergency medical response incident, and information regarding the assigned responding unit such as its identifier, type of equipment/unit, and the responding location.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

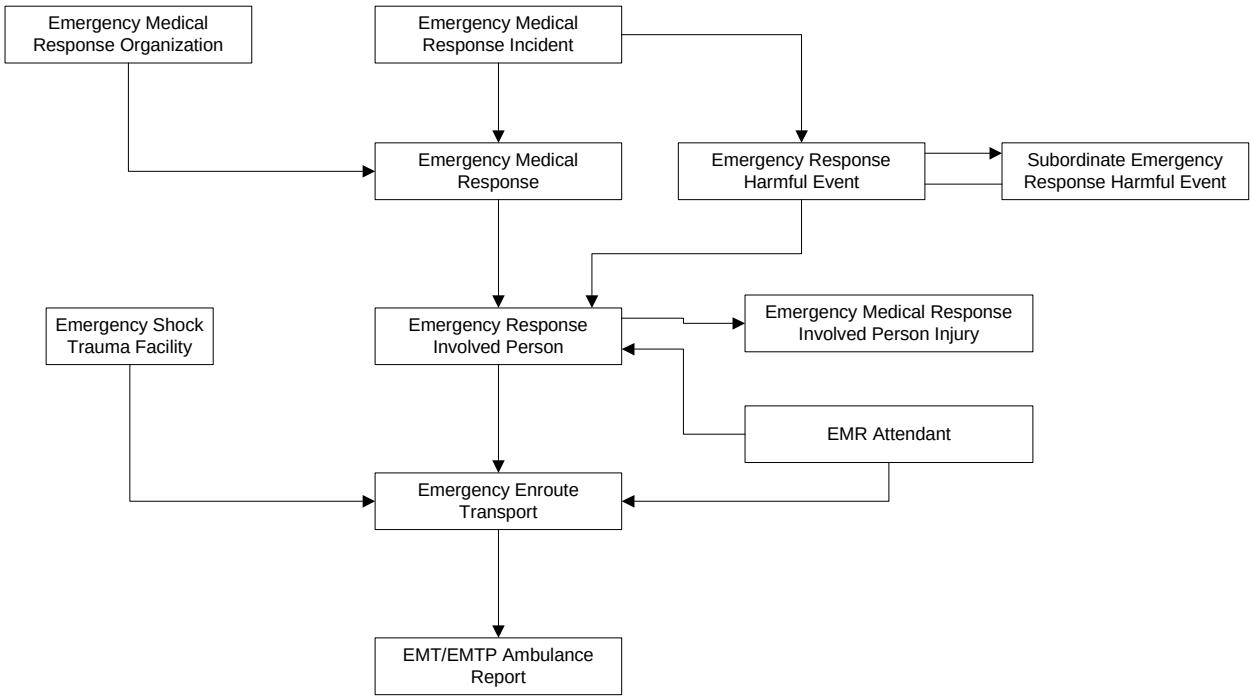


Figure 5.4. Emergency medicine response incident database object.

Incident Identifier
Emergency Medical Response Incident Identifier
Emergency Medical Response Incident Date
Emergency Medical Response Incident Time
Emergency Medical Response Incident Call-in Site Identifier
Emergency Medical Response Incident 911 Operator Identifier
Emergency Medical Response Incident Location Name
Emergency Medical Response Incident GIS Location Coordinates
Emergency Medical Response Incident Dispatch Unit Identifier
Emergency Medical Response Incident Dispatch Unit Type
Emergency Medical Response Incident Dispatch Unit Organization

Figure 5.5. Data Structure: Emergency Medical Response Incident

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Incident Identifier Emergency Medical Response Incident Identifier Emergency Attendants Untrained Count Emergency Attendants Cardiopulmonary Count Emergency Attendants Extrication Count Emergency Attendants Military Medical Count Emergency Date and Time Emergency Response Arrival Time Emergency Response Unit Identifier Emergency Response Elapsed Time

Figure 5.6. Emergency Medical Response

Emergency Medical Response Incident Identifier Harmful Event Sequency Number Harmful Event Parent Identifier Harmful Event Description Harmful Event Date Harmful Event Time Harmful Event Recorded by Identifier Geographic Coordinates of Harmful Event Geographic Coordinates of Harmful Event Hazardous Object Harmful Event Relative Rank

Figure 5.7. Harmful Event

Incident Identifier Emergency Medical Response Incident Identifier Emergency Medical Response Involved Person Identifier Emergency Medical Response Involved Person Emr Attendant Identifier Emergency Medical Response Involved Person Harmful Event Identifier Emergency Medical Response Involved Person Name Emergency Medical Response Involved Person Age Emergency Medical Response Involved Person Date of Birth Emergency Medical Response Involved Person Insurance Description Emergency Medical Response Involved Person Social Security Number Emergency Medical Response Involved Person Sex Emergency Medical Response Involved Person Home Address

Figure 5.8. Emergency Medical Response Involved Person

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Incident Identifier
Emergency Medical Response Incident Identifier
Emergency Medical Response Involved Person Identifier
Chief Complaint Description
History of Present Illness Description

Past Medical History
- Code

Current Medications
- Name
- Dosage

Allergies
- Name
- Description

Vital Signs Assessment
- Time
- Pulse
- Respiration
- Blood Pressure
- EKG

Oxygen Applied
- at LPM Quantity
- via Code

Survey
- Primary
  - Airway & Cervical Spine
    - Assessment Codes
    - Initial Intervention
    - Outcome Code
  - Breathing
    - Assessment Codes
    - Initial Intervention Codes
    - Outcome Codes
  - Circulation
    - Assessment Codes
    - Initial Intervention Codes
    - Outcome Codes
  - Exposure
    - Eye Opening Assessment Codes
    - Best Verbal Response Codes
    - Best Motor Response Codes

Figure 5.9. Emergency Medical Response Involved Person Injury

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Secondary
  - Skull & Face Findings
    - Code
    - Description
  - Pupils
    - Right Size in Mm
    - Right Reactive Indicator
    - Left Size in Mm
    - Left Reactive Indicator
  - Jugular Veins Code
  - Neck and C Spine
    - Code
    - Findings
  - Lung
    - Left Code
    - Right Code
  - Abdomen & Pelvis
    - Code
    - Findings
  - Extremities Back
    - Code
    - Findings
  - Neurological
    - Code
    - Findings
- Intubation
  - Medic Number
  - Quantity of Attempts
  - Success Code
  - ETT Oral or Nasal Code
  - Tube Size in Mm
  - EOA Code
  - Time
  - Sounds Code
  - Gastric Counts Code
  - Complications Description
- Iv Strategy Attempt
  - Nsss Quantity Code
  - Medic Identifier
  - Site
  - Time
  - Quantity of Attempts
  - Size

Figure 5.9 Emergency Medical Response Involved Person Injury (continued)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Other Management
  - Special Immobilization Code
  - Extrication
    - Code
    - Time
    - Aerosol Treatment
      - Measurements
        - Prepeak Flow
        - Prelung Sound
        - Postpeak Flo
        - Postlung Sound
      - Quality Indication Code
    - Fractures Splinting Code
    - Other Description
- Medications
  - Drug Name
  - Drug Time
  - Drug Amount
  - Drug Route
  - Standing Orders Followed Description
- Verbal Orders
  - Time
  - Description
- Medical Control Contacted Code
- Transported to Name
- BLS Unit Identifier
- Verbal Report Given to Name
- Supplemental Information Narrative
- Signature Person
  - Name
  - Identifier
- Weather Conditions Code
- Involved Medic Personnel
  - Name
  - Identifier
- Other Personnel
  - Name
  - Identifier

Figure 5.9 Emergency Medical Response Involved Person Injury (continued)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Incident Identifier
Emergency Medical Response Incident Identifier
Emergency Ems Unit Identifier
From Location
    Name
    Geographic Coordinates
    Description
    Start Date
    Start Mileage
    Start Time

To Location
    Emergency/shock Trauma Facility Identifier
    Name
    Geographic Coordinates
    Description
    Arrival Date
    Arrival Time
    End Mileage

Emergency Medical Response Involved Person
    Identifier

EMR Attendant
    Identifiers

Figure 5.10 Emergency Enroute Transport

Incident Identifier
Emergency Medical Response Incident Identifier
Emergency Ems Unit Identifier
EMT/Paramedic Ambulance Report Date
EMT/Paramedic Ambulance Report Alarm Time
EMT/Paramedic Ambulance Report 10 8 Time
EMT/Paramedic Ambulance Report 10 2 Scene Time
EMT/Paramedic Ambulance Report Arrival Scene Time
EMT/Paramedic Ambulance Report Clear Time
EMT/Paramedic Ambulance Report Call Categorization Codes
EMT/Paramedic Ambulance Report Transportation Initiated Time
EMT/Paramedic Ambulance Report Arrival Hospital Time
EMT/Paramedic Ambulance Report 10 7 Time
EMT/Paramedic Ambulance Report Received by Identifier
EMT/Paramedic Ambulance Report Driver Identifer
EMT/Paramedic Ambulance Report Driver Name
EMT/Paramedic Ambulance Report Attendant Name
EMT/Paramedic Ambulance Report Attendant Identifier

Figure 5.11. EMT/Paramedic Ambulance Report

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Emergency Facility Identifier Emergency Facility Name Emergency Facility Physicians Availability Identifier Emergency Facility Capability Emergency Facility Type Emergency Facility Radio Call Number Emergency Facility Address Emergency Facility Telephone Number Emergency Hours Trauma Center Designator

Figure 5.12. Emergency Shock Trauma Facility

Emergency Medical Response Attendant Emergency Medical Response First Name Emergency Medical Response Middle Initial Emergency Medical Response Last Name Emergency Medical Response Organization Identifier Emergency Medical Response Highest Certification Type Emergency Medical Response Highest Certification Identifier Emergency Medical Response Highest Certification Date

Figure 5.13. EMR Attendant

Emergency Organization Identifier Emergency Organization Primary Function Emergency Organization Address Emergency Organization Name Emergency Organization Capability Emergency Organization Radio Call Number Emergency Organization Telephone Number

Figure 5.14 Emergency Medical Response Organization

5.1.3 Emergency Medical Response

Figure 5.6 contains summary information that is both initially valued and then updated in at least two cycles. During the initial cycle, the segment instance is created and valued with the incident identifier, the date and time of the response. The second cycle is a update cycle in which the segment instance has its response arrival time and response elapsed time (duration from incident call-in time to arrival time) stored, as well as any update to the unit's identification in case there was an emergency unit redeployment. The third cycle includes an update of the count of the various emergency attendants (untrained, cardiopulmonary, extrication, and military) who were involved with the call. This information may be initially valued and subsequently updated as new counts are known.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

5.1.4 Emergency Medical Response Harmful Event

Figure 5.7 presents the data structure for the Harmful Event. This information is incrementally. When an emergency unit first arrives on a scene a very preliminary assessment of the incident becomes available. This information is fed into the top level segment instance of the Harmful Event hierarchy. As the harmful event unfolds, either the initial information is changed, and/or additional levels of Harmful Events unfold. At each unfolding, the information is recorded so that both a complete and accurate assessment can eventually be made.

Given the technology of the mid 1990s, it is unlikely that the Harmful Event information can be accurately decoded from voice to ASCII. Consequently, the initially recording mechanism may be notes on paper, or voice recording into either a hand held recorder or a remote recorder. Once the overall Harmful Event has concluded, information other than the basic structural information may be recorded. The recording may therefore be in two phases: one for the basic structural information and the second for the appropriate Harmful Event description.

5.1.5 Emergency Medical Response Involved Person

Figure 5.8 contains the data structure, Emergency Medical Response Involved Person. It has a two part identifier, one part identifying its Emergency Medical Response Involved Person Incident "owner" and the other part is the identifier of the person who is really the rightful owner of the Emergency Medical Response Involved Person. In the same way that person is a co-owner of a contained data structure within the Emergency Medical Response Involved Person segment of the Emergency Medical Response database object, the other database object co-owners of Emergency Medical Response Involved Person are:

- Person (figure not provided)
- Emergency Response Harmful Event (see Figure 5.7)
- EMR Attendant (see Figure 5.13)

The relationships between these database object tables and their co-owners is represented through traditional value based relationship mechanisms. This relationship mechanism is used for illustrative purposes only as the implementing DBMS may provide other non-value-based and invisible to the user relationship mechanisms.

In the case of transportation public safety, these database object table co-owners only contain a simple data structure that serves as the relationship mechanism between the two database objects. If the database were primarily concerned with shock trauma facilities then, for example, the Emergency/Shock Trauma Facility data structure would be a co-owned data structure with the Shock/Trauma Facility database object. Similarly, the EMR/Attendant data structure is a co-owned data structure with the EMT/Paramedic Human Relations database.

The database object data structure, Emergency Facility Medical Event Involved Person, is co-owned by the Emergency Facility Medical Event Incident database object and the Person database object. If the database were the hospital's database, then the contained data Structure, Emergency Facility Medical Event Involved Person Emergency Doctors would be co-owned by

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

the Emergency Incident database object and the hospital's medical records business database object for the emergency doctor.

5.1.6 Emergency Medical Response Involved Person Injury

Figure 5.9 contains a very complex database object, Emergency Medical Response Involved Person Injury. This figure is shown with indentations to illustrate the nested structures of data fields. A few of the fields, like EMERGENCY MEDICAL RESPONSE INCIDENT IDENTIFIER and CHIEF COMPLAINT DESCRIPTION only represent one value. Fields like PAST MEDICAL HISTORY contains a list of codes.

The CURRENT MEDICATIONS field contains two fields, NAME and DOSAGE and represents sets of values. The group fields ALLERGIES, VITAL SIGNS ASSESSMENT, OXYGEN APPLIED are similarly constructed. The VITAL SIGNS ASSESSMENT field, for example, contains time, pulse, respiration, blood pressure, and EKG. Each different time this information is captured it is as a new instance within VITAL SIGNS ASSESSMENT.

The field SURVEY contains two main substructures, PRIMARY and SECONDARY. The PRIMARY substructure contains four different substructures: AIRWAY & CERVICAL SPINE, BREATHING, CIRCULATION, and EXPOSURE. Each of these contained substructures represent multiple sets of values. The SECONDARY substructure contains eight different contained substructures, SKULL & FACE FINDINGS, PUPILS, JUGULAR VEINS CODE, NECK AND CERVICAL SPINE, LUNG, ABDOMEN & PELVIS, EXTREMITIES BACK, and NEUROLOGICAL. Each of these contained substructures contains one or more sets of information related to the EMERGENCY MEDICAL RESPONSE INVOLVED PERSON INJURY. The final sets of substructures, INTUBATION through VERBAL ORDERS, five single valued fields (MEDIAL CONTROL CONTACTED CODE, ..., WEATHER CONDITIONS CODE), and two final multi-field groups, INVOLVED MEDIC PERSONNEL, and OTHER PERSONNEL.

5.1.7 Emergency Enroute Transport

Figure 5.10 provides information about an Emergency Enroute Transport. The reason there can be multiple occurrences of this information is that the transport may have been by ground ambulance and then by air ambulance and finally by ground ambulance. For each leg of the journey information about the EMS unit, from location, to location, and involved persons is recorded. If during a transport additional Emergency Medical Response Involved Person Injury information (see Figure 5.9) is recorded, that is, assessments conducted and treatments are performed then that information is recorded as additional Emergency Medical Response Involved Person Injury information. The final information stored in this database object table is the identifier of the Emergency Shock Treatment Facility.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

5.1.8 EMT/Paramedic/Ambulance Report

Figure 5.11 provides the information for an EMT/Paramedic Ambulance Report. This information consists mainly of the dates, various types of times (alarm, arrival, transport initiation, hospital arrival, etc.) and the appropriate names and identifiers of the various drivers and attendants.

5.1.9 Emergency Shock Trauma Facility

Figure 5.12 contains the information that is automatically available whenever the Emergency Shock Trauma Facility identifier is properly used when the Emergency Enroute Transport database object table is valued. Similarly, Figure 5.13 contains the information for the EMR Attendant when the proper identifier is employed within either the Emergency Response Involved Person database object table or the Emergency Enroute Transport database object table. Finally, Figure 5.14 contains the information that fully identifies the Emergency Medical Response Organization when its identifier is used within Emergency Medical Response database object table instances.

An examination of all the other public safety database objects show that the contained database object data structures are of the following types:

- Simple that contain only single valued fields
- Complex that contain single valued, multiply valued, groups, repeating groups and nested repeating groups
- Co-owned data structures that are shared with other database objects
- Data structure instance add, delete and modify
- Field and data structure constraints
- Inter database object reference integrity clauses and their automatic actions

The key difference between database objects and traditional SQL data structures is that with SQL tables 3GL and 4GL procedural languages accomplish, control, and undo any database object process that is determined to be in violation. In contrast, the database object data structure can only be added, deleted, or modified through the database object process which consists of the actual database object process actions and also the surrounding programming logic, control, and error processing. Neither 3GL nor 4GLs can directly affect the value state of database objects.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

5.2 Database Object Processes

Each database object table process is restricted to just one database object's data structure or a segment within the database object. In practice, a database object process' action should address only one database object database object table so that database objects can be flexibility built.

The overall logic for entering an entire Emergency Medicine Response Incident database object, that is, database object information system, is provided in Figure 5.15. This logic contains a nested set of database object table processes. It starts with the insertion of the Emergency Medical Response Incident. Given success, the Emergency Medical Response Harmful event and its available subordinate hierarchies are stored. Stored then are the Emergency Medical Response itself. Then stored is the first Emergency Medical Response Involved Person. After that is stored, the Emergency Medical Response Involved Person Injury and Emergency Medical Response Enroute Transport is stored. Finally, the Emergency Medical Response EMT/Paramedic Ambulance Report is stored.

Several of these insertions require database object referential integrity checks require that already stored database object be present so that dependent database objects can be linked to them. Figure 5.16 identifies the database objects that must be present prior to the insert of the Emergency Medical Response Incident.

Because the database object data structure, Emergency Response Harmful Event, can contain subordinate Harmful Events it is likely that its inclusion database object process logic is likely to be removed from within the database object process that stores the Emergency Medical Response Incident database object. The Harmful Event database object is best installed into the database through its own database procedure. Figure 5.17 presents the type of database object process logic required to locate, position, and then store the Emergency Response Harmful Event database object.

If the Emergency Medical Response Incident is stored without the Emergency Response Harmful Event, then when the Emergency Response Involved Person is stored its foreign key reference to the Emergency Response Harmful Event would remain NULL. Later, there would have to be a database object modify routine that would contain the logic contained in Figure 5.18

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Insert Emergency Medical Response Incident

Insert Emergency Medical Response Harmful Event, else rollback

Insert Emergency Medical Response (validate Emergency Medical Response Organization existence)

Insert Emergency Response Involved Person (Validate Emergency Response Harmful Event if valued)

Insert Emegency Response Involved Person Injury

Insert Emergency Enroute Transport (validate Emergency/shock Trauma Facility existence, validate EMR Attendant existence)

Insert Ambulance Report, else rollback
Insert EMT/Paramedic Report, else rollback

Else rollback

Else rollback

Else rollback

Else rollback

Figure 5.15. Emergency Medical Response Insert process.

Emergency Medical Response Organization
Emgency/shock Trauma Facility
EMT/Paramedic attendant

Figure 5.16. Required Database Objects Before Insert of Emergency Medical Response Incident

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

```
Locate emergency medical response incident

Insert emergency response harmful event

    Loop
        Insert subevent until end
    EndLoop
```

Figure 5.17 Database object process logic for the Emergency Response Harmful Event database object.

```
Locate Emergency Response Incident

    Locate Emergency Response Harmful Event

        Locate Emergency Medical Response

            Locate Emergency Response Involved Person

            Modify Emergency Response Involved Person foreign key to Emergency Response Harmful
            Event primary key value
```

Figure 5.18 Database object modify routine.

5.3 Database Object Information Systems

As stated above, each database object information system consists of a well-ordered collection of database object table processes. After each database object table process there would be result success or failure alternative actions. The failure actions are implicit in the pseudo code contained in Figure 5.15. There could also be other collections of code such as computations that then cause updates, obtaining of other data from related tables, and the like. Figure 5.19 enumerates a set of database object information systems for this transportation example.

In this example there are eight distinct database object information systems. In the Enter Emergency Medical Response Involved Person database object information system, there are three database process. Each process then operates on one or more database object table processes. The actual process code involved in the database object table process would address the correct insertion, deletion or change. Each such change would involve selection, validation, computation, and the like, including the validation of any foreign key look ups.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Database Object: Emergency Medical Response Incident		
Database Object Information Systems	**Database Object Table Processes**	**Database Object Table**
Enter Emergency Medical Response Incident	Insert Emergency Medical Response Incident	Emergency Medical Response Incident
Enter Emergency Medical Response	Locate Emergency Medical Response Incident Insert Emergency Medical Response	Emergency Medical Response Incident Emergency Medical Response
Enter Emergency Response Harmful Event	Locate Emergency Medical Response Incident Insert Emergency Response Harmful Event	Emergency Medical Response Incident Emergency Response Harmful Event
Enter Emergency Response Involved Person	Locate Emergency Medical Response Incident Locate Emergency Medical Response Insert Emergency Response Involved Person	Emergency Medical Response Incident Emergency Medical Response Emergency Response Involved Person
Enter Emergency Medical Response Involved Person Injury	Locate Emergency Medical Response Incident Locate Emergency Medical Response Locate Emergency Response Involved Person Insert Emergency Medical Response Involved Person Injury	Emergency Medical Response Incident Emergency Medical Response Emergency Response Involved Person Emergency Medical Response Involved Person
Enter Emergency Enroute Transport	Locate Emergency Medical Response Incident Locate Emergency Medical Response Locate Emergency Response Involved Person Insert Emergency Enroute Transport	Emergency Medical Response Incident Emergency Medical Response Emergency Response Involved Person Emergency Enroute Transport
Enter Ambulance Report	Locate Emergency Medical Response Incident Locate Emergency Medical Response Locate Emergency Response Involved Person Locate Emergency Enroute Transport Insert Ambulance Report	Emergency Medical Response Incident Emergency Medical Response Emergency Response Involved Person Emergency Enroute Transport Ambulance report
Enter EMT/Paramedic Report	Locate emergency medical response incident Locate emergency medical response Locate emergency response involved person Locate emergency enroute transport Insert EMT/Paramedic Report	Emergency Medical Response Incident Emergency Medical Response Emergency Response Involved Person Emergency Enroute Transport Ambulance report EMT/Paramedic Report

Figure 5.19. Database Object Information Systems.

5.4 Database Object States

Database objects proceed through states. Each state is accomplished by one or more database object information systems. If the state is achieved there is success. Otherwise there is a rollback to the previous state. Figure 5.20 enumerates the states for the database object, Emergency Medical Response. There are five states. Each state triggers one or more database object information systems. The state, Involved Person Identification has three database object information systems. The first, Record Involved Person involves three database object processes, and each of those causes an insert and/or update to a database object table.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Database Object States and Transformations				
Database Object Class	**Resource State**	**Database Object Information System**	**Database Object Table Process**	**Database Object Table**
Emergency Medical Response	EMR incident identification	Emergency medical response incident identification	Insert Emergency Medical Response Incident	Emergency Medical Response Incident
	Incident response	Incident Medical Response	Locate Emergency Medical Reponse Response Incident	Emergency Medical Response Incident
			Validate Emergency Medical Response Organization	Emergency Medical Response Organization
			Insert Emergency Medical Response	Emergency Medical Response
	Harmful event recording	Record Harmful Event	Locate Emergency Medical Response Incident	Emergency Medical Response Incident
			Insert Emergency Medical Response Harmful Event	Emergency Medical Response Harmful event
	Involved Person Identification	Record Involved Person	Locate Emergency Medical Response Incident	Emergency Medical Response Incident
			Locate Emergency Medical Response	Emergency Medical Response
			Insert Emergency Medical Reponse Involved Person	Emergency Medical Response Involved Person
		Connect Involved Person to Harmful Event	Locate Emergency Medical Response Harmful Event	Emergency Medical Response Harmful event
			Modify Emergency Response Involved Person	Emergency Medical Response Involved Person
		Record Involved Person Injury	Insert Emergency Medical Response Involved Person Injury	Emergency Medical Response Involved Person Injury

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Database Object States and Transformations				
Database Object Class	**Resource State**	**Database Object Information System**	**Database Object Table Process**	**Database Object Table**
	Emergency Enroute Transport	Record Emergency Enroute Report	Locate Emergency Medical Response Incident	Emergency Medical Response Incident
			Locate Emergency Medical Response	Emergency Medical Response
			Locate Emergency Medical Response Involved Person	Emergency Medical Response Involved Person
			Validate Emergency Shock Trauma Facility	Emergency Shock Trauma Facility
			Validate EMR Attendant	EMR Attendant
			Insert Emergency Enroute Transport	Emergency Enroute Transport

Figure 5.20. Database Object States and Transformations

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

6

COURTS DATABASE OBJECTS

The essential mission of the courts is to deliver justice that is:

- Easily accessed
- Fair, impartial justice with integrity
- Efficient, effective court operations.

In greater detail, the courts establish policies and procedures that deal with court calendars, cases, courts, court transactions, issues of law, participants, and workload management. Unlike the transportation public safety set of database objects, the set of court objects were simpler. That is, there weren't multiple classes of objects within which were multiple objects. Excluded from the analysis were database objects related to court finance, pre-court activities such as police, apprehension, and pretrial detainment. Also excluded from the analysis were the set of database objects dealing with post court detention, probation, and fine payment management. If the scope had therefor been broader then there would have been multiple object classes. Analysis efforts in these related but out-of-scope areas would cause changes to the "edges" of the database objects so their semantics and instances can be exchanged.

6.1 Data Structure

Derived directly from the mission description, the primary court database objects are:

- Court calendars
- Cases
- Courts
- Court transactions
- Issues
- Participants, and
- Workload management.

Figure 6.1 depicts the overall interrelationship among all the court database objects. There are secondary database objects that are required by the overall court environment and are reference by these primary court objects. Collectively these court database objects embrace all the types and kinds of data that must be defined, structured, and controlled to support the court's effective management and to provide for a historical record of all court activities.

6.1.1 Calendar Management

Court calendars indicate when the courts are in session. Affecting court calendars are local and state-wide holidays, appointments, the schedule of cases, and prescribed schedules of events

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

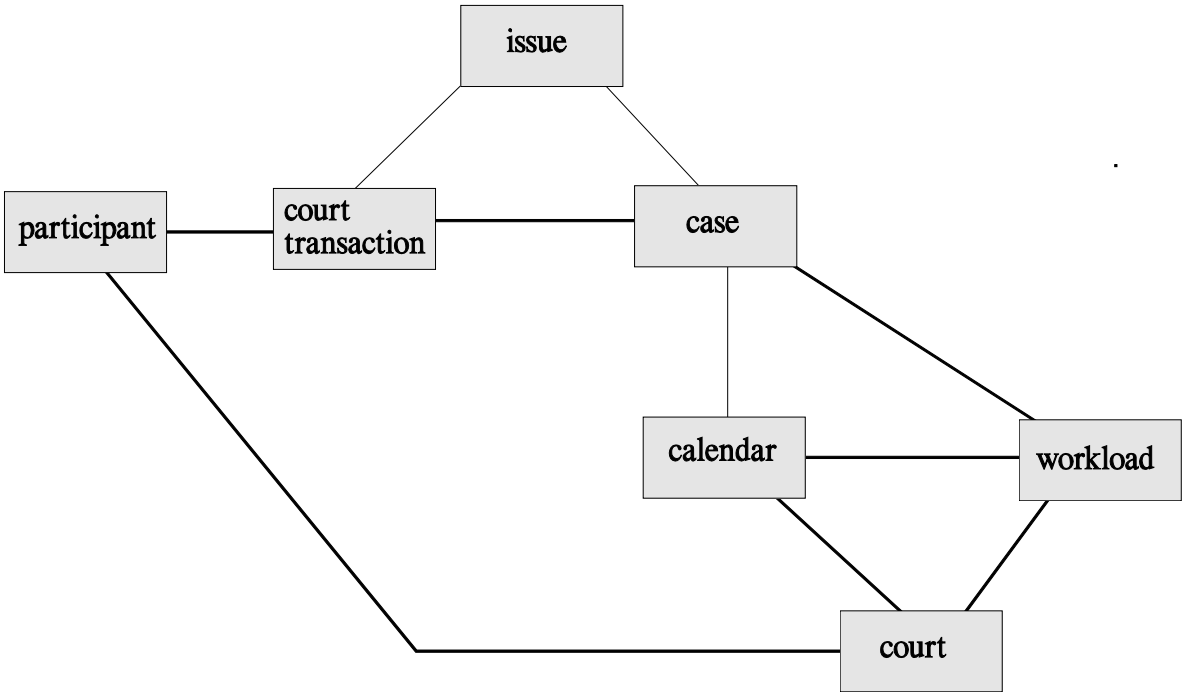


Figure 6.1. Fundamental database objects involved in Courts.

within cases of certain types. Figure 6.2 identifies the set of tables involved in calendar management. The court calendar database object consists of two tables: calendar day of year and day's event.

Referenced tables that fully identify the court calendar day's event are:

- State-wide court event,
- Local court event,
- Specific assignment (j/r (judge or referee) assigned county court type, division and room)
- Case event

These references are to database objects of their own, that is, state wide court events, local court events, specific assignments, and case events. The calendar (that is, day's event) is not referenced within any other database object.

6.1.2 Case Tracking Management

Cases are concerned with, for example, predetermined schedules of activities, appearances, the admissibility of evidence, evidence, motions, orders, filings, trials, juries, application of specific issues within the law, case consolidations and separations. The primary database object for case

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

tracking is the case. Figure 6.3 identifies the tables associated with the case database object. It consists of two main tables, case and case event.

Referenced tables that provide complete identification to the case and its event include:

- Case event type (that in turn references case type and case event type)
- Incident (that is the precipitating action that causes the case)
- Consolidation and consolidation reason and duration

The case event is referenced within the following database objects:

- Court calendar (day's event)
- Trial or hearing
- Consolidation reason and duration
- Court transaction considered within the case event

Associated with the case is the trial database object. A special form of a trial is a hearing. Both are scheduled, heard, reported, and for the case events that are either trials or hearings, documents are considered, motions and orders dealt with, and decisions made. At some trials the judge determines the verdict/decision and at other types of trials juries render the verdict. For the jury trials, there is a jury panel. The jury panel, a database object in its own right, consists of jurors, who are a type of participant.

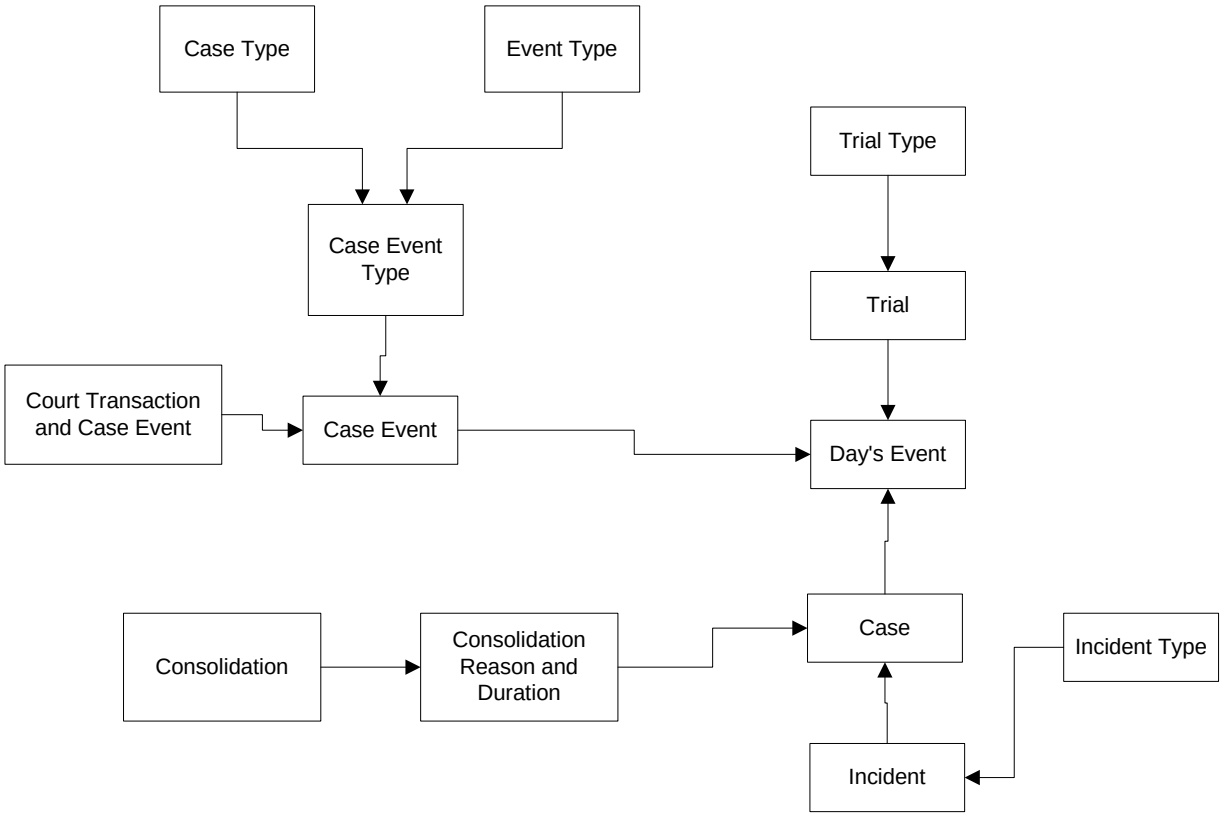


Figure 6.2. Case Tracking Management

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Cases can be consolidated into other cases for a period of time. This commonly occurs during class action suits. The consolidation is a database object which contains a referencing subtable for the individual cases that comprise it.

Finally, the incident that is the cause of the case is a database object that originates externally to the courts database. The set of tables that comprise the incident are from other database classes. Examples of incidents are contained in the prior chapter, Chapter 5.

6.1.3 Court Transaction Management

Court transactions catalog, store, and interrelate all the transactions that deal with participants, issues in the law, case evidence, filings, documents, motions, orders, findings, bonds, bail, warrants, charges, complaints, traffic tickets, notice types and actual notifications such as summons and subpoenas, depositions, dispositions of various kinds such as verdicts and sentences, financial obligations including fines, penalties, restitutions, progress schedules, receipts, deposits, balances, accounts, journal entries, balances, banks and bank accounts.

Court transaction, depicted in Figure 6.3, is a complex database object. When a court transaction is presented to the court is must be presented by a court participant. Court transactions are either filings or orders. Filings are presented to the court and orders are issued by the court. A filing can either be a document or a motion. An example of a document is the incident report that instigates the case. Other documents can be reports from the defendant, briefs by lawyers, reports by other agencies, and the like. A motion is the other type of filing. A motion is presented to the court by an attorney and requests a specific action by the court.

An alternative to the filing is an order. The court issues orders. One or more orders may be related to one or more motions.

Court transactions may be involved in a case event, or may result in an obligation (e.g., monetary fine) by a participant. The obligation is satisfied through some action.

6.1.4 Offense or Issue Management

Issues of law deal with the various statutes, codes, ordinances, and requests for enforcement brought before the court. These issues are based on state-wide, county, and local laws and ordinances.

The issue database object is portrayed in Figure 6.4. It consists of three tables, court matter, civil action, and offense. Court matters, that is, issues or offenses are defined and controlled by jurisdictions, such as municipalities, counties, and the state. If the matter is civil then it is completely cited through the civil action subtable. Otherwise the matter is criminal and is cited through the offense subtable.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

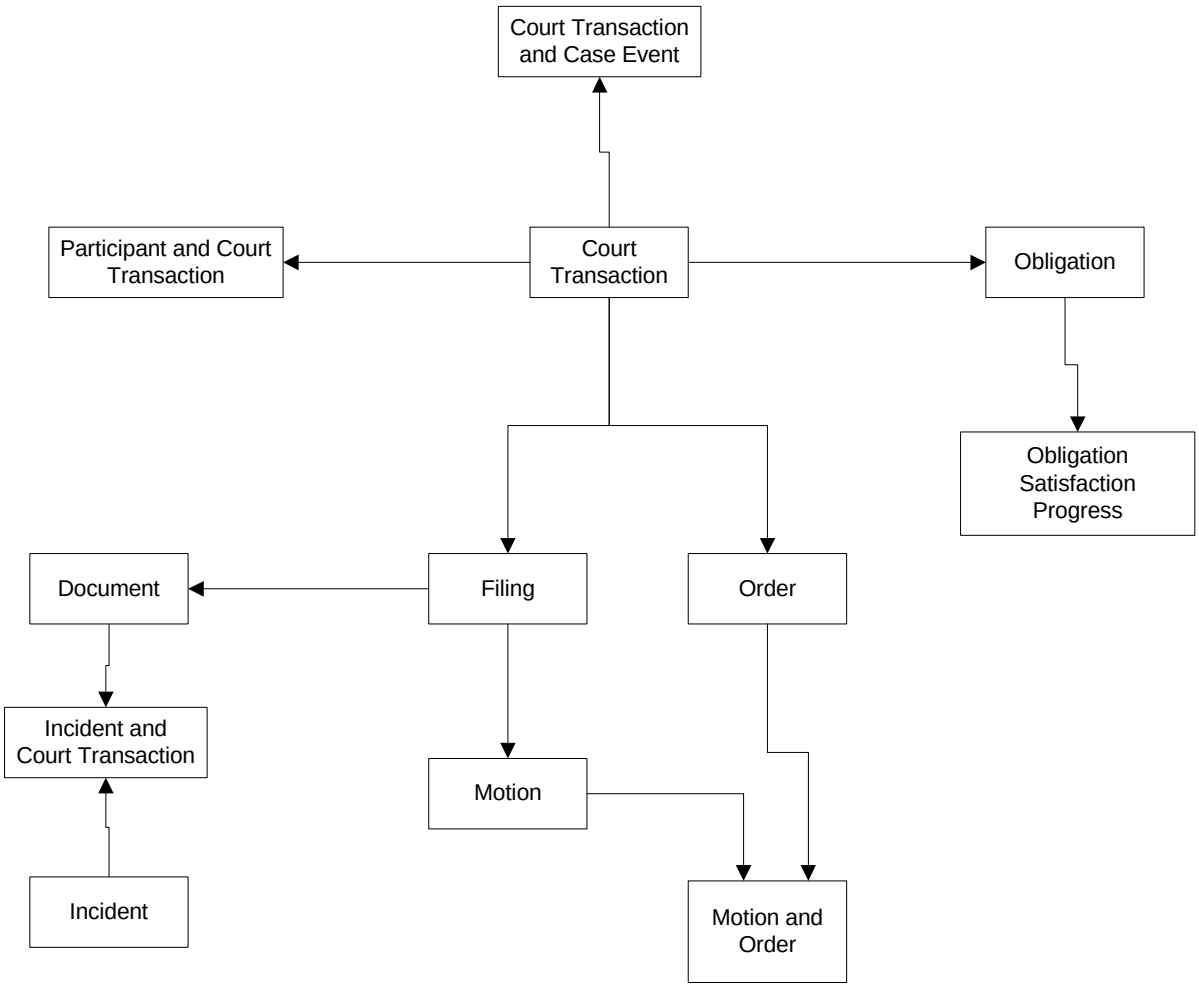


Figure 6.3. Case Transactions.

The court matter database object may be related to a court transaction that is part of case event. All persons involved with that relationship are also identified. While there are only three main tables in the court matter, there are a number of distinct relationship data structures to other database objects that cite the reason for the association and key audit information like the date, time, and person constructing the relationship.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

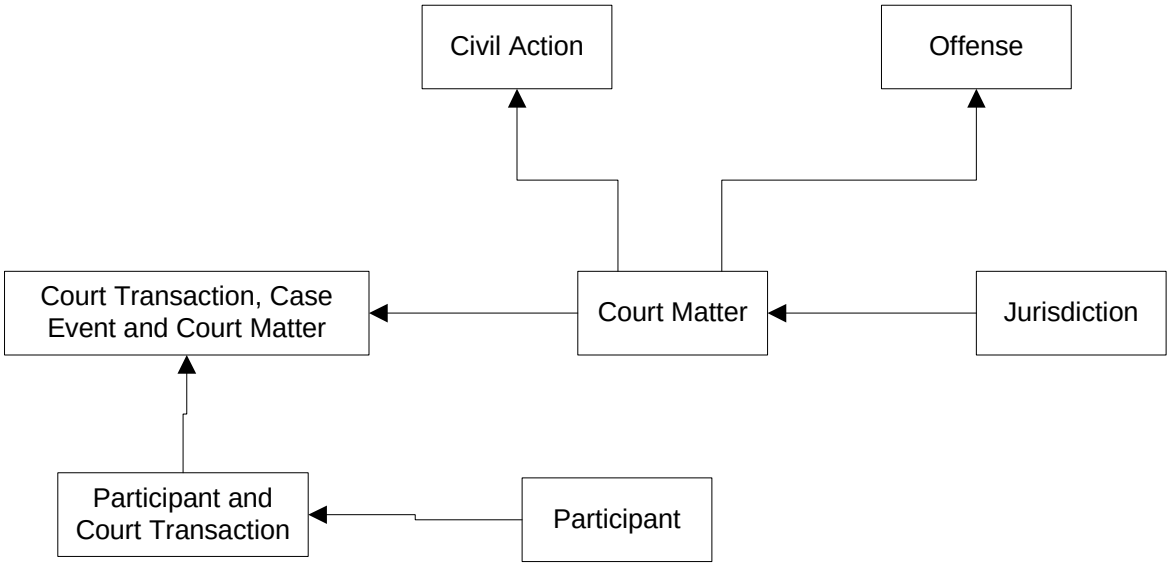


Figure 6.4. Offence or Issues.

6.1.5 Workload Management

Workload management are the derived assessments of the quantity, types, durations, and complexity of cases. Workload management provides support for scheduling, court capacity planning, the prediction of overloads. Because workload management is derived, it has no original data database objects, only derived data. Nonetheless, the data is real.

Figure 6.5 presents the tables involved in workload management. The three tables are:

- Court staff workload category
- Case categories and workload types
- Case categories, workload types and open and close reasons
- Case opening and close transactions

These three tables are typed through the following reference tables:

- Workload types
- Case category types
- Case open and close reason types

The Case opening and close transactions signal the beginning and ending of a case. The various case transactions in between provide the information necessary to know who, when, how long, etc. for all the different activities involved in the case. Once that information is known then the workload statistics can be computed against the assigned staff. These database objects are untraditional as they are completely created as a consequence of embedded rules and already stored data. Most database objects are directly created from externally fed data.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

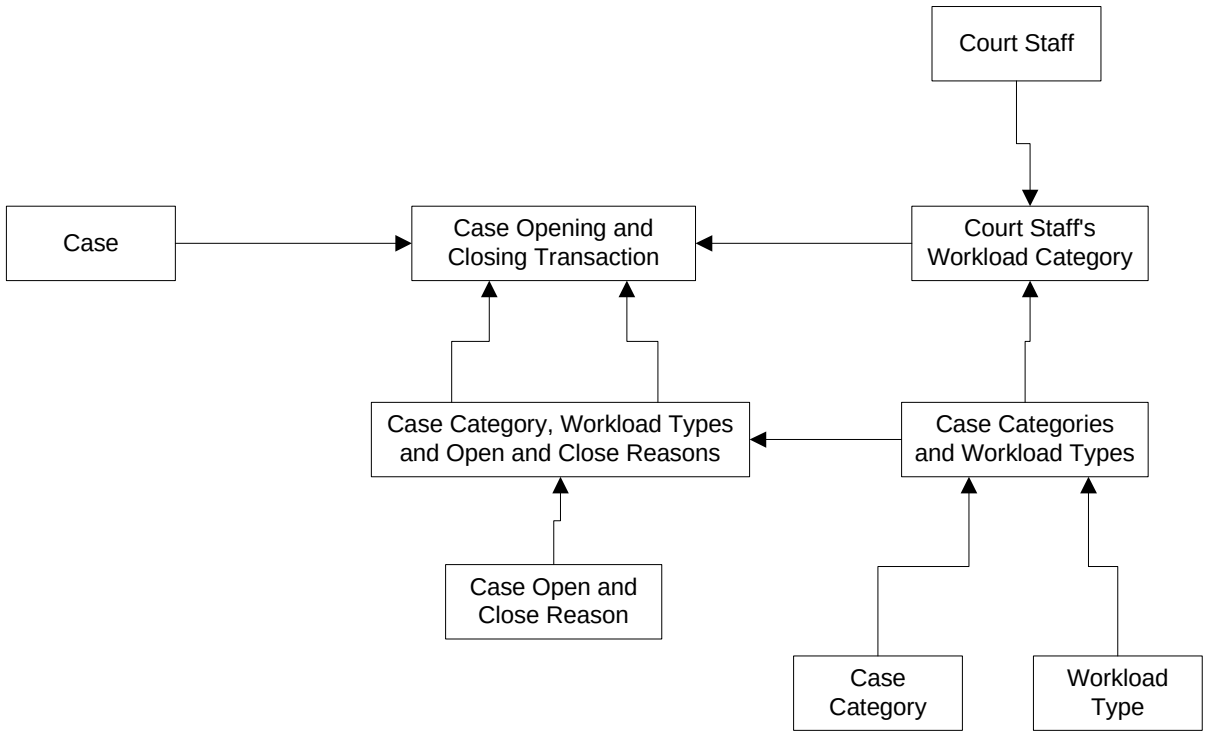


Figure 6.5. Workload.

6.1.6 Participant Management

Participants within the courts are persons, corporations, and organizations. Participants server may roles such as attorney or counsel, witnesses, defendants, jurors, litigants, co-parties, witness, victim, complainant, child, parent, guardian. Participants are described by their physical, mental, fiscal characteristics, and their relationships to other participants such as spouse, parent, child or other relative. Participants are also known by their addresses, employment, health conditions and assessments. Finally participants are the authors, subject of, or involved in various court transactions, the subjects of cases, and the members of court staff.

Figure 6.6 presents the database objects associated with participants. This database object is classic. It has a single significant root (participant), three subtypes starting with address information and the other two major exclusive subtypes (person or organization), and then the set of subtypes from person, which are:

- Identifying information
- Person unavailability dates
- Person as attorney information and its exclusive subtypes of judge and referee
- Court staff and its exclusive subtypes of judge or referee
- juror

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Associated with the person database object are the appropriate relationship tables that support connection to affiliations or organization, and court transactions.

The set of tables that fully exposit the roles a participant plays are:

- Participant's role type
- Role type
- Participant's role type and case event instance
- The relationship between instances of participant's role type and case event instance
- Inter-relationship basis for the relationship between instances of participant's role type and case event instance

The database object process logic is essential to properly record the correct set of policy-based instances and the relationships among them.

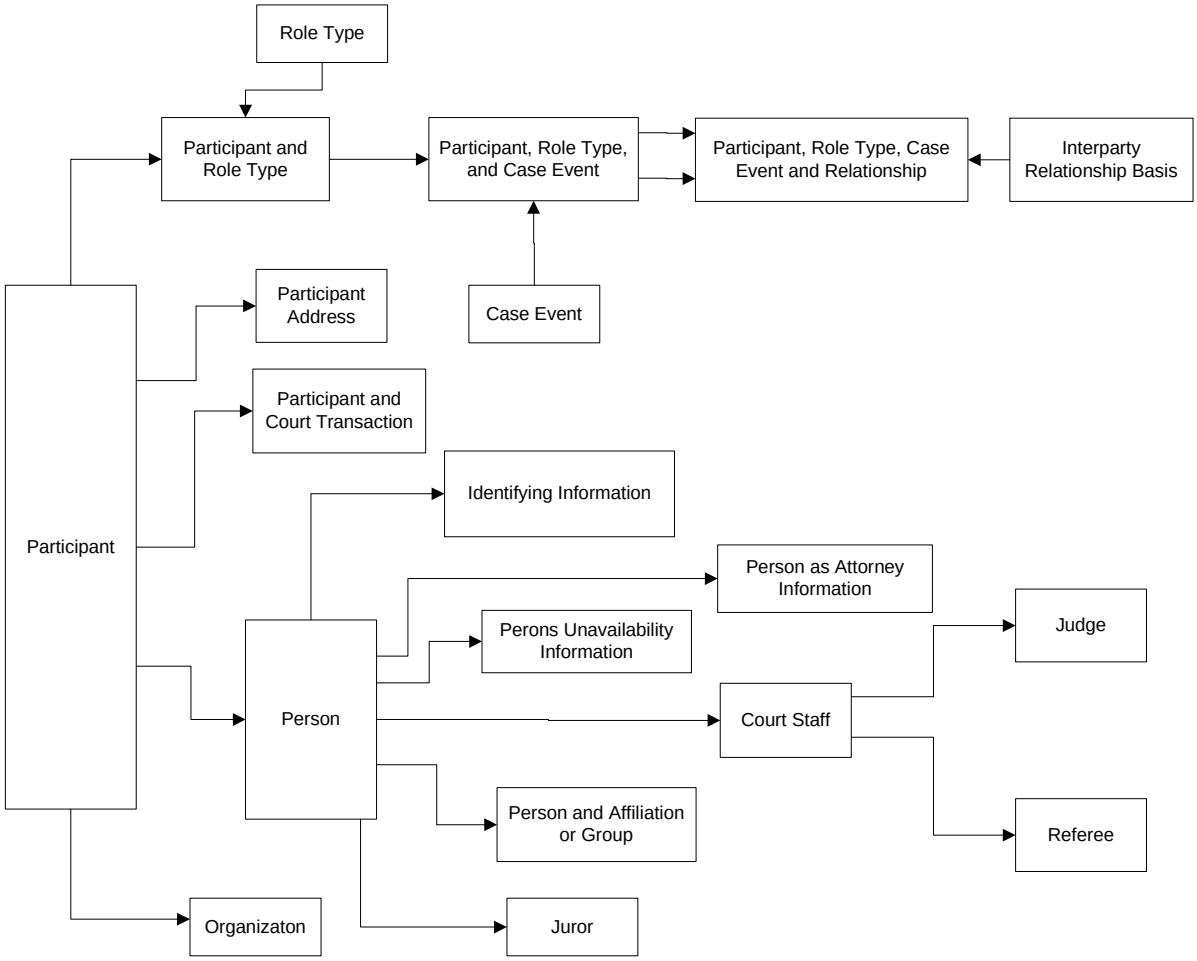


Figure 6.6. Participants.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

6.1.7 Notification Management

Participants in court proceedings, originators of court transactions, and those affected by court motions and orders all must receive proper notification of the court's actions.

Figure 6.7 Notifications are database objects that includes only two primary tables: notification and notification type. Related to notification are two previously identified database object tables:

- Participant address
- Participant role type and case event

In the case of participant address, the address to which the notification is mailed is retained in the notification instance because the "current address" of the participant may have changed from the address used for the notification. Address history is not however maintained. Only the current address and the notification contained address.

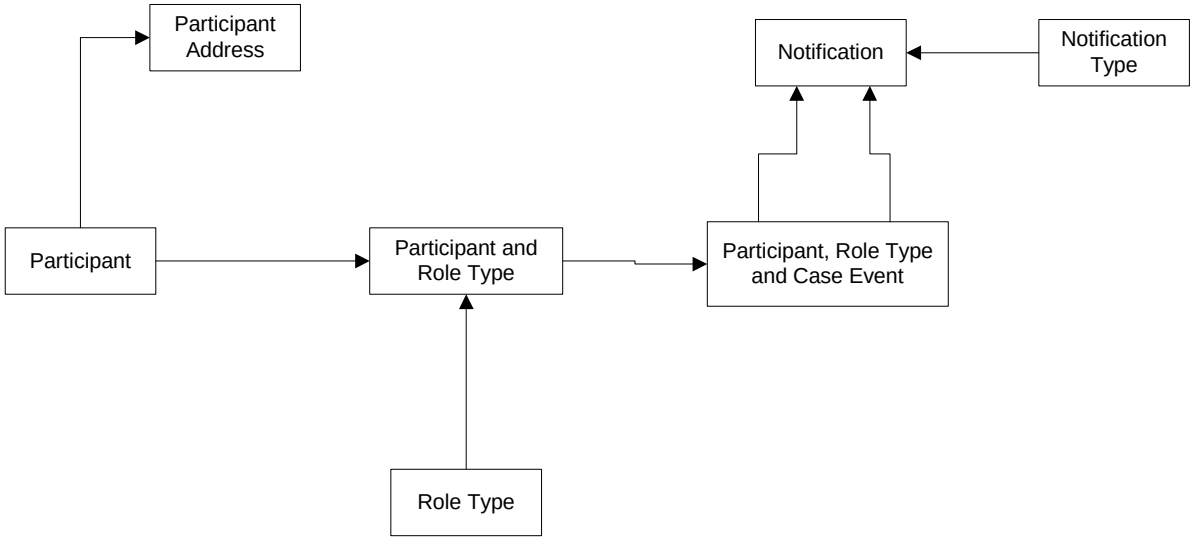


Figure 6.7. Notifications.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

6.1.8 Court Management

Courts are concerned with general and specific case schedules, case loads, jurisdictions, the various court types and divisions, the assignment of court room facilities, judges and referees as well as various court staff such as guards, bailiffs, and juries.

Figure 6.8 depicts the set of tables involved in court management database objects. The tables are:

- Appellate district
- County
- County court type and division
- Court division
- Court room
- Court staff
- Court type
- Court type and division
- Judge/referee assigned county court type and division
- Judge/referee assigned county court type and division and room

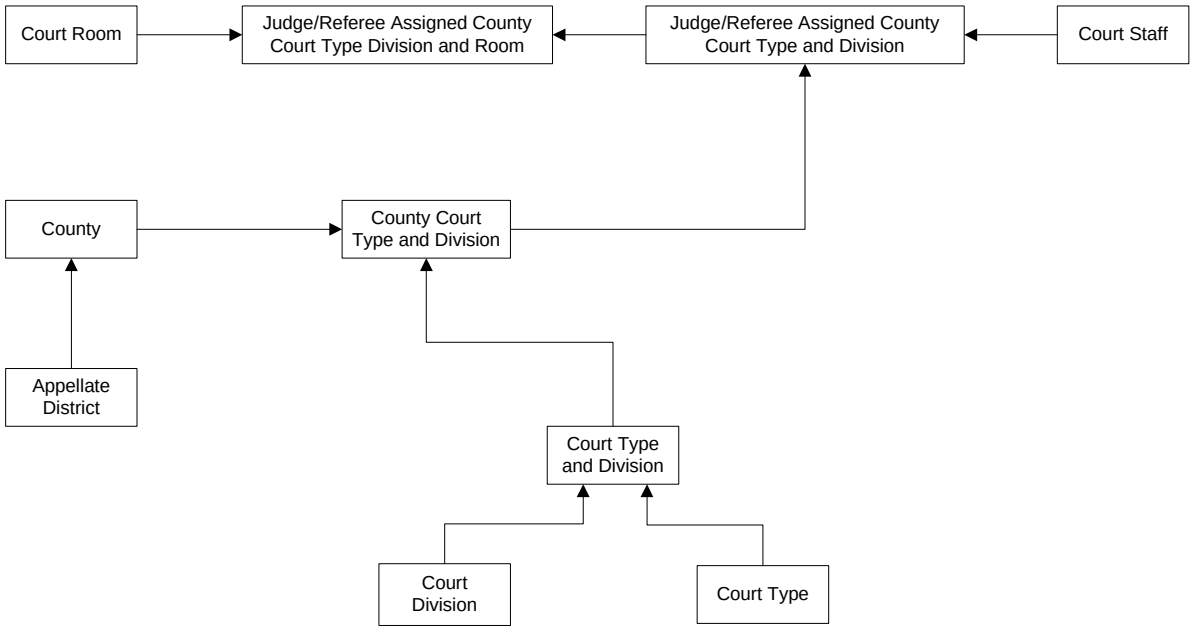


Figure 6.8. Courts

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Within this set, the original data database object tables are:

- Appellate district
- County
- Court division
- Court room
- Court staff
- Court type

The remaining tables are those created when specific relationships exist among the original database object tables. Collectively these tables provide positive control and management over the entire court facility throughout the state. As cases are scheduled, the specific court facility that is available for assignment is captured within the calendar management record, and that assignment and its critical information such as start time, end time, staff assigned, etc, is employed for computing overall workload statistics.

6.2 Database Object Processes

The database object processes are analogous to those described in the previous chapter. Every segment is supported by an add, delete, and modify. Additionally all valid values are checked, and all referential integrity rules are enforced.

This database has an entire suite of database objects that are created through derived data. That means that the database object processes for original data capture database object tables have to signal, in some way, the instigation of the database object processes to automatically build the workload statistics data when ever the original data is changed, deleted, or added. Placing this type of responsibility onto the database object process would break encapsulation. Therefore these responsibilities are the proper purview of database object information systems. This means that certain of the data integrity rules[21] must be enforced by the database object information systems who can have domain over multiple database objects.

6.3 Database Object Information Systems

The courts automation environments throughout the state were quite heterogeneous. Some courts had "dumb" terminals connected to mainframes. Others had stand alone PCS, and finally, some courts had multi-level client-server environments. Because of this great diversity, the major emphasis was placed on specifying almost all the court functionality within the database object information systems. This meant that the design had a very large database object footprint and a

[21] There are seven basic types of data integrity rules. They range from the most simple, that is, single column, single row constraints to the most complex, that is, multiple row and multiple table calculations and updates to column value. A full discussion of the rules in contained in Chapter 5 (Section 5.5, page 99-108) of *Enterprise Database in a Client Server Environment.* This book is now available from Whitemarsh Press.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

very small business information system footprint. The database object information systems were thus very closely related to the database objects and were:

1. Calendar
   a. Calendar Day of Year
   b. Local Court Events
   c. State-Wide Court Events
   d. Assign Trial to Day

2. Offense/Issue
   a. Jurisdiction
   b. Court Matters
   c. Offenses
   d. Civil Actions

3. Case Tracking
   a. Case Instigation
      i. Incidents
      ii. Cases
      iii. Case Types
      iv. All participants for case
      v. Incidents not yet cases
      vi. Cases resulting from incident
      vii. Case Journal
      viii. Case Report

   b. Case Events
      i. Case Event Types
      ii. Linkage of Case Types to Case Event Types
      iii. Case Events
      iv. Participant Assignments
      v. Court Transaction & Case Event Linkages

   c. Jury Panels
   d. Trials
   e. Consolidations

4. Court Transactions
   a. Court Transaction Entry
      i. Acquire Transaction
      ii. Classify Transaction
         (1) Determine Categories
         (2) Encode Categories
         (3) Encode Tracking Milestone Dates
   b. Document Filing
   c. Establishment of Filing as a Motion
   d. Orders
   e. Relate Orders and Motions
   f. Obligations
   g. Linking Court Transactions and Case Events
   h. Linking Incident with Court Transactions
   i. Linking Court Transactions with other Court Entities

   j. Court Transaction Reports

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

i. All Filings in a Case
ii. Motions without Orders
iii. Motions and Their Response Orders
iv. Orders not in Response to Motions

5. Workload
   a. Workload Types
   b. Case Categories
   c. Case Open/Close Reasons
   d. Linking Court Staff, Case Categories and Workload Types
   e. Case Transactions
   f. Workload statistics computation

6. Participant
   a. Persons and Organizations
      i. Organization Participants
      ii. Person Participants
         (1) Individual Participants
         (2) Identifying Information
         (3) Person Unavailability Dates
         (4) Attorney Information
         (5) Jurors
         (6) Affiliations & Groups
            (a) Affiliations or Groups
            (b) Link Persons with Affiliations or Groups
         (7) Court Staff
            (a) Court Staff
            (b) Attorney Information
            (c) Judges
            (d) Referees
            (e) Link Participants with Court Transactions
   b. Participant Addresses

   c. Participant Roles
      i. Role Types
      ii. Link Participants with Role Types
      iii. Participants by Role Type

   d. Participant Relationships
      i. Interparty Relationship Basis
      ii. Establish Interparty Relationship

   e. Notifications
      i. Notification Types
      ii. Notifications

7. Courts
   a. Appellate districts and Counties
   b. Court Types and Districts
   c. Linking counties with Court Types & Divisions
   d. Court Rooms
   e. Assignment of Judge or Referee to County, Court Type, and Division
   f. Assignment of Judge or Referee to Court Room

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

6.4 Database Object States

The key step in determining the set of states for database objects is to determine its life cycle. For example, if the court transaction was a document that was submitted on behalf of a defendant, the states that demarche the accomplishment of the resource life cycle[22] are:

- Identification
- Classification
- Storage
- Acted Upon
- Scheduled for Follow up
- Finally Determined
- Archived

After each state is determined, the database object information systems that accomplish the states are identified and connected. The database object information systems, in turn, cause the associated database object tables to be modified from one valid state to the next. When the entire state change is accomplished the state transformation transaction is said to be successful. Otherwise it is rolled back.

[22] A business resource, according to Ron Ross in *Resource Life Cycle Analysis, A business Modeling Technique for IS Planning* (Database Research Group, Inc., Boston, MA 1992) is "something of significant scope, substantial complexity, and enduring value that has actual substance and/or structural design, which enables the business to achieve its mission and carry out its functions. These functions manipulate, exploit, direct, and/or transform such resources. The resource life cycles are the basis from which database objects are identified, designed, implemented and deployed.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

7

SALES AND MARKETING DATABASE OBJECTS

At the highest level, the sales and marketing system is a computer based system that takes extracts of data from operational, direct end-user inputs, and other computer systems and stores this data into a database that has been designed for maximum data retrieval flexibility for field and central office sales and marketing staff. The database stores key information in a historical manner to support longitudinal analysis and studies.

The system serves two masters: direct and indirect reporting. Direct reporting is accomplished by obtaining and formatting data directly from the data warehouse through the use of a generalized report writer. Indirect reporting is accomplished by creating a specialized extract from the database to end-user specialized reporting tool.

Figure 7.1 presents the database objects for the sales and marketing environment. The three classes of database objects are:

- Sales Data Environment (Sales Persons, Sales Organizations, and Customers)
- Item or Product (Items/products and Inventory)
- Sales Data (Orders, Shipments, and Sales Finances)

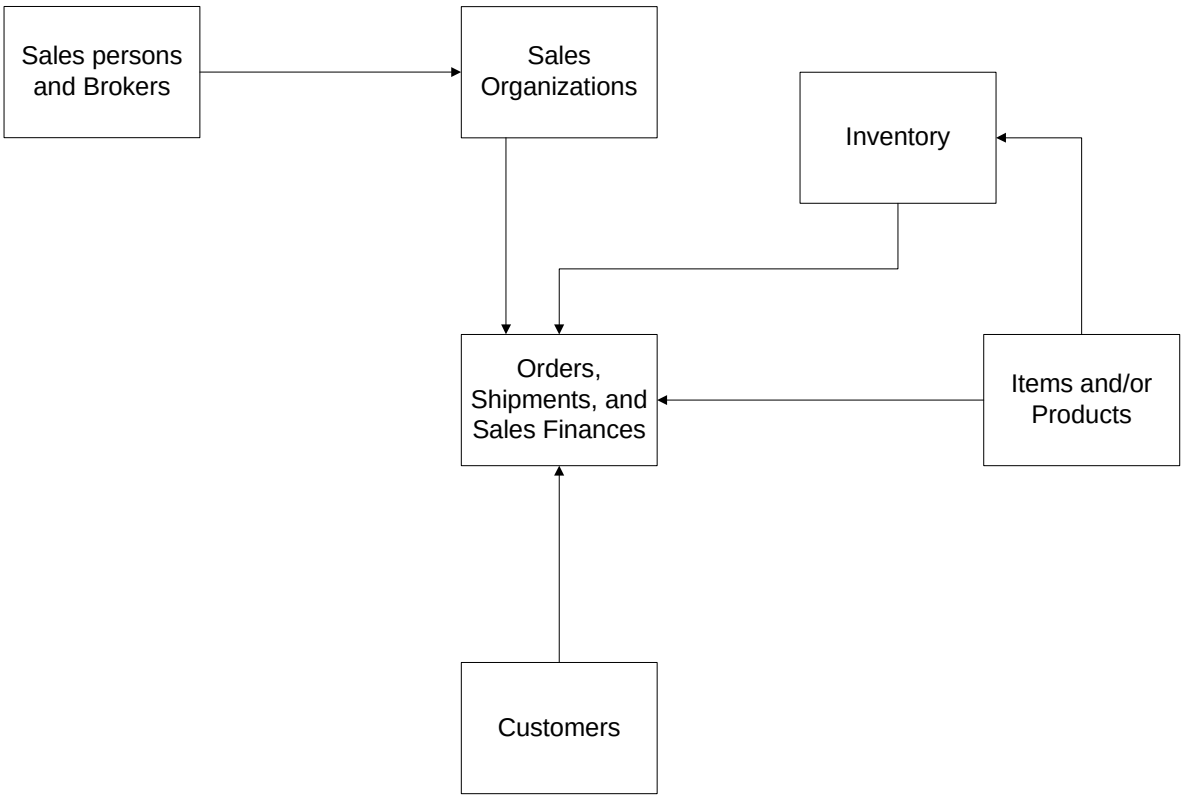


Figure 7.1. Sales and Marketing database objects.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

7.1 Data Structure

7.1.1 Sales Data Environment

The three classes of database objects within the sales data environment are:

- Salespersons and Third Party Sales Organizations
- Sales Organizations
- Customers

7.1.1.1 Salespersons and Brokers

The salesperson and broker database object tables depicted in Figure 7.2 consist of:

- Employee
- Position
- Position Assignment
- Position Type
- Sales Organization Position Assignment
- Sales Organization Position/third Party Customer Location Role Assignment
- Sales Organization Third Party Assignment
- Subordinate Third Party
- Third Party

The salesperson position hierarchy consists of position and position type. These database object tables enable the enterprise to define standard positions, for example, various types of titles such as manager and salesperson as well as the relationships among these titles. Manager might be further classified organizationally. For example, regional sales manager, and district sales manager. Once these are defined their values are installed in the entity: position type. Attributes might include the position type name, the salary range, and a reference to the list of skills

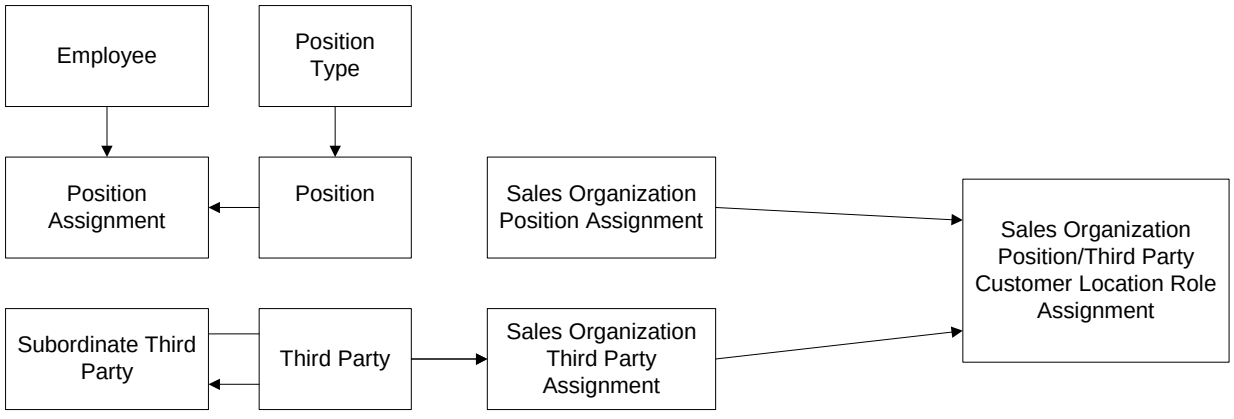


Figure 7.2. Salesperson/broker database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

required, etc.

The relationship among these values is installed in the subordinate position type database object table. The complete position hierarchy is materialized through the interaction of the position names in position type and the relationships among the position names in subordinate position type.

The entity, position, is used to record the fact that a person of that position type can be hired. Attributes might include the date of authorization, and the specific geographic or departmental location within the enterprise for whom the new hire is to work.

A straw-man proposal for position type are:

- Sales Manager
- Sales Person

There must be prefixes to sales manager. These prefixes are the standard names for the enterprise Standard Selling database object tables which is described below. An example is Region[al] Sales Manager.

The third party database object table is an database object table that represents an alternate provider of product orders. Salesperson is the other provider of product orders. A third party is typically a corporation who obtains orders for product from customers. Since a third party may be a corporation it may have its own set of offices and ultimately salespersons. The level to which the enterprise must know of the third party's organization is determined by the level to which sales data is to be aggregated. If, for example, subordinate third parties act as individual territories, then that level of granularity is required to report sales by subordinate third party. For example, a third party corporation might be contracted to sell to all customers in British Columbia, where the product is shipped to the third party's warehouse rather than to the individual customer location. If the enterprise desires to report the sales -- by individual third party and third party shipped-to -- within British Columbia then the third party must provide the enterprise their sales organization information such that the specific subordinate third party identifier and the third party's shipped-to is recorded on each order.

If the sales position hierarchies are installed and if all orders are required to report the individual salesperson/third party identifier then sales will be able to be aggregated by specific salesperson type.

7.1.1.2 Sales Organization

The sales organization database object tables depicted in Figure 7.3 consist of:

- Sales Organization
- Sales Organization Position Assignment
- Sales Organization Structure
- Sales Organization Structure Type
- Sales Organization Third Party Customer Location Role Assignment
- Sales Organization Third Party Assignment

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

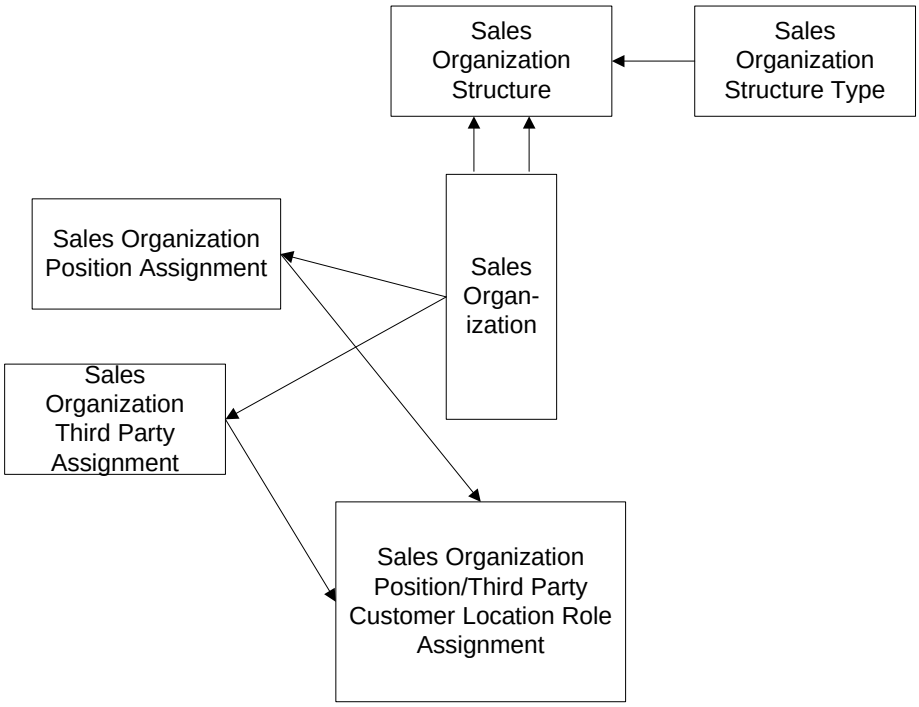


Figure 7.3. Sales organization database object.

Sales organization is an abstraction of a level in the enterprise sales organization. Either a salesperson or a third party is assigned to sell product to the customer locations assigned to the sales organization. There may be multiple selling database object tables on any sales organization level. For example multiple sales territories within a sales district, multiple sales districts within a sales region, and multiple sales regions within a sales division. The quantity of levels and the quantity of selling database object tables of a given type (division, region, district, and territory) is a management decision, and is based on the needs of a certain geographic area. The sales organization structure is more dense for Ontario than for British Columbia. And British Columbia would be more dense than Manitoba.

A sales organization is not a third party nor a sales person. Sale unit is a component of the enterprise sales organization that is staffed/operated by a salesperson or a third party that may have been assigned customer locations. Sales organizations may exist but never staffed. While clearly that makes sales difficult, they are still not the same. Sales organizations are most often divided according to geography. Thus, the following hierarchy is practical to consider:

- Domestic
    - Regional
        - Provincial
            - District
                - (sales)Territory
- International
    - United States

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Europe
  - England
  - France
- Asia

It is also likely that while there are 10 provinces, not all 10 would have districts, or even territories. Consider Nova Scotia, New Brunswick, Prince Edward Island, and New Foundland versus Ontario or Quebec? This would have to be determined based on standard management practices such as *span of control* that exists within the enterprise. For example, it is possible that Ontario, if it is serviced entirely by employees might have a first level sales manager for every 8 employees, and a second level manager for every 8 first level mangers. But, for third parties, the ratio might be one first level sales manager for every 30 third parties. The difference in ratios might be because "human resources" management is not of concern for third party management, but is for human resource management.

Based on the maximum quantity of selling database object tables of a given type (third party or salesperson), the determination can easily be made about the existence of a sales management level and the quantity of sales mangers at a given level.

The named instances of each sales organization level for each in-place sales organization is recorded in the Sales organization database object table. The relationship among all the selling database object tables that are in-place for is recorded in the Subordinate Sales organization. The real and implemented the enterprise sales organization is printed through a computer program that uses data from both sales organization and subordinate sales organization. Sale unit provides the names, and subordinate sales organization provide the hierarchy among the names.

There are two common types of sales organization structures. The data model allows for both. These sales organization structures are:

- Geography Based
- Customer Organization Based

Geography based sales organizations are the most.

The organization of customers, for example Kmart, is not the same as the enterprise sales organization for Kmart. The first is a grouping of all Kmart customer locations that facilitates reporting. The second is the named organization of the enterprise selling database object tables targeted to only one customer organization, Kmart. Sales tracking and customer-wide discounting is possible under both structures. If, however, sales goals and sales quotas are established for various Kmart locations, then actuals against plan requires a dedicated sales organization. If there are no sales organizations that are dedicated to, for example, Kmart, then the customer organization structure as addressed in Section 7.1.1.3, Customers, is sufficient.

The two database object tables, sales organization type and subordinate sales organization type are employed to store a particular hierarchy of a sales organization (geographic or sales channel), without regard to the quantity of individual selling database object tables on each level. The difference between: sales organization type and subordinate sales organization type and the two database object tables sales organization and subordinate sales organization is the difference between template and actual use of the template. For example, the Ontario region might have districts and territories and this Ontario sales organization template is stored in sales organization

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

type and subordinate sales organization type. The actual implementation of the Ontario sales organization might be six districts and a total of 40 territories distributed over the six districts. This actual Ontario sales organization is stored in sales organization and subordinate sales organization.

The two database object tables, Enterprise Standard Sales organization Type and Enterprise Standard Subordinate Sales organization Type are employed to store the enterprise standard sales organization (geographic, customer, or sales channel).

Specific customer locations might "belong" to multiple selling database object tables. For example, a specific Kmart store might be in the sales organization: Northwest Toronto, and Kmart Canadian National sales. How this shared "ownership" is managed (that is, credit for sale and splitting of commissions) is a management policy issue, not a technology issue.

In summary, the enterprise's national office defines the values and relationships for its various sales organizations. Then each next level down sales manager might be empowered to create their own proper subset of the enterprise standard. A subset is proper if it is either the whole set or is subset but with the same ordering.

The assignment history of a specific third party and/or salesperson to the sales organization is accomplished through the third party & sales organization, and through the salesperson & sales organization. This many to many relationship allows for the complete capture of all history of assignments, and for comparing the performance of different salespersons that have migrated through a territory, the comparison of different third parties, or finally the comparison of third parties and salespersons.

7.1.1.3 Customer

The customer set of database object tables consist of three types:

- Customer
- Customer Groups
- Sales Data Classifications

7.1.1.3.1 Customers

The customer database object tables depicted in Figure 7.4 include:

- Customer
- Customer Location
- Customer Location Address Information
- Customer Location Call-upon Information
- Customer Location Role
- Customer Location Telecom Information
- Customer Structure
- Customer Structure Type

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

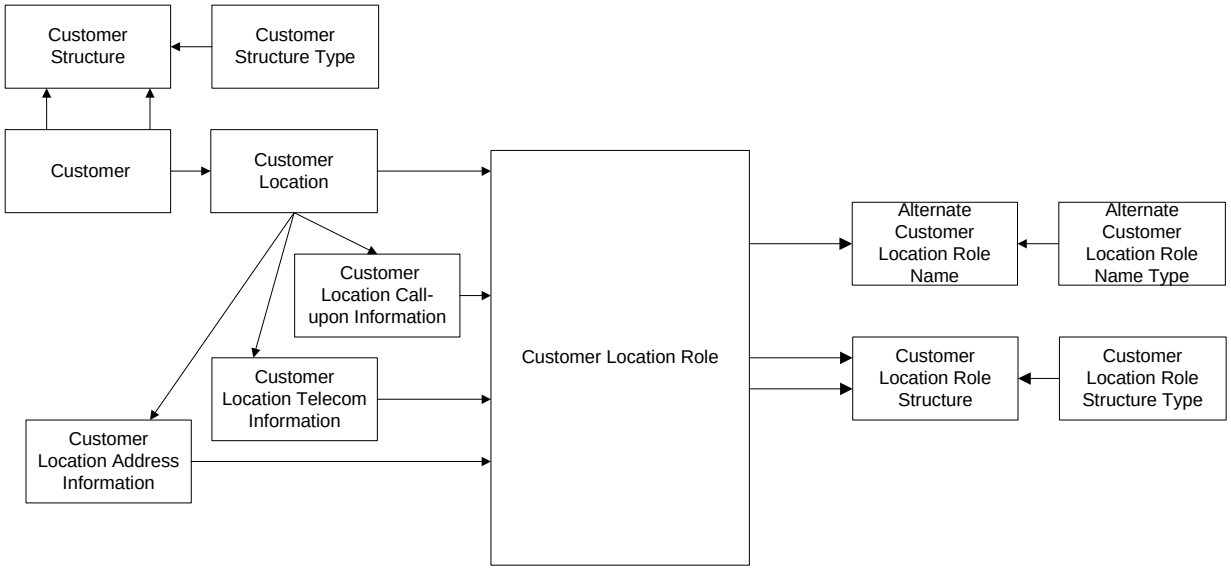


Figure 7.4. Customer database object.

These database object tables exist in groups. The first group, customer, customer location, customer structure, and customer structure type store the hierarchy of all those locations. At the top could be the Canadian National organization for Kmart. The next level down could be the provincial sales offices, and so forth. The names of all the customer locations are stored in customer location. The relationship among all the locations is stored in customer structure and customer structure type.

The enterprise has established names of the various levels of the customer location hierarchy. These standard names and the hierarchy are stored in alternate customer location role name and alternate customer location role name type.

Not all customers conform to the enterprise standard. Either because they contain more levels than the enterprise standard, or less. The actual set of the enterprise roles and relationships among those enterprise roles that apply to the actual customer are stored in customer location role structure and customer location role structure type.

The customer locations and the sales organization are connected by a many-to-many relationship database object table sales organization & customer locations between sales organization and customer location. This many to many relationship enables the retention of the complete history of sales organization assignment to customer locations. In essence, the set, one sales organization and one or more customer locations forms a territory for the associated salesperson or third party.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

7.1.1.3.2 Customer Groups

Customer groups depicted in Figure 7.5 consists of the following database object tables:

- Customer Group
- Subordinate Customer Group
- Customer Group Assignment
- Customer Location Role

Customers form buying groups for various reasons, such as shipping, distribution, shared advertising costs, and discounts. A particular set of customers within a shopping center might coordinate all their orders through a single buyer. This customer group is akin to a customer's own buying organization in which orders are consolidated into larger orders to which greater discounts apply. The name of the customer group is stored in the customer group database object table. The relationships among various levels and peers of customer groups is stored in the subordinate customer group database object table.

The relationship between the various buying groups and the customer location is stored in customer group and customer location. This enables both storing multiple customer groups for each customer location and also a customer group history.

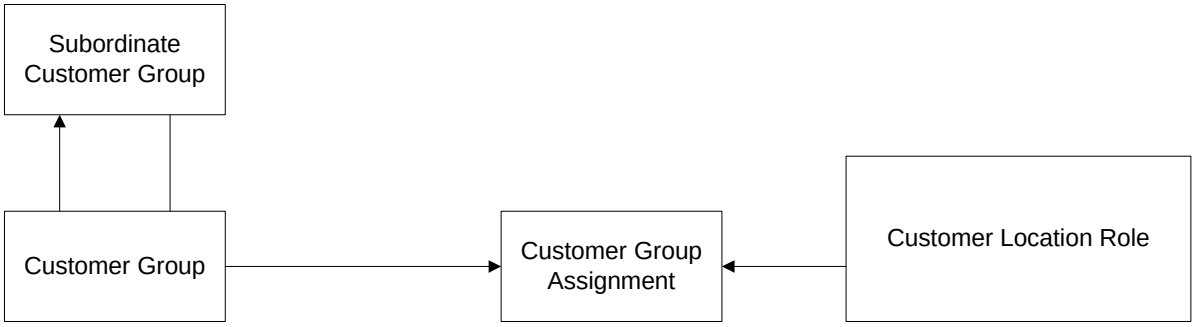


Figure 7.5. Customer group database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

7.1.1.3.3 Sales Data Classifications

Sales data classifications depicted in Figure 7.6 consists of the following database object tables:

- Customer Location Role
- Customer Location Role Sale Statistics
- Customer Location Role Item Sales Statistics

Sales data from customer locations are classified differently. These sales data classifications are mechanisms for reporting sales data by customer location. A sales data classification characterizes the specific customer location as being, for example, a pet care store, or a pet department in a multi-department store, or a veterinarian. As to a veterinarian, the veterinarian could either be an employee of a zoo, or be operating from within a medical research organization, operating out of a multi-department super pet-store, or be in private practice.

The database architecture permits sales data classification hierarchies through the database object tables, sales data classification and subordinate sales data classification. The database object table, sales data classification and customer location enables multiple sales data classifications to be assigned to each customer location. Attributes in this database object table would include start and end dates of assignment person making the assignment, and a notation of the assignment reason.

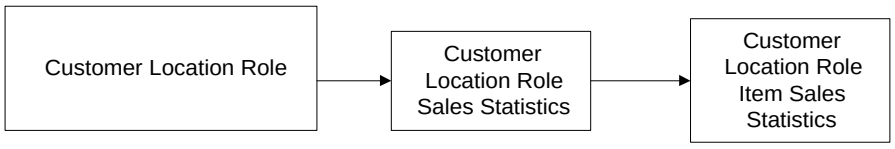


Figure 7.6. Sales data classification database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

7.1.2 Item and Inventory

The item and inventory data consists of all products, their composition hierarchies, and the deployment of all finished goods throughout the warehouse system.

7.1.2.1 Items

The item data depicted in Figure 7.7 consists of ten database object tables

- Component Item
- Component Item and Item Variant
- Item Classification
- Item Classification Assignment
- Item Classification Structure
- Item Classification Structure Type
- Item Composition
- Item Variant Type
- Item Variant Group (Product)
- Unit of Measure

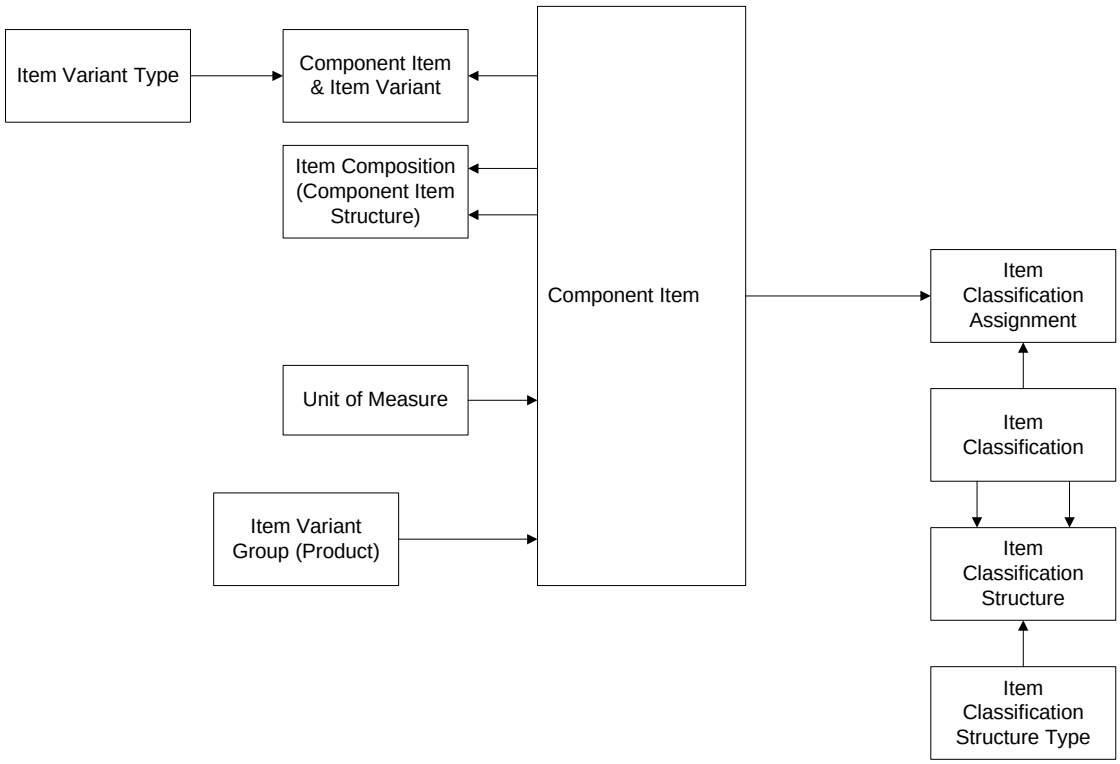


Figure 7.7. Item database object.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

A fully defined item contains item, brand, and packaging information. Each represents a hierarchy. The first hierarchy is item. Brand becomes part of the item's definition when the product is no longer generically divisible. Packaging is introduced after brand, but before the ultimate end item.

Item and Item Change History contains both the current information on an item and its change history. Change history is retained only if the change affects sales data reporting. Each item record contains an effective start date and an effective end date. Every order detail record contains a reference to a item record so that the ordered item's description and other relevant information can be reported.

Subordinate item contains the relationships that exist among the subordinate instances of the item. The apex of an item may be the major divisions of product such as snack food, pet care, and food. The next levels further subdivide the product until brand and then packaging becomes appropriate.

Brands and the brand hierarchy is stored in brand and subordinate brand. Packaging and the packaging hierarchy is stored in package and subordinate package.

Sales data is collected at the order detail row, which is referenced from the bottom most level of the item hierarchy. Because of these item, brand and packaging hierarchies, statistical roll-ups are possible up to the top most level of item, or to the top most level of packaging, or to the top most level of brand.

7.1.2.2 Warehouse and Product Distribution

The warehouse and product distribution database object tables are depicted in Figure 7.8. This data consists of the following tables:

- Order/shipment Detail
- Warehouse
- Warehouse Component Item Inventory
- Warehouse Customer Location Role Assignment

The warehouses are distributed by general and specialized product types. Customers, by customer location role are assigned specific warehouse for primary product distribution. The actual inventory of product by ware house is contained in the warehouse component item inventory, and the product distribution history is represented by the order/shipment detail.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

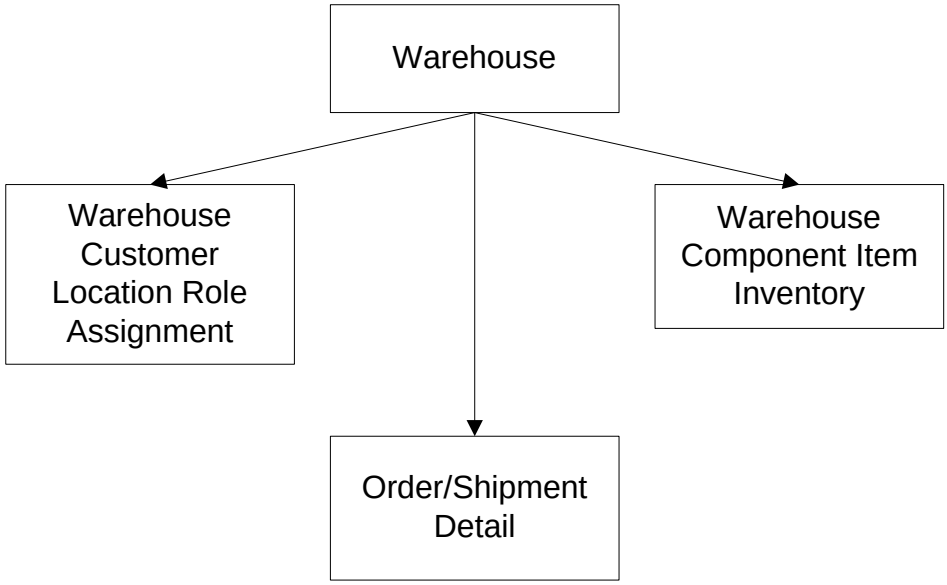


Figure 7.8 . Warehouse and product distribution database object.

7.1.3 Sales Data

Sales data depicted in Figure 7.9 consists of the following database object tables:

- Customer Location
- Customer Location Address Information
- Customer Location Call-upon Information
- Customer Location Role
- Customer Location Role Item Sales Statistics
- Customer Location Role Sales Statistics Order/shipment Header
- Customer Location Tele-com Information
- Customer Structure
- Customer Structure Type
- Invoice Header
- Order/shipment Detail

The order header and order detail data is obtained directly from operational systems. Each order detail record refers to an item record for ordered item information.

The customer location role item sales statistics database object table represents a fiscal summary of all item sales activity to a particular customer location. The word sales in this usage means ordered, shipped, invoiced, returned and canceled. These five sales measures provide a complete picture of the business transactions for the particular customer location. This database object table provides item level detail to the business transactions. Item description information is available through access to the item.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

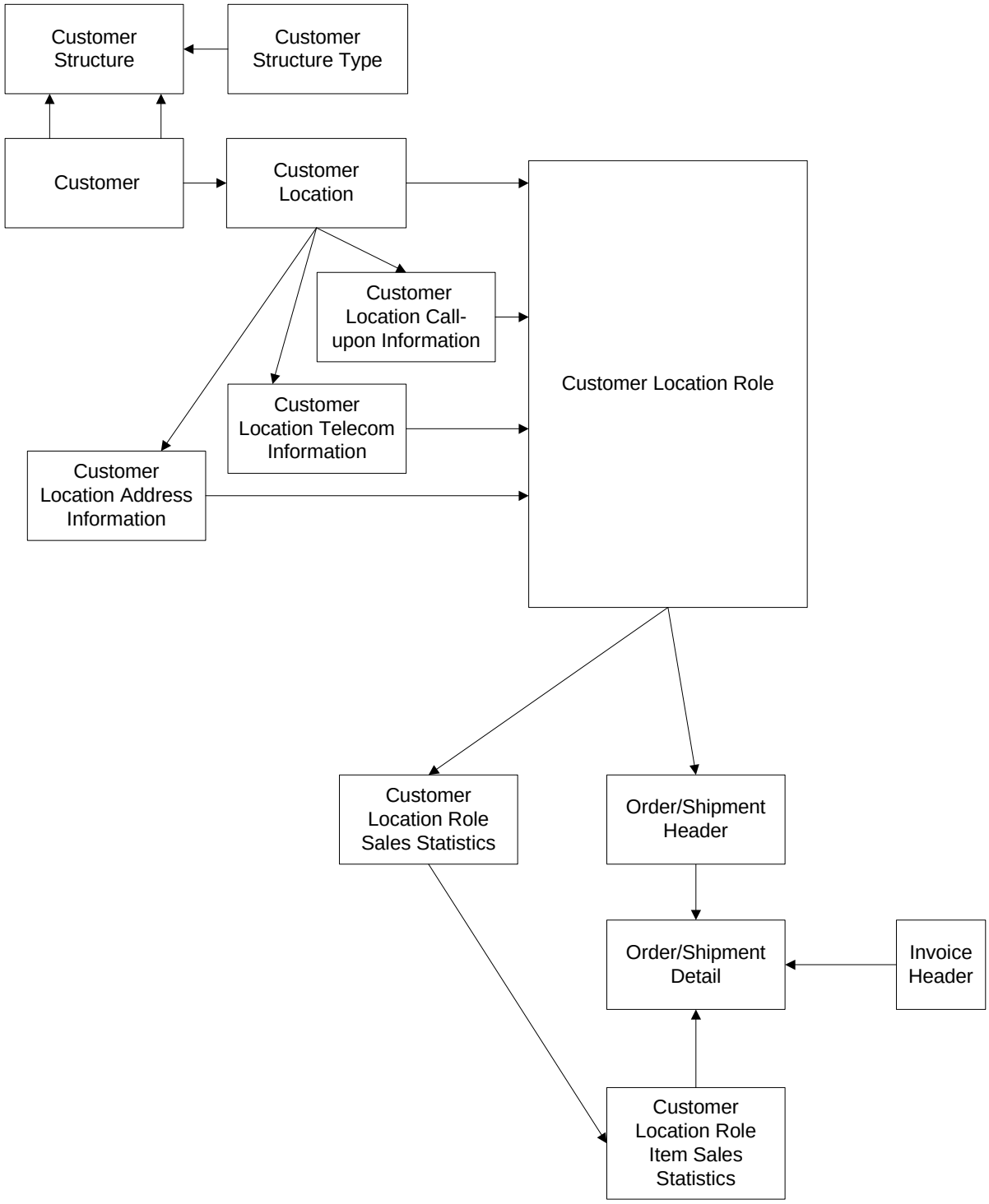


Figure 7.9. Sales database object

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The customer location role sales statistics database object table is a summary level record to the customer location role item sales statistics database object table.

The customer location role sales statistics can only provide the dollar summary of the impact of all ordered, shipped, invoiced, returned and canceled product. At the customer location role item sales statistics level, not only is this information available on an item basis, but also other information is available such as quantities of cases, tonnes and promotion indicators. The customer location role item sales statistics record cannot however indicate which promotions, as this information may vary across the order detail records from the various orders that were employed to create the information for the customer location item sales statistics row.

The order detail record provides information about each ordered item such as the specific quantity, and the specific promotion that may have affected the price.

A complete current and past business profile can be created for each customer location. For example, if a customer location placed five orders in one day then the customer location sales statistics record would contain a summary of all those orders. The customer location role item sales statistics would contain the summary level sales information by item. And to determine the exact nature of the orders for that day, the required data is:

- All the order detail records that were used to derive the customer location item sales statistics
- The order header records that support the order detail records
- The item records that are explanatory of the items.

Weekly, period, quarterly, and annual sales statistics are determined by summing all the daily sales statistics records for the week, period, quarter, and year. If these same statistics are required for a particular item, then the specific order detail records have to be selected based on both the customer location and item components of the order detail row's key.

In addition to the *actual* information about ordered, shipped, invoiced, returned and canceled product dollars in customer location sales statistics and that same information by item in customer location item sales statistics, these same two database object tables also provide *planned* and *forecast* information. *Planned* and *forecast* information can be established for existing selling database object tables with assigned customer locations, or can be established through modeling for proposed/future selling database object tables with customer locations that represent hypothesized classes or types of customers that are predicted to materialize in a proposed configuration of sales organizations.

7.2 Database Object Processes

Again, like the database object processes in Chapters 5 and 6, the extent of their specification is to enforce all valid values, invalid values and the myriad of referential constraints.

This set of database objects also contains significant derive data within the sales data database object tables. However, this derived data is created from "outside" the domain of the

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

contained database objects. Therefore, these derived data are treated just like any other externally captured original data.

7.3 Business Information Systems

The business information systems are similar in scope and intent as those in Chapter 6. However, a significant quantity are not instigated through on-line transactions. Rather, they automatically invoked through batch processes that are triggered by other systems like item classification, product inventory, on-line ordering and product order distribution. Additionally, there are a number of client based systems that maintain the current status of customer organizations, and sales organizations.

These systems update their own databases and then prepare TDSA data sets (see Section 3.7, Enterprise Data Architectures Evolution). These TDSA data sets are automatically loaded into the sales and marketing database. Finally, the database contains opportunities for end-user analysts to record interpretations regarding sales organizations and product sales performances. Thus, the database is a subject area database rather than a strict wholesale data warehouse.

7.4 Database Object States

The database states for sales and marketing range from simple to moderately complex. For example, the set of states for orders and shipments are merely that the order/shipment is recorded or not. An example of a moderately complex state is the customer location role. A role within an organization can range from one role for every named function as would exist in a simple retail outlet to a large organization that has executive, shipping, receiving, ordering, finance, etc. In this case the set of states for a large customer must deal with all these roles, the sequencing of the roles, and the relationships among the roles. The states are also moderately complex for sales persons, positions, and the relationships that exist between sales organizations, sales persons/third parties, and customers. This database object table, Sales organization position/third party customer location role assignment is probably the most critical within the entire sales and marketing database. Ensuring that its state is correct given the feeding states is essential to the database's usefulness.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

8

DATABASE OBJECT SUMMARY

The sequencing of the Whitemarsh methodology models is very carefully crafted to achieve the following four database objectives:

- Accomplish the maximum leveraging of prior work
- Minimize the political conflicts until they are seen as not mattering
- Force attention onto the fundamental policy issues of the enterprise that are derived from the enterprise's missions
- Allow differing styles of enterprise database accomplishment through individualized business function and organization models.

Because of these database objectives, the enterprise specification models exist in the following sequence:

- Mission
- Database Objects
- Business Information Systems
- Business Events
- Business Functions
- Business Organizations

The business function, and business organization are not last by accident. The mission model is done first because it defines the overall scope and domain of the entire enterprise database effort. The strategy by which mission is developed is apolitical. The mission descriptions are not to contain either *who* or *how*.

The database object model is created second, which represent the policy proofs that missions are accomplished. Creation of this model is not a political or functional activity, database object models can be database objectively proven to be correct.

The business information systems model is apolitical as well because it is restricted to represents the accomplishment of database object state transformations from within the context of business functions. Different business functions may cause the execution of the same business information system. If, in any of the business functions that employ a business information system, the database object value state transformation is not accomplished, the entire set of database object transformations are rolled back to their previous state.

The business event model serves as the interface between the business information systems model and the business function model. It further enables an information system to be employed by multiple business functions, and a business function to employ multiple information systems.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The business function model is political. That is, it is a design calculated to achieve a particular scheme of data transformations and/or reporting. Because no two organizations would ever agree on the minute details of a business, it does no good to attempt to force a common set of business function definitions. Consequently, the paradigm of the Whitemarsh methodology allows multiple, yet equivalent, business function hierarchies. The test of equivalence is simple: Are the results the same?

The organization model is truly political. Organizations are essentially styles of arranging people to accomplish particular business functions. Like a business function, the Whitemarsh methodology allows for the creation of different organizations that achieve the same business functions. For example, if a large company has a local office in which all aspects of human resources, finance, and many support services are accomplished by a single office, does it matter that the organization is *one person doing all things*? Back at headquarters there may be separate offices for each of the business functions. Again, that should not matter. With the paradigm set out in this book, the information system is independent of the arrangement of the business function, and the orchestration of the business function is independent of the corporate organization.

The business function and organization models are purposely done last so that there already exist large bodies of agreement, accord, and compromise within the information systems, database object processes, data models, and missions.

It is essential to have the organizational model as independent as possible from the business function model. If dependent, then as the organization changes so too must the business functions. This type of dependency is all too common with the consequence of too many, unnecessary enterprise database functional reorganizations just to reflect the organizational style changes.

Similarly, the information systems model must be independent from the business functions model. Independence prevents functional changes—commonly know as *management style*—from having an affect on information systems content and structure.

The mutual independence of the six models (mission through business organization) is crucial to the proper definition of enterprise database. These models must be defined in a manner completely unconstrained by technology so that they do not bias any of the possible implementation alternatives.

If both these characteristics exist in the six models, they can form the platform from which enterprise database can be efficiently and effectively deployed in any of the four different client/server environments. Heterogeneous organizations often require different and coexistent implementations of the same functionality. Different implementations are very difficult and/or impossible if the foundation models are defined in a technology dependent manner.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

9

METHODOLOGY SUPPORT

Methodology is a collection of actions whose sole purpose is to produce product. The foundation of the Whitemarsh methodology is the meta model of the repository described in Chapter 10. As stated in Chapter 1, data is executed policy, and the policy basis of enterprise database are the products necessary to accomplish it. Consequently, the repository's meta model is enterprise database policy. Methodology are just the procedures to carry out policy. The key questions that must be answered about methodology include:

- Are the generated products of high quality?
- Are the products generated in a cost-effectiveness manner?
- Do the products support enterprise-wide information systems?
- Are the steps in the methodology organized to accomplish "just-in-time" products?
- Are the products accomplished with maximum leveraging from prior built products?
- Are the products built through easy-to-understand, common-sense based steps?
- Is the methodology engineered to accommodate multiple projects against the same enterprise database?
- Can the methodology support multiple, concurrent projects in different stages of development: requirements, assessment, design, implementation, and delivery?

The measure of effective enterprise database project management is the optimal application of methodology to produce the highest quality, lowest risk, and lowest cost product set. Quality methodologies are metric based, predictable, and cause the capture of work-accomplishment statistics so that future projects are of even higher quality, lower risk, and even lower cost.

Repository, methodology, and project management are the three faces of a single effort. All are critical and all must be done in concert. This chapter presents the methodology necessary to accomplish the work products of the repository.

The Whitemarsh Methodology supports specification of database objects through the following technology independent models:

- Mission
- Database object
  - Data Structure
  - Process
  - Information system
  - State

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- Business Information System
- Business Event
- Business function, and
- Organization

These models are built in the order listed. The models' design and build sequence has been developed over the past 25 years to maximize consensus, and, at the same time minimize political battles. Because of the Whitemarsh methodology, enterprise database models are built faster, and with the highest quality.

To create complete business information systems, these database object specifications are bound to technology (classes of client/server, mainframes, specific DBMSs, network operating systems, etc.). The models resulting from the technology binding process are:

- Implemented data
- Implemented information systems
- Security, and
- Deployed hardware and networks

Figure 9.1 depicts the relationship among all the models. Because the database object models (mission,.., organization) are built separately from the implementation models the database objects are able to be implemented--differently--on different physical platforms (PC and DBMS, power-server and DBMS, mini-computer and DBMS, and main-frame and DBMS). Traditional techniques would have caused a different design for each platform.

The six models above the line in Figure 9.1 are the technology independent models. The models below the line are the technology dependent models. The reason the models are separated is to enable a single technology independent specification design to be transformed to one or more technologically dependent implementation designs. This reality is required because a given database could be implemented on a PC, small to very large server, or a mainframe. Because of fundamental hardware, operating system, and DBMS internal differences there is an absolute need for multiple and different technology implementation dependent designs.

The relationship between the Whitemarsh methodology phases and these critical models is depicted in Figure 9.2. These models are built and stored in the repository such that they can be continuously expanded and improved upon throughout the application life cycle. These models are also built such that they can be used whenever required for one or more information systems.

A final characteristic of the Whitemarsh meta models is that they are related to each other through many-to-many relationships. That means that metadata, defined once, may be involved in many different aspects of the mission model. An information model may involve different database objects (appropriate combinations of metadata and database object processes). This enables all database objects to be *defined once but used many times*. Such a strategy has been show to maximize consistency, minimize redundancy, eliminate unnecessary errors, accelerate systems implementation, and make maintenance many times more certain and efficient.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

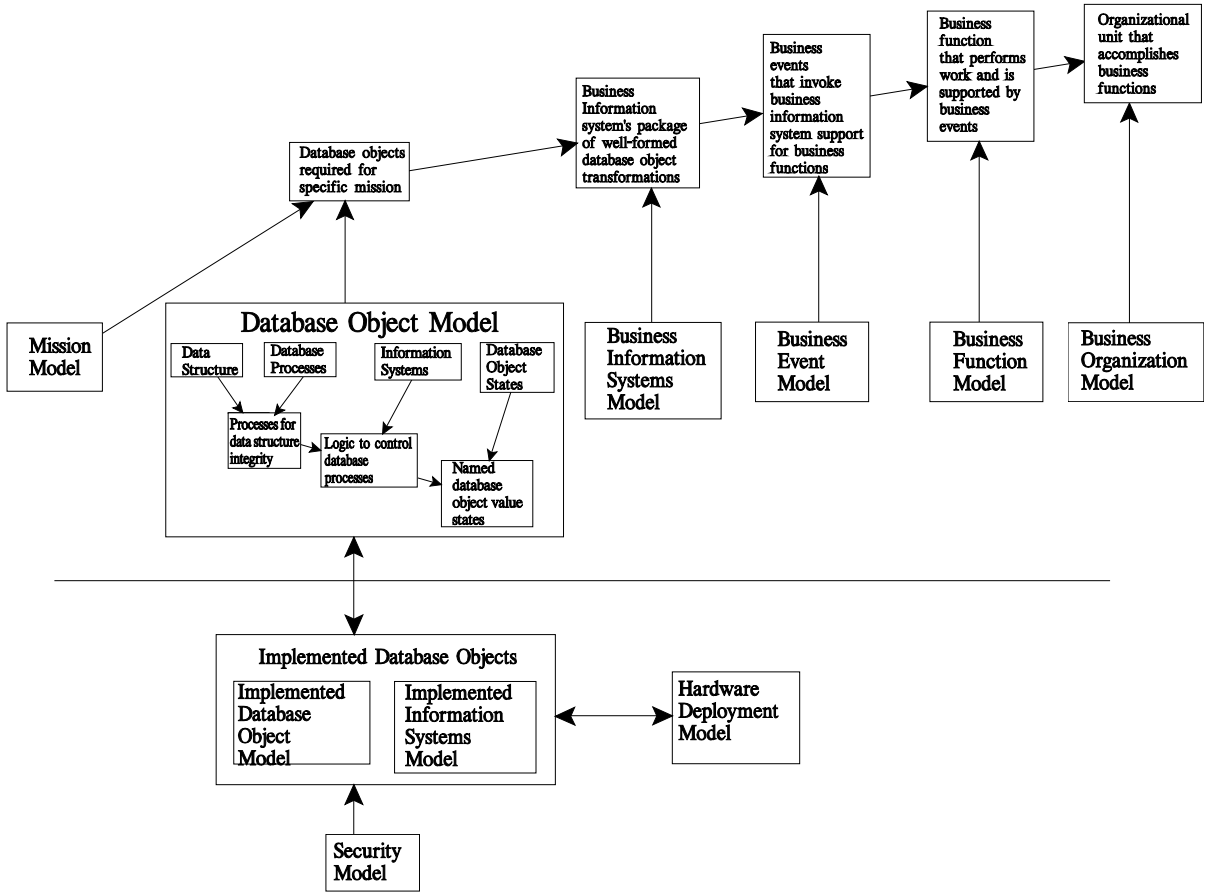


Figure 9.1 Relationship among Whitemarsh Knowledge Worker Columns and the implied Whitemarsh Meta Models.

9.1 Developing Missions

Business missions represent the ultimate purposes of the enterprise. The mission model wich represents the enterprise's mission is hierarchical. The components of the mission model consist of missions, database object models (one for each coherent mission), high level function models, and business terms. The mission model for the enterprise should be accomplishable in a matter of months. Once done, the *What*, *Why*, *Where*, and *How* is known. Once the information systems plan is set down the *When* will also be known.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Enterprise Database Metabase Models		Preliminary Analysis	Conceptual Specification	Binding	Implementation	Conversion & Deployment	Production & Administration
Mission Model		✔					✔
Database Object Components	Data Object Data Structures	✔	✔				✔
	Database Object Processes		✔				✔
	Database Object Information Systems		✔				✔
	Database Object States		✔				✔
Data Models	Semantic Hierarchies		✔				
	Data Elements		✔				✔
	Specified Data Model		✔				✔
	Implemented Data Model			✔			✔
	Operational Data Model				✔		✔
	View Data Model				✔		✔
Business Function Model		✔	✔				✔
Organization Model						✔	✔
Business Information Systems Model				✔	✔		
Security Model			✔		✔		
Deployed Hardware and Network Models			✔	✔	✔	✔	✔

Figure 9.2 Relationship between Whitemarsh Metabase Models and Whitemarsh Database Project Methodology

Mission is not the same as function. Mission is what you are trying to accomplish. Function is how you to accomplish it. Function based information systems commonly fail because as the way to do work changes so to must the information system. Each technology change requires either staying behind, or massive system changes. Both alternatives are unacceptable. Mission based systems are designed independent of technology and independent of day-to-day functions. Thus, mission based systems are not subject to the functionally based

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

system vagaries. Mission models, once correctly done, last 20 years or more while function based systems are seldom finished before becoming out of date for not mirroring the then current way a business operates.

Organizations typically have two sets of interconnected missions: programmatic and infrastructure. Programmatic missions embrace various enterprise business programs, around which information systems have been build. Examples from a state's public safety mission would include systems that build roads, perform emergency medicine, maintain vehicles registrations, and inspections. Infrastructure missions are those that the organization requires to efficiently operate in support of its programmatic missions. Examples include personnel and payroll systems, accounts receivable and payable systems, human resource systems, etc. While it might be perceived that infrastructure and programmatic systems are independent from each other, they are not. They commonly use personnel, funds, facilities, computers, networks, and the like. Systems that bring about infrastructure efficiencies may well free up resources to enhance programmatic efforts. Not only must the missions of each be known, but the interconnection of the missions must be known as required resources are in common.

The mission model is done for the enterprise as a whole. It sets the tone and temper for all the other models. It is essential for the Information Systems Plan. Once the mission model and Information Systems Plan is accomplished, the organization has the Information Systems Battle Plan. What is done when and why. The business reasons that justify a certain sequence are set down. The cost justification studies are completed. And, when (not if) technology changes, the Information Systems Plan, a live model, reacts and dynamically reconfigures the *makes business sense* sequence of information systems accomplishment.

The work breakdown structure to accomplish the mission model is:

1.03 Create mission description model
1.03.01 Create business mission submodel
1.03.01.01 Succinctly create business mission in business policy terms
1.03.01.01.01 Name mission
1.03.01.01.02 Describe mission in business policy terms
1.03.01.01.03 Utilize prototypical source data and reports to assist in definition
1.03.01.01.04 Remove any organizational references from missions
1.03.01.01.05 Remove any techniques or details that indicate how the mission is accomplished
1.03.01.01.06 Review missions to ensure that only the end-result *what* is described, and that both *who* and *how* has been removed
1.03.01.01.07 Review mission description with key users and revise until complete and acceptable
1.03.01.01.08 Enter mission description into repository

1.03.01.02 Partition business mission into sufficient stand alone subordinate missions
1.03.01.02.01 Identify stand alone subordinate missions
1.03.01.02.02 Describe subordinate missions in business terms
1.03.01.02.03 Use prototypical data and reports to assist in definition
1.03.01.02.04 Review subordinate mission with key users and revise until complete and acceptable
1.03.01.02.05 Repartition subordinate mission descriptions into lower level subordinate mission descriptions and repeat description until no longer necessary
1.03.01.02.06 Enter subordinate missions into repository

1.03.01.03 Create mission description diagram

1.03.02 Create database domain submodel

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1.03.02.01 Create database domains
1.03.02.01.01 Succinctly describe database domain from subordinate mission descriptions in business policy terms
1.03.02.01.01 Name database domain
1.03.02.01.02 Describe database domain in business policy terms
1.03.02.01.03 Utilize prototypical source data and reports to assist in definition
1.03.02.01.04 Validate database domain sufficiency by balancing the implied major database outputs against the major database contents
1.03.02.01.05 Review database domain with key users and revise until both non-redundant, complete, and acceptable
1.03.02.01.06 Relate database domains to missions and enter into repository

1.03.02.02 Partition domains into sufficient stand alone subordinate database domains
1.03.02.02.01 Identify stand alone subordinate database subdomains
1.03.02.02.02 Describe subordinate database domains in business terms
1.03.02.02.03 Utilize prototypical data and reports to assist in definition
1.03.02.02.04 Review database subordinate database subdomains with key users and revise until complete and acceptable
1.03.02.02.05 Repartition subordinate database subdomains into lower level subordinate database subdomains and repeat description until no longer necessary
1.03.02.02.06 Enter subordinate database domains into repository

1.03.02.03 Create initial database domain diagram for each database subdomain
1.03.02.03.01 Identify and define entities in non-redundant fashion
1.03.02.03.02 Identify and define obvious relationships among entities
1.03.02.03.03 Construct database domain diagrams
1.03.02.03.04 Review diagrams, entities, and relationships with users to assess relevance & revise as necessary

1.03.02.04 Merge subordinate database domain diagrams
1.03.02.04.01 Select most complex subordinate database domain diagram
1.03.02.04.02 Attempt merger of other database domain diagrams
1.03.02.04.03 Adjust subordinate database domain entities & relationships until merger is possible
1.03.02.04.04 Review combined database domain diagram for common understanding & agreement with key users
1.03.02.04.05 Finalize database domain diagram

1.03.03 Create database object submodel
1.03.03.01 Select entity from database domain diagram as a potential database object
1.03.03.02 Create name for potential database object
1.03.03.03 Create definition for potential database object
1.03.03.04 Determine if multiple property classes exist for potential database object
1.03.03.05 Record as database objects those potential database objects that have multiple property classes
1.03.03.06 Record as property classes those potential database objects that do not contain multiple property classes
1.03.03.07 Record as data elements those potential database objects that are not even property class; rather they describe single business fact
1.03.03.08 Review discovered database objects, property classes and data elements with users to ensure proper business names and business definitions
1.03.03.09 Provide an example for each database object, property class, and data element
1.03.03.10 Enter database objects, property classes and data elements into repository
1.03.03.11 Present defined database objects and review as appropriate
1.03.03.12 Create database object diagram

1.03.03.13 Create relationships between database objects and database domains and then enter into repository

1.03.04 Create high-level function submodel
1.03.04.01 Succinctly create business function in business policy terms

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1.03.04.01.01 Name business function
1.03.04.01.02 Describe business function in business policy terms indicating appropriate business activities
1.03.04.01.03 Utilize prototypical source data and reports to assist in definition
1.03.04.01.04 Remove any organizational style indications from business functions
1.03.04.01.05 Identify organizations that perform same high-level description of business function
1.03.04.01.06 Remove any detailed techniques that indicate how the business function is accomplished
1.03.04.01.07 Enter discovered business functions into repository
1.03.04.01.08 Enter discovered organizations into repository
1.03.04.01.09 Interrelate organizations and business functions, and enter interrelationships into repository
1.03.04.01.10 Interrelate business functions with missions and database objects, and enter into repository
1.03.04.01.11 Identify frequency of accomplishment by organization of business function
1.03.04.01.12 Review business functions to ensure that how the business function is kept at a high-level
1.03.04.01.13 Review business function description with key users and revise until complete and acceptable

1.03.04.02 Partition business function into sufficient stand alone subordinate business functions
1.03.04.02.01 Identify stand alone subordinate business functions
1.03.04.02.02 Describe subordinate business functions in business terms
1.03.04.02.03 Utilize prototypical data and reports to assist in definition
1.03.04.02.04 Review subordinate business function with key users and revise until complete and acceptable
1.03.04.02.05 Repartition subordinate business function descriptions into lower level subordinate business function descriptions and repeat description until no longer high-level

1.03.04.03 Enter subordinate business functions into repository

1.03.04.04 Create high-level business function diagram
1.03.04.04.01 Identify high-level business functions & define its purpose, scope, etc., in terms of business activities
1.03.04.04.02 Identify the business environment & activities in which business function occurs
1.03.04.04.03 Identify organizational responsibility for business function
1.03.04.04.04 Identify frequency of business function
1.03.04.04.05 Provide an example of each business function flow diagram
1.03.04.04.06 If business function contains business functions then repeat only a few levels
1.03.04.04.07 Record business event data into repository

1.03.05 Create business and technical terms glossary, and organize in learning order

1.03.06 Create mission model report
1.03.06.01 Prepare mission scope and narrative
1.03.06.02 Create plan & format for presenting mission descriptions, database domains, database objects, high level functions and all graphics
1.03.06.03 Prepare report for mission description model

1.03.07 Create preliminary analysis presentation
1.03.07.01 Review & revise preliminary analysis phase
1.03.07.02 Construct features, advantages and benefits (FAB) of database system
1.03.07.03 Utilize FABs to create use scenarios
1.03.07.04 Graphically identify entities/processes that participate in scenario
1.03.07.05 Create narrative for use scenario
1.03.07.06 Identify key individual for use scenario
1.03.07.07 Review use scenario & narrative with key individual and modify as appropriate
1.03.07.08 Create A/V materials for use scenario presentation
1.03.07.09 Review and finalize A/V materials prior to review

1.03.08 Conduct phase review

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

1.03.08.01 Present mission description model
1.03.08.02 Receive and respond to comments
1.03.08.03 Modify model components as necessary
1.03.08.04 Seek concurrence with phase

9.2 Specifying Database Objects

Database objects represent the specification of the enterprise's policies and procedures. Database objects are specified through the next four models. An example of an database object for the public safety environment is vehicle. An database object within the vehicle class is, for example, the vehicle Ford Aero Star, VIN # 123456784.3... A database object is defined through:

- The set of data structures that map onto all the different value sets for real world database object such as an auto accident, vehicle and emergency medicine incident.

- The set of database object processes that enforce the integrity of data structure columns, referential integrity and actions among rows, the proper computer-based rules governing row insertion, modification, and deletion. For example, the proper and complete storage of auto accident application data for the building of auto accident.

- The set of database object information system specifications that achieve specific value state changes in the database objects. Each database object state change must reflect the accomplishment of a business policy.

- The set of business recognizable named database object states such as auto accident initiation, involved vehicle entry, involved person entry, and auto accident DUI involvement.

Collectively, these four components must be built to have complete database objects. Once built, the business knowledge, embodied in these technology independent database object specifications are reviewed by the business experts prior to any consideration of technology. A badly specified set of database objects or information system even if accomplished through the best of technology still results in a bad system.

Designing and building database objects is new only for data processing professional. Businesses have been operating through database objects since business began. To wit, every business form had its directions for completion on the reverse. Included in the directions were codes, conditions, if-then-else logic and requirements for minimum essential information. Database objects are new for data processing because they are very complex and require sophisticated DBMS facilities that have only now started to exist. The required DBMS facilities include:

- triggers
- assertions
- table and column check clauses
- stored procedures

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- nested transactions
- multiple table updatable views
- savepoints and two phase commit

Because of ANSI SQL, database objects are now possible. Database object analysis and design is essential for quality client/server system development. That is because a database object, which is the expression of coherent policies and procedures, must never be split across servers. Some aspect of a database object may be distributed, but there is only one definitive occurrence of the database object. The rest are just copies. The benefits derived from designing and building database objects include:

- Systems that appear more business like to perceive and understand.
- Systems that are designed and built by business experts.
- Systems that can be reviewed and repaired by business experts.
- Systems that can be built-up from wholly contained database objects.
- Systems that are easier to maintain because they are self-contained and are not intertwined with other systems' components.
- A coherent strategy exists for distributing both data and processes in the client/server environment.

These benefits translate to better, higher quality systems, faster because business experts are involved earlier and remain longer. Because of this, system designs start on track, remain on track, and because of tools (CASE, repository and generators), system deliveries are timely.

The four models that collectively define an database object include: data structure, database object processes, database object information systems, and database object states.

9..2.1 Developing Database Object Data Structures

Data structure, the first component of the database object, is the data requirements expression from the mission. Proof that the mission is accomplished is embodied in the data that is captured and stored. Policy is executed data. Policies that exist in the organization are those that operate within the purview of the missions. A policy is not valid if it operates outside the organization's mission. Database designs then are clear, organized expressions of policy within missions.

Database designs exist as data structures that abound in rules. While each rule is an expression of a requirement, it is expressible as a test case. Test cases start at the column, then the table, then inter-table, then inter-row, then as assertions, triggers and constraints. These *future test cases* are numbered, identified and stored in the repository. Each rule is individually selectable. The applicability of the rule is known in every program in which it is used. Business rules are know separately from the business information systems that implement them. As the

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

business changes the business information systems too must change. With CASE, repositories and code-generators, changes are possible in days and weeks that formerly took months and even years.

The work breakdown structure to develop database object data structures is:

2.02.04.07 Create database object data structure component

2.02.04.07.01 Allocate data elements to database objects
2.02.04.07.01.01 Select database object for data element allocation
2.02.04.07.01.02 Review database object's property classes for suggestions of data elements
2.02.04.07.01.03 Select data elements from repository and assign to database objects as candidate data elements for database object data structures
2.02.04.07.01.04 Identify candidate data elements as single valued or multi valued
2.02.04.07.01.05 Evaluate property classes from database objects to determine suggested data element complete usage
2.02.04.07.01.06 Search for additional data elements if property classes are not exhausted
2.02.04.07.01.07 Modify repository with additionally discovered data elements
2.02.04.07.01.08 Evaluate repository to determine complete usage of all contained data elements
2.02.04.07.01.09 Incorporate additional data elements into repository if they are justified
2.02.04.07.01.10 Adjust property classes section of database object definition for addition of newly discovered data elements
2.02.04.07.01.11 Assign newly discovered data elements to an database object
2.02.04.07.01.12 Create localized candidate data element names where appropriate

2.02.04.07.02 Create normalized data structure for each database object
2.02.04.07.02.01 Examine candidate data elements for similar purpose
2.02.04.07.02.02 Assign candidate data elements to candidate database object tables of obvious name within database object
2.02.04.07.02.03 Examine candidate database object tables and split them into owner and member database object tables as required by single or multi-set status of subcollections of candidate data elements
2.02.04.07.02.04 Identify the set of data elements that make up a primary key for each database object table
2.02.04.07.02.05 Examine candidate data structures for commonly employed candidate data elements
2.02.04.07.02.06 Create a supertype data structure of the commonly employed candidate data elements
2.02.04.07.02.07 Examine candidate data structures for candidate data elements that are used for only subsets of data structure instances
2.02.04.07.02.08 Create one or more subtype data structures for the candidate data elements that are used for only subsets of data structure instances
2.02.04.07.02.09 Finalize data structure descriptions
2.02.04.07.02.10 Provide an example of the kind of data structure that would be typified by the data structure name
2.02.04.07.02.11 Create instances diagrams for each database object as a vehicle to illustrate data

2.02.04.07.03 Verify that database object data structures are in third normal form
2.02.04.07.03.01 Identify data structure primary key
2.02.04.07.03.02 Check for multi-set data elements, put into new or lower data structure
2.02.04.07.03.03 Check for partial primary-key dependencies, put into new data structure s
2.02.04.07.03.04 Check for nonprimary key data element dependencies into new data structures
2.02.04.07.03.05 Remove data elements dependent on data elements from other data structures, and note the required relationship for later inclusion
2.02.04.07.03.06 Reevaluate & correct stated name, function, etc. of database object as it relates to data structures, if necessary

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.02.04.07.03.07 Collect data elements that were discarded for allocation to other data structures of database object or to other database object's data structures
2.02.04.07.03.08 Revise data structure description
2.02.04.07.03.09 Reassemble data structures
2.02.04.07.03.10 Create data structure graphic for each database object

2.02.04.07.04 Create global database object data structure graphic

2.02.04.07.05 Modify data structures to allow for historical data
2.02.04.07.05.01 Determine periodicity of data change
2.02.04.07.05.02 Determine whether history is required
2.02.04.07.05.03 Set forth policy if history is kept as data changes, or at specific intervals
2.02.04.07.05.04 Specifically enumerate the intervals
2.02.04.07.05.05 Specify rules for historical data retention if they change, i.e., daily for a month, then monthly for a year, then quarterly, etc.
2.02.04.07.05.06 Specify algorithms for computing historical summaries, i.e., average, high, low, etc.
2.02.04.07.05.07 Adjust segment descriptions and structures to account for historical data
2.02.04.07.05.08 Revise data structure graphics

2.02.04.07.06 Update database object definition
2.02.04.07.06.01 Check database object for proper name & purpose
2.02.04.07.06.02 Revise example of the database object as necessary
2.02.04.07.06.03 Review and revise repository as appropriate
2.02.04.07.06.04 Review and revise database object diagram as appropriate

2.02.04.07.07 Verify completeness of database object graphic
2.02.04.07.07.01 Review complete data structure graphic against database objects to verify correctness
2.02.04.07.07.02 Create global graphic of all data structures within database objects
2.02.04.07.07.03 Review global data structure graphic against database domain diagram to verify correctness

9.2.2 Specifying Database Object Processes

Database object processes are the second component of the database object. While data structures represent executed policy, database object processes represent the corresponding procedures. Both policies and procedures, are required to define database objects. Quality business information systems are the policy based manipulations of database objects. Since data represents policy, and processes represents procedures, then valid representations of policy and procedure combinations represent the valid states of database objects. Database objects transform from state to state. Through the use of database objects, business information systems are ensured to move business policy from valid state to another valid state.

It is essential to use database objects as the boundaries of client/server database and business information system implementations. Whole database objects should be implemented never partial ones. For example, if there is distributed client/server system for human resources, then only whole employee database objects are stored. It may be built-up on the client, but until an employee database object reaches a valid state, it should not be stored on the server.

The work breakdown structure to build create database object processes is:

2.02.04.08 Specify database object database object process component

2.02.04.08.01 Identify data integrity requirements

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.02.04.08.01.01 Specify as data integrity rules those inferred though relationship descriptions contained within and between database objects
2.02.04.08.01.02 Specify as data integrity rules intra-data structure data element dependencies
2.02.04.08.01.03 Specify as data integrity rules the self contained row conditions that determine row acceptance into the database
2.02.04.08.01.04 Specify as data integrity rules the inter data structure instance conditions that determine whether a dependent data structure is accepted into the database
2.02.04.08.01.05 Specify as data integrity rules inter-data structure element value conditional dependencies
2.02.04.08.01.06 Specify as data integrity rules inter-data structure derived data dependencies

2.02.04.08.02 Identify data integrity requirements for data transfer and data conversion
2.02.04.08.02.01 Specify as data integrity rules changes in valid and invalid values for edit data structures
2.02.04.08.02.02 Specify as data integrity rules changes in range values for edit data structures
2.02.04.08.02.03 Specify as data integrity rules changes in data structure look-ups for edit data structures
2.02.04.08.02.04 Specify as data integrity rules all changes from derived external view data elements to atomic database view data elements
2.02.04.08.02.05 Specify as data integrity rules all changes from atomic external view data elements to derived database view data elements
2.02.04.08.02.06 Provide example of the data integrity rule

2.02.04.08.03 Specify all identified data integrity rules with a name & definition, function/role, owner & member data structure, and relational algebra
2.02.04.08.04 Provide an example of the data integrity rule
2.02.04.08.05 Record DIR in repository

2.02.04.08.06. Specify all database object processes
2.02.04.08.06.01 Create an add, delete, and modify database object process for each database object table
2.02.04.08.06.02 Create selection logic to obtain each database object table instance
2.02.04.08.06.03 Create appropriate messages for different types of database object process actions
2.02.04.08.06.04 Provide an example of database object process
2.02.04.08.06.05 Record database object process data into repository

2.02.04.08.07. Specify intra database object referential action database object processes
2.02.04.08.07.01 Identify data structure within a database object
2.02.04.08.07.02 Create appropriate referential action add, delete, and modify database object process for all data structures within the database object
2.02.04.08.07.03 Create selection logic to obtain each database object table instance
2.02.04.08.07.04 Create appropriate messages for different types of database object process actions
2.02.04.08.07.05 Provide an example of referential action database object process
2.02.04.08.07.06 Record referential action database object process data into repository

2.02.04.08.08 Specify inter database object referential action database object processes
2.02.04.08.08.01 Create data structures that intersect database objects
2.02.04.08.08.02 Create appropriate referential action add, delete, and modify database object process for all database object intersection data structures
2.02.04.08.08.03 Create selection logic to obtain each database object intersection data structure instance
2.02.04.08.08.04 Create appropriate messages for different types of database object process actions
2.02.04.08.08.05 Provide an example of referential action database object process
2.02.04.08.08.06 Record referential action database object process data into repository

2.02.04.08.09 Specify all DIR-based database object processes
2.02.04.08.09.01 Identify appropriate data structure for DIR-based database object processes
2.02.04.08.09.02 Create appropriate add, delete, and modify database object processes to accomplish DIR
2.02.04.08.09.03 Create selection logic to obtain each database object table instance

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.02.04.08.09.04 Create appropriate messages for different types of database object process actions
2.02.04.08.09.05 Provide an example of DIR-based database object process
2.02.04.08.09.06 Record DIR-based database object process data into repository

2.02.04.08.10 Create specified views for database object processes
2.02.04.08.10.01 Name and describe the specified view
2.02.04.08.10.02 Identify appropriate specified view data elements from database object data structure
2.02.04.08.10.03 Identify appropriate specified view navigation logic to navigate database objects
2.02.04.08.10.04 Identify appropriate specified view select clauses to obtain database object database object table instances
2.02.04.08.10.05 Create appropriate messages for different types of view actions
2.02.04.08.10.06 Record specified view definition into repository

2.02.04.08.11 Create additional required database object process model data integrity rules
2.02.04.08.11.01 Specify any additional data integrity rules that are inferred though relationship descriptions contained in data structures
2.02.04.08.11.02 Specify any additional data integrity rules that are intra-data structure data element dependencies
2.02.04.08.11.03 Specify any additional data integrity rules that are self contained record conditions that determine database object table instance acceptance into the database
2.02.04.08.11.04 Specify any additional data integrity rules that are inter database object table instance conditions that determine whether a dependent database object table instance is accepted into the database
2.02.04.08.11.05 Specify any additional data integrity rules that are inter-data structure element value conditional dependencies
2.02.04.08.11.06 Specify any additional data integrity rules that are inter-data structure derived data dependencies
2.02.04.08.11.07 Specify all identified data integrity rules with a name & definition, function/role, owner & member data structure , and relational algebra
2.02.04.08.11.08 Provide an example of the data integrity rule
2.02.04.08.11.09 Record DIR in repository
2.02.04.08.11.10 Identify appropriate data structure for DIR-based primitive database object processes
2.02.04.08.11.11 Create appropriate add, delete, and modify primitive database object processes to accomplish DIR
2.02.04.08.11.12 Create selection logic to obtain each database object table instance
2.02.04.08.11.13 Create appropriate messages for different types of database object process actions
2.02.04.08.11.14 Provide an example of DIR-based primitive database object process
2.02.04.08.11.15 Record DIR-based primitive database object process data into repository

2.02.04.08.12 Validate database object process model
2.02.04.08.12.01 Evaluate each database object process for clear, unambiguous meaning
2.02.04.08.12.02 Evaluate specified views for appropriate construction
2.02.04.08.12.03 Evaluate database object processes for conflicting updates
2.02.04.08.12.04 Evaluate all data integrity rules by information system process to ensure that no rules are in conflict.

2.02.04.08.13 Create database object process model deliverable
2.02.04.08.13.01 Create database object process model narrative
2.02.04.08.13.02 Finalize all database object process model information system process flow diagrams

2.02.04.08.14 Conduct subphase review

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

9.2.3 Specifying Database Object Information Systems

Database object information systems, the third component of the database object, are the specification of a well bounded set of database object transformations. For example, a database object information system may be entering a police report for an accident. First, the accident has to be in the database and then all the parts of the policeman's report can be stored.

Each database object information system must be semantically complete. An action to a database object cannot be split among peer database object information systems because that would risk a broken database. A database is broken when it is semantically inconsistent. A database object information system may however, contain multiple database object information systems. In such a case, a database object, containing database objects may have its total activity accomplished by the contained database object information systems. Because of this definition of database object information system, some whole parts can be deployed on the client and other whole parts on the server. Because each part represents the valid transformation of database objects, there cannot be an incomplete or broken partitioning.

Database object information systems are designed to accomplish database object transformations that serve the accomplishment of some aspect of the enterprise's mission.

The work breakdown structure for creating database object information system specifications is:

2.02.04.09 Create database object information system component

2.02.04.09.01 Create detailed information system processes and diagrams
2.02.04.09.01.01 Identify each information system process & define its purpose, scope, etc., in terms of database object transformation
2.02.04.09.01.02 Record each information system process descriptively and place appropriately into a information system process diagram
2.02.04.09.01.03 Identify specific subordinate information system processes that are included in information system process
2.02.04.09.01.04 Specify rules that governs information system process success within information system process flow diagram
2.02.04.09.01.05 Specify required reactions as a consequence of information system process failure
2.02.04.09.01.06 Provide an example of each information system process flow diagram
2.02.04.09.01.07 If information system process contains subordinate information system processes then repeat until unnecessary

2.02.04.09.02 Estimate implementation phase efforts for each database object information

2.02.04.09.03 Create documentation for each database object information system
2.02.04.09.03.01 Create database object information system narrative
2.02.04.09.03.02 Create summary of design requirements

2.02.04.09.04 Conduct subphase review

9.2.4 Specifying Database Object States

Database object states, the final component of the database object, is the named set of states for database objects. The database object states are determined through an analysis of the life cycles

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

of the essential business resources. The business resources are the database objects uncovered through the analysis of the business mission and the database domains that are derived from the missions.

Each business resource is studied and its life cycle is determined from its original null state to a series of valued states returning to the null state. As each resource takes on is business policy form, the database object states that are to reflect the policy's accomplishment are identified. The database object data structures are examined to determine what instance manipulation is required. After the database object state changes are identified, the specifications of the change are recorded in terms of the database object information system actions required to effect the database object value state changes through the execution of the database object processes.

The work breakdown structure employed to identify database object states is:

2.02.04.11 Specify database object state component
2.02.04.11.01 Identify the database life resources
2.02.04.11.01.01 For each database life cycle resource
2.02.04.11.01.02 Identify the life cycle nodes
2.02.04.11.01.03 For each life cycle node
2.02.04.11.01.03.01 Identify the database objects that are modified by the life cycle node
2.02.04.11.01.03.02 For each identified database object
2.02.04.11.01.03.02.01 Identify the database object tables that must be added/deleted/or modified to accomplish the business policy of the node
2.02.04.11.01.03.02.02 Name the end result of the database object transformation as the database object state
2.02.04.11.01.03.02.03 Identify the database object information systems that must be executed to accomplish the database object transformation
2.02.04.11.01.03.02.03 Store the database object state and the algebra for accomplishing the database object state transformation through database object information systems into the metabase
2.02.04.11.02 Identify the database life resources

9.3 Specifying Business Information Systems

The business information systems are evidenced through five subordinate models. These subordinate models are:

- Information system
- Report
- File
- Screen
- Internal process

These subordinate models do not interact directly with the database through views. Rather, they interact with database objects which in turn causes database changes. Each business information system receives data from operating system files or as direct inputs from interactive users. The business information system passes the data onto the database objects through database object invocations. The database objects process the data according to their predefined database object processes that transform the database objects from one value state to another.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The work breakdown structure that employed to specify the business information systems is:

2.03.01 Specify business information systems model
2.03.01.01 Identify business information systems appropriate for subordinate mission description

2.03.01.02 Create online data update system specification

2.03.01.02.01 Create detailed information system processes and diagrams for online data update system
2.03.01.02.01.01 Identify each user-required information system process & define its purpose, scope, etc., in terms of business activities
2.03.01.02.01.02 Record each information system process descriptively and place appropriately into a information system process diagram
2.03.01.02.01.03 Review the business environment & activities in which information system process occurs to ensure the information system process's sufficiency
2.03.01.02.01.04 Identify and record organizational responsibility for information system process
2.03.01.02.01.05 Identify frequency of information system process
2.03.01.02.01.06 Identify specific subordinate information system processes that are included in information system process
2.03.01.02.01.07 Specify rules that governs information system process success within information system process flow diagram
2.03.01.02.01.08 Specify required reactions as a consequence of information system process failure
2.03.01.02.01.09 Provide an example of each information system process flow diagram
2.03.01.02.01.10 If information system process contains subordinate information system processes then repeat until unnecessary

2.03.01.02.02 Create screen specification
2.03.01.02.02.01 Name, describe, and sketch the screen, and identify the information system process that invokes the screen
2.03.01.02.02.02 Review the screen for simplicity and single-purpose. If complex, then divide the screen into subordinate single-purpose screens.
2.03.01.02.02.03 Identify appropriate screen elements
2.03.01.02.02.04 Use where ever possible the list of data elements, but if necessary add new data element and its new definition to the list of data elements
2.03.01.02.02.05 Specify the help message that is to be present for each screen element
2.03.01.02.02.06 Specify the valid and invalid value DIRs for each screen element
2.03.01.02.02.07 Specify the legal and illegal range value DIRs for each screen element
2.03.01.02.02.08 Specify the data structure look up DIRs for each screen element
2.03.01.02.02.09 Specify which screen elements must be valued for legal entry
2.03.01.02.02.10 Identify which screen elements should be protected from update
2.03.01.02.02.11 Identify any internal processes that must be accomplished for each screen, and/or screen element
2.03.01.02.02.12 Specify the required function keys and the actions that are to be taken for each
2.03.01.02.02.13 Identify appropriate specified navigation logic among screens if different than that implied by the information system process diagram
2.03.01.02.02.14 Connect all input screens to appropriate external views with information system control logic units
2.03.01.02.02.15 Connect all input screen specified views to appropriate information system control logic units

2.03.01.02.03 Design file operations
2.03.01.02.03.01 Identify file that must be read or written
2.03.01.02.03.01 Identify appropriate external view with information system control logic unit
2.03.01.02.03.03 Identify any internal processes that must be accomplished for each file cell or file record

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.01.02.04 Create internal processes
2.03.01.02.04.01 Identify and name internal process
2.03.01.02.04.02 Describe internal process
2.03.01.02.04.03 Create external view with information system control logic unit, or with screen, or with screen element, or with file operation, or with file cell
2.03.01.02.04.04 Create data transformation algorithm
2.03.01.02.04.05 Record internal process into repository along with relationship metadata

2.03.01.02.05 Estimate implementation phase efforts for each online data update subsystem

2.03.01.02.06 Review and revise human support subsystems affected by new on-line data update subsystems
2.03.01.02.06.01 Identify each human support subsystem
2.03.01.02.06.02 Create process diagram to determine all manual processes
2.03.01.02.06.03 Optimize manual processes to take advantage of new tools
2.03.01.02.06.04 Identify new training requirements
2.03.01.02.06.05 Identify any necessary job description revisions
2.03.01.02.06.06 Formulate plan to implement changes to human support subsystems

2.03.01.02.07 Create documentation for each online data update subsystem
2.03.01.02.07.01 Create online data update subsystem narrative
2.03.01.02.07.02 Create summary of design requirements

2.03.01.02.08 Conduct subphase review

2.03.01.03 Create batch data update system specification

2.03.01.03.01 Create detailed information system processes and diagrams for batch data update system
2.03.01.03.01.01 Identify each user-required information system process & define its purpose, scope, etc., in terms of business activities
2.03.01.03.01.02 Record each information system process descriptively and place appropriately into a information system process diagram
2.03.01.03.01.03 Review the business environment & activities in which information system process occurs to ensure the information system process's sufficiency
2.03.01.03.01.04 Identify and record organizational responsibility for information system process
2.03.01.03.01.05 Identify frequency of information system process
2.03.01.03.01.06 Identify specific subordinate information system processes that are included in information system process
2.03.01.03.01.07 Specify rules that governs information system process success within information system process flow diagram
2.03.01.03.01.08 Specify required reactions as a consequence of information system process failure
2.03.01.03.01.09 Provide an example of each information system process flow diagram
2.03.01.03.01.10 If information system process contains subordinate information system processes then repeat until unnecessary
2.03.01.03.01.11 Connect all information system processes to appropriate specified views
2.03.01.03.01.12 Connect all information system processes specified views to appropriate database object processes

2.03.01.03.02 Design file operations
2.03.01.03.02.01 Identify file that must be read or written
2.03.01.03.02.02 Identify appropriate external view with information system control logic unit
2.03.01.03.02.03 Identify any internal processes that must be accomplished for each file cell or file record

2.03.01.03.03 Create internal processes
2.03.01.03.03.01 Identify and name internal process

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.01.03.03.02 Describe internal process
2.03.01.03.03.03 Create external view with information system control logic unit, or with screen, or with screen element, or with file operation, or with file cell
2.03.01.03.03.04 Create data transformation algorithm
2.03.01.03.03.05 Record internal process into repository along with relationship metadata

2.03.01.03.04 Estimate implementation phase efforts for each batch data update subsystem

2.03.01.03.05 Review and revise human support subsystems affected by new batch data update subsystems
2.03.01.03.05.01 Identify each human support subsystem
2.03.01.03.05.02 Create process diagram to determine all manual processes
2.03.01.03.05.03 Optimize manual processes to take advantage of new tools
2.03.01.03.05.04 Identify new training requirements
2.03.01.03.05.05 Identify any necessary job description revisions
2.03.01.03.05.06 Formulate plan to implement changes to human support subsystems

2.03.01.03.06 Create documentation for each batch data update subsystem
2.03.01.03.06.01 Create batch data update subsystem narrative
2.03.01.03.06.02 Create summary of batch data update subsystem design requirements

2.03.01.03.07 Conduct subphase review

2.03.01.04 Create data loading subsystem specification
2.03.01.04.01 Identify, name, and then create information system process diagram for each data load subsystem that illustrates any load sequence requirements

2.03.01.04.02 Evaluate initial data source quality for each data load subsystem
2.03.01.04.02.01 Identify data source for each element
2.03.01.04.02.02 Identify source data format
2.03.01.04.02.03 Identify difference between database & source data formats, editing, etc.
2.03.01.04.02.04 Sample source data to evaluate quality
2.03.01.04.02.05 Evaluate differences among multiple sources of same data
2.03.01.04.02.06 Devise strategy to resolve data source differences
2.03.01.04.02.07 Estimate resources required to bring source data into conformance with database requirements
2.03.01.04.02.08 Prepare source data evaluation report

2.03.01.04.03 Review and revise human support subsystems affected by new data load subsystems
2.03.01.04.03.01 Identify each human support subsystem
2.03.01.04.03.02 Create information system process diagram to determine all data processes
2.03.01.04.03.03 Optimize processes to take advantage of new tools
2.03.01.04.03.05 Identify new training requirements
2.03.01.04.03.05 Identify any necessary job description revisions
2.03.01.04.03.06 Formulate plan to implement changes to human support subsystems

2.03.01.04.04 Create specifications for each load subsystem
2.03.01.04.04.01 Identify size & number of records for each data structure
2.03.01.04.04.02 Create a logical record-type loading sequence chart
2.03.01.04.04.03 Specify editing & validation for each data structure
2.03.01.04.04.04 Specify inter-record editing & validation
2.03.01.04.04.05 Determine error alternatives for all editing and validation
2.03.01.04.04.06 Connect all data loading file definitions to appropriate specified views
2.03.01.04.04.07 Connect all data loading file specified views to appropriate database object processes
2.03.01.04.04.08 Estimate implementation phase efforts for each data loading subsystem
2.03.01.04.04.09 Determine required backup during database loading

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.01.04.05 Create screens
2.03.01.04.05.01 Name, describe, and sketch the screen, and identify the information system process that invokes the screen
2.03.01.04.05.02 Review the screen for simplicity and single-purpose. If complex, then divide the screen into subordinate single-purpose screens.
2.03.01.04.05.03 Identify appropriate screen elements
2.03.01.04.05.04 Use where ever possible the list of data elements, but if necessary add new data element and its new definition to the list of data elements
2.03.01.04.05.05 Specify the help message that is to be present for each screen element
2.03.01.04.05.06 Specify the valid and invalid value DIRs for each screen element
2.03.01.04.05.07 Specify the legal and illegal range value DIRs for each screen element
2.03.01.04.05.08 Specify the data structure look up DIRs for each screen element
2.03.01.04.05.09 Specify which screen elements must be valued for legal entry
2.03.01.04.05.10 Identify which screen elements should be protected from update
2.03.01.04.05.11 Identify any internal processes that must be accomplished for each screen, and/or screen element
2.03.01.04.05.12 Specify the required function keys and the actions that are to be taken for each
2.03.01.04.05.13 Identify appropriate specified navigation logic among screens if different than that implied by the information system process diagram
2.03.01.04.05.14 Connect all input screens to appropriate external views with information system control logic units
2.03.01.04.05.15 Connect all input screen specified views to appropriate information system control logic units

2.03.01.04.06 Design file operations
2.03.01.04.06.01 Identify file that must be read or written
2.03.01.04.06.02 Identify appropriate external view with information system control logic unit
2.03.01.04.06.03 Identify any internal processes that must be accomplished for each file cell or file record

2.03.01.04.07 Create internal processes
2.03.01.04.07.01 Identify and name internal process
2.03.01.04.07.02 Describe internal process
2.03.01.04.07.03 Create external view with information system control logic unit, or with screen, or with screen element, or with file operation, or with file cell
2.03.01.04.07.04 Create data transformation algorithm
2.03.01.04.07.05 Record internal process into repository along with relationship metadata

2.03.01.04.08 Estimate implementation phase efforts for overall load subsystem
2.03.01.04.09 Prepare documentation for each load subsystem
2.03.01.04.10 Conduct subphase review

2.03.01.05 Create specification for each data transfer and data conversion subsystem
2.03.01.05.01 Review and revise information system process diagram for each data conversion subsystem that illustrates any data conversion sequence requirements
2.03.01.05.02 Optimize processes to take advantage of new tools
2.03.01.05.03 Identify size and number of records for each data structure
2.03.01.05.04 Create a logical record-type data conversion sequence chart
2.03.01.05.05 Specify editing & validation for each data structure
2.03.01.05.06 Specify data integrity rules
2.03.01.05.07 Determine error alternatives for all editing and validation
2.03.01.05.08 Estimate implementation phase efforts for overall data conversion subsystem
2.03.01.05.09 Determine required back-up during database data conversion
2.03.01.05.10 Prepare documentation for each data conversion subsystem
2.03.01.05.11 Conduct subphase review

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.01.06 Create enterprise-wide data collection system specification

2.03.01.06.01 Identify corporate forms for which data collection systems are to be created

2.03.01.06.02 Analyze each corporate form and design specific information system to collect data
2.03.01.06.02.01 Identify each user-required information system process & define its purpose, scope, etc., in terms of business activities
2.03.01.06.02.02 Record each information system process descriptively and place appropriately into a information system process diagram
2.03.01.06.02.03 Review the business environment & activities in which information system process occurs to ensure the information system process's sufficiency
2.03.01.06.02.04 Identify and record organizational responsibility for information system process
2.03.01.06.02.05 Identify frequency of information system process
2.03.01.06.02.06 Identify specific subordinate information system processes that are included in information system process
2.03.01.06.02.07 Specify rules that governs information system process success within information system process flow diagram
2.03.01.06.02.08 Specify required reactions as a consequence of information system process failure
2.03.01.06.02.09 Provide an example of each information system process flow diagram
2.03.01.06.02.10 If information system process contains subordinate information system processes then repeat until unnecessary

2.03.01.06.03 Create data collection screen specification
2.03.01.06.03.01 Name, describe, and sketch the screen, and identify the information system process that invokes the screen
2.03.01.06.03.02 Review the screen for simplicity and single-purpose. If complex, then divide the screen into subordinate single-purpose screens.
2.03.01.06.03.03 Identify appropriate screen elements
2.03.01.06.03.04 Use where ever possible the list of data elements, but if necessary add new data element and its new definition to the list of data elements
2.03.01.06.03.05 Specify the help message that is to be present for each screen element
2.03.01.06.03.06 Specify the valid and invalid value DIRs for each screen element
2.03.01.06.03.07 Specify the legal and illegal range value DIRs for each screen element
2.03.01.06.03.08 Specify the data structure look up DIRs for each screen element
2.03.01.06.03.09 Specify which screen elements must be valued for legal entry
2.03.01.06.03.10 Identify which screen elements should be protected from update
2.03.01.06.03.11 Identify any internal processes that must be accomplished for each screen, and/or screen element
2.03.01.06.03.12 Specify the required function keys and the actions that are to be taken for each
2.03.01.06.03.13 Identify appropriate specified navigation logic among screens if different than that implied by the information system process diagram
2.03.01.06.03.14 Connect all input screens to appropriate external views with information system control logic units
2.03.01.06.03.15 Connect all input screen specified views to appropriate information system control logic units

2.03.01.06.04 Create internal processes
2.03.01.06.04.01 Identify and name internal process
2.03.01.06.04.02 Describe internal process
2.03.01.06.04.03 Create external view with information system control logic unit, or with screen, or with screen element, or with file operation, or with file cell
2.03.01.06.04.04 Create data transformation algorithm
2.03.01.06.04.05 Record internal process into repository along with relationship metadata

2.03.01.06.05 Create normalized data structure for each data collection form

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.01.06.05.01 Examine candidate data elements for similar purpose
2.03.01.06.05.02 Assign candidate data elements to candidate data structures of obvious name within data collection form
2.03.01.06.05.03 Examine candidate data structures and split into owner and member data structures as required by single or multi-set status of subcollections of candidate data elements
2.03.01.06.05.04 Identify the set of data elements that make up a primary key for each data structure
2.03.01.06.05.05 Examine candidate data structures for commonly employed candidate data elements
2.03.01.06.05.06 Create a supertype data structure of the commonly employed candidate data elements
2.03.01.06.05.07 Examine candidate data structures for candidate data elements that are used for only subsets of data structure instances
2.03.01.06.05.08 Create one or more subtype data structures for the candidate data elements that are used for only subsets of data structure instances
2.03.01.06.05.09 Finalize data structure descriptions
2.03.01.06.05.10 Provide an example of the kind of data structure that would be typified by the data structure name

2.03.01.06.06 Estimate implementation phase efforts for each data collection system

2.03.01.06.07 Review and revise human support subsystems affected by new data collection systems
2.03.01.06.07.01 Identify each human support subsystem
2.03.01.06.07.02 Create process diagram to determine all manual processes
2.03.01.06.07.03 Optimize manual processes to take advantage of new data collection system
2.03.01.06.07.04 Identify new training requirements
2.03.01.06.07.05 Identify any necessary job description revisions
2.03.01.06.07.06 Formulate plan to implement changes to human support subsystems

2.03.01.06.08 Create documentation for each data collection subsystem
2.03.01.06.08.01 Create online data collection subsystem narrative
2.03.01.06.08.02 Create summary of design requirements

2.03.01.06.09 Conduct subphase review

9.4 Specifying Business Events

Business events activate one or more information systems in support of accomplishing the named business event. For example, budget modification might be accomplished through multiple information systems.

The set of all involved information systems might include some that perform on-line update, some that accomplish detailed validation that all new budgets are within certain ceilings, and then the execution of a whole new set of reports. While each of these business information systems is complete in their own right, the collection is also complete, and thus should be seen from a more generalized viewpoint: Perform Budget Maintenance. The business event as that catalytic role. Here again, this relationship strategy between business event and business information system enables the *defined once used many times principle.*

All database object metadata is stored in the repository. All parts of a database object is either able to be declared directly through ANSI standard SQL or are able to be code generated. All database objects information systems are maintained through CASE, repository, and code generators. Experience has shown this approach to be about 50% faster with constant staff, or to take the same time with 50% less staff.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

The work breakdown structure that supports the creation of the business event model is:

2.03.02 Create business event model
2.03.02.01 Identify and describe business events that are information system triggers
2.03.02.02 Identify the business information system that are to be performed to adequately respond to the business event
2.03.02.03 Provide an example of each business event
2.03.02.04 Record business event and interconnect with business information systems into repository

9.5 Specifying Business Functions

Business functions as represented by their business function models mirror the day-to-day activities for various organizational units within the enterprise. Because of the Whitemarsh Methodology, different organizational units won't have to hide the fact from management that they have different functional models for the same set of activities. Each organizational unit can now have their own style of accomplishing a business function, just so long as when their style requires use of database objects that are transformed in a standard way.

If there has been anything proven over the past 30 years of data processing methodology technique evolution, is that it is impossible to straight-jacketing the way people do work.

There have been countless--unnecessary--resystemizations in the name of changed functionality. If the business has truly changed, that is, radically changed its mission, then there is a real need to change its database objects. If however the only change has been technology binding, that is, changing from main-frame to client/server, or changing DBMS (IDMS to Oracle), then the only models that have changed the set of implemented models (implemented data, implemented information systems, security and hardware/networks). Once the key models (mission, and database objects (database object data structure, database object processes, database object information system, and database object states) are complete and stored in the repository then they can be used over and over again whenever a new binding or changes in binding is required.

Because the Whitemarsh methodology breaks the very strong linkage between specification (mission through organization) and implementation (the implemented data, information systems, and hardware models), database objects can be specified once and implemented many times for different hardware, DBMS, and operating systems. Each such rebinding saves all the funds formerly spent or the respecification effort. In short, the specified once, bound many times cannot help but save considerable funds and time. With either constant time or even decreasing funds, the Whitemarsh methodology enables the enterprise to specify and implement many more information systems.

The work breakdown structure that specifies the business function model is:

2.03.03 Create detailed business function model
2.03.03.01 Select high-level business function for detailing
2.03.03.02 Select organization(s) relevant for business function detailing
2.03.03.03 Identify appropriate next levels of business functions that are to be accomplished by the specific organization(s)
2.03.03.04 Identify business forms involved with business function
2.03.03.05 Identify document involved with business function

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

2.03.03.06 Record business form and/or document details and inter-relationships with business function into repository
2.03.03.07 If business function contains business functions then repeat until business event boundary
2.03.03.08 Identify the business events required to trigger information systems
2.03.03.09 Provide an example of each business function
2.03.03.10 Record business function hierarchy and interconnect with business events into repository

9.6 Specifying Business Organizations

Business organizations are a corollary to business functions. As organizations are configured differently, their business functions may be the same or be configured differently. The purpose of specifying business organizations is to enable the overall organization, the enterprise, to see that even though there is the appearance of difference, all policy (recorded data) is actually be accomplished in a standard, uniform, and consistent manner.

2.03.04 Create Organization Model
2.03.04.01 Identify organizations involved in various missions
2.03.04.02 Identify subordinate organizations as necessary and enter all organization units into repository
2.03.04.03 Identify the business functions that are performed by the organization
2.03.04.04 Identify the role the organization plays with respect to the business function
2.03.04.05 Record business function role relationship between organizational units and business functions
2.03.04.06 Identify the quantity of staff hours and quantity of staff required to perform the identified function

9.7 Methodology Support Summary

The work breakdown structures cited above only address the specification component of mission through organization. Clearly there is database object implementation and long term database object evolution and maintenance. The key to both however is quality database object specification. Database object implementation is not addressed in this text however as it depends too much on the specific implementation environments. Thus, an Oracle's RDBMS implementation through Oracle/CASE and all its code generators while similar in concept to Informix's or Sybase's is different in sufficient degree to warrant chapters for each.

When enterprise database is fully underway there will be a number of projects concurrently underway. Figure 9.3 illustrates a continuous-flow environment. Because of the continuous flow nature, these project development environments that can handle:

- Multiple, concurrent, but differently scheduled projects against the same operations database or warehouse database
- Single-database projects that affect multiple operations and warehouse databases
- Projects that develop completely new MIS capabilities, that can assess required changes to existing MIS capabilities, and that can accommodate a variety of systems generation alternatives (COTS, package, and custom programming)

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

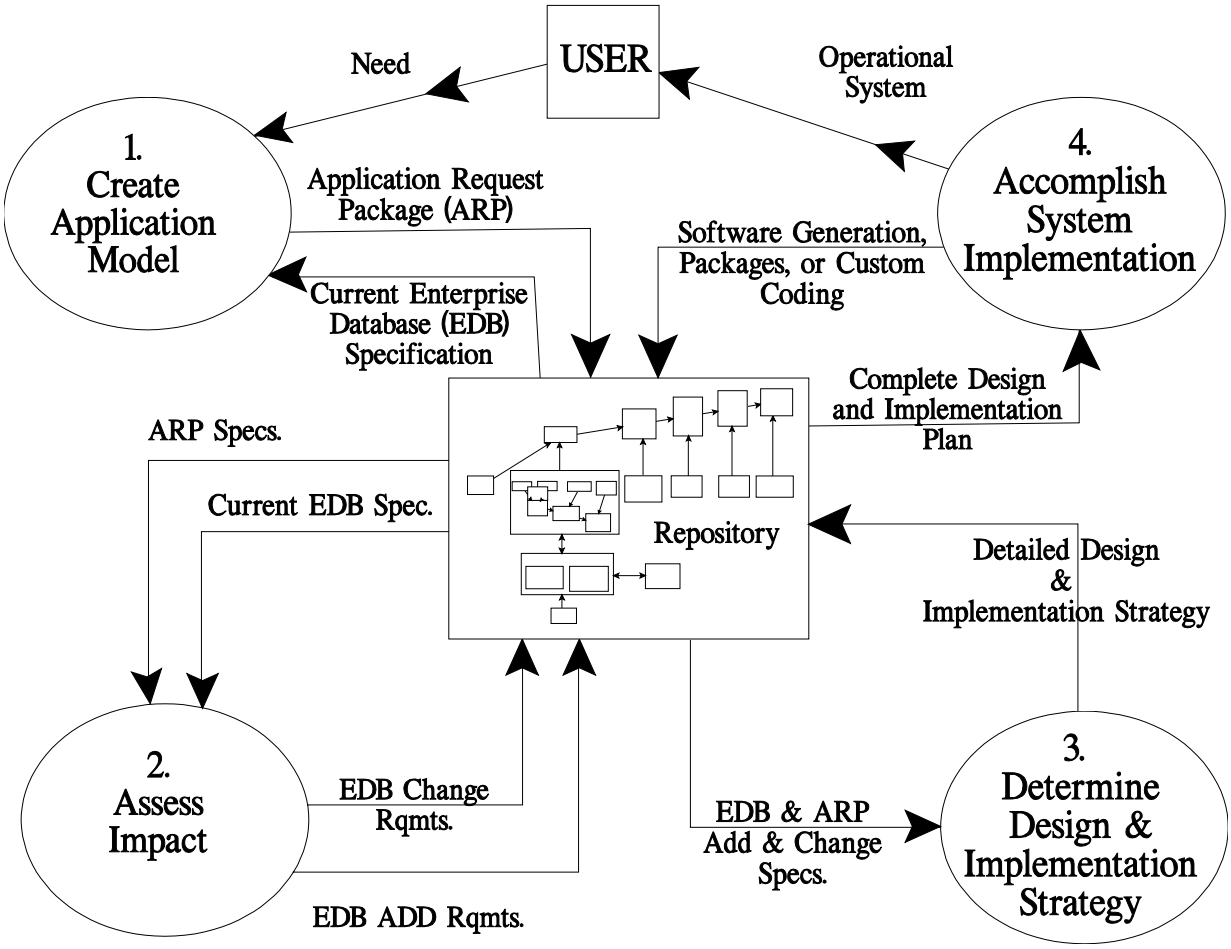


Figure 9.3. Continuous flow development model.

Figure 9.3 presents a diagram containing the four major sets of activities for enterprise database's continuous flow environment. The user/client is represented at the top in the small rectangular box. Each of the ellipses represents an activity list to accomplish a specific need. The four basic needs are essentially:

- Need Identification
- Need Assessment
- Design
- Deployment

The box in the center is the repository. Specification and impact analysis are represented through the left two processes. Implementation design and accomplishment are represented by the right two processes.

In the first process, Create Application Model, the user provides requirements to an application's analyst, who interrogates the existing repository about the existing enterprise database specification and implementation models. If the user's request can be handled by

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

existing enterprise features, it is. Otherwise, the application's analyst formulates the requirement into an application request package (ARP). The ARP is represented as a need and includes a series of brief requirement statements in terms of additions to or modifications of an existing enterprise database. That is, additions or modifications of database objects, and additions and/or modifications of business information systems.

Because this is a continuous-flow model, an ARP may be handled by either existing enterprise database capability, or those now in implementation (process 4), or, in time, through process 3 (design) or process 2 (impact assessment).

At predetermined intervals, for example, monthly or quarterly, all the ARPs are bought into the second process, Assess Impact. During this process, all the ARPs are examined in unison against the then current enterprise's database specifications. Because of the continuous-flow nature of this process, the enterprise database specification may have changed during the time when a set of ARPs is being batched. When the enterprise database add or change requirements are formulated, they will reflect the state of enterprise database at the time it is to be changed. The form of the enterprise database add and/or change requirements package is similar to that of the ARP. When the impact of a change is assessed, there may also be changes to existing facilities. These changes must be accomplished as well. Thus, when the final enterprise database add or change package is developed, it contains these additional change requirements.

The third process, Determine Design & Implementation Strategy, determines the actual detailed specifications of the computing systems environment changes. The detailed design of the change is in the form of specific changes to existing database objects, views, computer systems, documentation, procedures, and the like.

In addition to the detailed design of the enterprise database changes, an implementation plan for a project has to be created. Included is the appropriate PERT chart, WBS, resource loadings, and then a Gantt schedule. Once this project proposal is received and approved, the complete set of changes becomes a new enterprise database version that moves on to the implementation process.

The final process, Accomplish System Implementation, performs all the normal implementation activities. At the end of test and integration, the new enterprise database release is accomplished.

Through this continuous-flow process, several unique features are present:

- All four processes are concurrently executing.
- Changes to enterprise database occur either quarterly or less often and in a very controlled manner.
- The enterprise database repository always contains all the enterprise database specifications, current or planned. Simply put, if some semantic is not in the enterprise database, it is not corporate information systems policy.
- All changes are planned, schedule, measured, and subject to accounting.
- All documentation of all types is generated from the enterprise database repository.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Supporting the continuous flow life cycle must be sophisticated project management that provides for an understanding of all projects in planning and execution, and the storage of lessons learned from projects that have completed.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

10

METADATA REPOSITORY SUPPORT

The metadata repository system support for database objects necessarily includes:

- Metadata model
- Processes

10.1 Database Object Metadata Repository Meta Model

The metadata model for database objects consists of 16 different tables. Of these 16, six tables are read only tables as they are created in other metabase modules. The tables that are created within database objects are:

- Database Domain and Database Object
- Database Object
- Database Object Information System
- Database Object Information Systems and Database Object Process
- Database Object Process
- Database Object Process Column
- Database Object State
- Database Object State and Database Object Information System
- Database Object Table
- Membership Rational

The tables that are created in other metabase modules are:

- Column (created in the implemented data model module)
- Data Element (created in the data element module)
- Database (created in the implemented data model module)
- Database Domain (created in the mission, organization, function module)
- Schema (created in the implemented data model module)
- Table (created in the implemented data model module)

Of these six, five are created within the data modeling metabase module. The metadata model design for database objects is presented in Figure 10.1.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

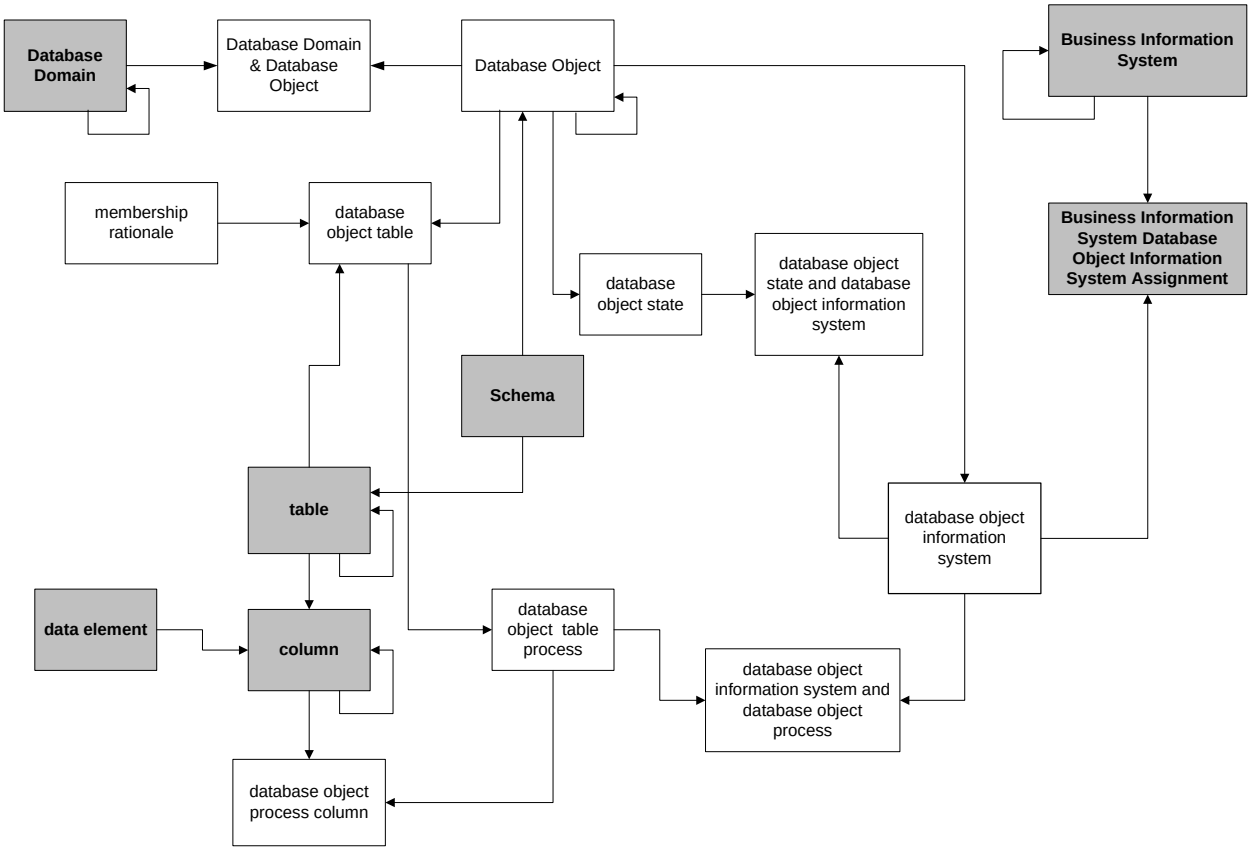


Figure 10.1. Metadata model for database objects.

Explicit in this database are the following structures:

- A database object is a collection of one or more tables. The rows from these tables are transformed from one valid state to another via database object table processes and database object information systems. Database objects are related to one or more database domains.

- A Database Domain and Database Object is the association of one or more database objects with one or more database domains.

- A database object is a collection of one or more tables. The rows from these tables are transformed from one valid state to another via database object table processes and database object information systems. Database objects are related to one or more database domains.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- A Database Object Information System is a collection of processes defined within the domain of the DBMS usually as a stored procedure that transforms one or more rows of a database object from one valid state to another. A database object information system accomplishes one or more database object table processes.

- A Database Object Information Systems and Database Object Process is the association of one or more Database Object Information Systems and Database Object Processes. These are set into a sequence for proper accomplishment.

- A Database Object Table Process is a process such as insert, change, or delete that occurs against one row of a single table within a database object. A table owns (and is thus acted upon by) one or more database object table processes. A database object table process may be invoked by one or more database object information systems.

- A Database Object Table Process Column is an association of a specific database object table process and a specific column of a table.

- A Database Object State is a well defined value state of a database object. States occur in a particular sequence, typically from the null state through a set of value states and returning to a null state. A database object state is accomplished through one or more database object information systems.

- A Database Object State and Database Object Information System is the association of one or more Database Object States and Database Object Information Systems. The association exists within a sequence so that the state is properly achieved.

- A Database Object Table is an association of a table with a database object. Membership rationale classifies the reason why a table belongs to the database object.

- Membership Rational is a classification of the reasons why a table belongs to a database object as evidenced through the database object table.

- Schema and database object is the association between a schema and a database object. This means that a database object may reside in more than one schema.

These statements must be carefully studied as they represent the paradigm of database objects. Because a table can be in multiple database objects, this paradigm support multiple inheritance. In this special case, there are only two roles for tables. The first is a proper table, such as job-skills of an employee. The other is a referenced table, that is, the job-skills-types. Reference tables are merely abstraction representations, and could be completely encapsulated. They are not so as to achieve data standardization across multiple database objects within the enterprise.

The only reference data table is membership rationale. The purpose of membership rationale is to identify the reason why a table is included in a particular database object. Since there is only one reference data table it has been included in the database object Data Structure menu item.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Once the application is installed it is ready to use. Just invoke the software from the metabase program. The application is a traditional windows application. Metabase reports are accomplished through any ODBC class report writer such as Crystal Reports.

10.2 Database Object Metadata Repository Process Model

The table that follows contains the overall metadata repository system process model for database objects.

```
-- Database Objects

    -- DataStructure
        -- Database Object Tables
        -- Database Object Table Membership Rationales

    -- InformationSystem
        -- Database Object Information System

    -- State
        -- Database Object State

    -- Processes
        -- Database Object Tables Processes

    -- Assignments
        -- Assign Database Domains to Database Objects
        -- Assign Database Objects to Tables
        -- Assign Database Object Table Processes to Columns
        -- Assign Database Object State to Database Object Information System
        -- Assign Database Object Information System to DBO Table Processes

    -- ReEngineering
        -- Reassign Database Object to Database Domain
        -- Reassign Database Object to Schema
        -- Reassign Database Object Tables
        -- Reassign Database Object Table Process Columns
        -- Reassign Database Object State to Database Object Information Systems
        -- Reassign Database Object Processes to Database Object Information Systems
```

Descriptions of each of these processes are contained in the Whitemarsh Metabase Database Object User Guide that is available from the Whitemarsh website.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

11
SUMMARY

11.1 What Database Objects Are and Are Not

As stated at the outset, there are generally considered to be three classes of objects: display objects, wholly contained process objects, and business objects. The closest to database objects are business objects which encompass business components like an insurance policy that performs in a certain manner.

All three object classes have their proponents and detractors. What all three object classes have in common is that they are first and foremost self contained software in the form of an executable or embedded process that behaves according to certain fixed rules. *Database objects is none of them.*

As usual, however, data-biased persons see the world inside-out from those who are process-biased. To the process-extremist, an object is a process with encased data structure components. To the data-fanatic, an object is a data structure with contained process components. Can they be reconciled? Aren't they just inside-out's of each other? No. It's the same kind of difference as exists between data flow diagrams and entity-relationship diagrams. They are NOT inside-out's of each other. They have a fundamentally different orientation.

The data driven and process driven orientations don't have to come together. Many articles about object-oriented analysis and design highlight techniques that were very popular during the days of highly structured Fortran programming; where complex systems were put together out of commonly used and reused pieces (and subroutines) of code from libraries. While those code objects are important and useful, they are not database! Nor, are they what database was EVER intended to be.

Database objects share some common names and some common definitions as the other three object types. Because of common names and some common definitions, the reader may initially be confused as to the nature of database objects. Database objects however, are unique to database. They are identified, designed, implemented, operated through, evolved, or maintained through just one type of data processing facility, a database management system (DBMS).

Database objects are not new. They existed in certain DBMS types in the late 1960s. Relational DBMS, however, stopped the march to database objects. It was not until the ANSI SQL:1999 data model moved away from the relational model and not until a whole programming language was incorporated into ANSI SQL:1999 that the march towards database objects was restarted. Even if the delay had not happened, computers, networks, languages and operating systems were just not sophisticated enough to make database objects successful.

But, what forms the basis of an database object? Simply, it is a business' policies and procedures. While policies can exist without procedures, the converse is not true. This ontological priority dictates that procedure is dependent on policy. Not only do they go together like hand and glove, the glove (procedure) serves no useful purpose without the hand (policy).

A database object is a person, place, or thing that has internal consistency, and is transformed from one state to another through well defined rules. The minimum value states are

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

null and valued. The internal behavior of a database object as it transforms from one state to another is immaterial to its user. Database objects conform to the requirements of business rather than the converse.

Policies and procedures, that is, database objects, bring order, consistency, and predictability. The larger the enterprise, the greater the dependence on policies and procedures. Data is the evidence of policy execution. An employee's record is proof that policies have been carried out. Procedures are the techniques, methods, or processes by which policies are carried out. If an enterprise has the policy is to be profitable, then its balance statement, produced by processing all the general and subsidiary journals is the measure of adherence to the policy. If policy is met, the enterprise must be profitable.

Policies and their associated data, address the well bounded infrastructure and programmatic areas. The data takes on common every day names such as employees, facility, mortgage, insurance policy, and student. The data representing these common names are complex, that is, whole multiple-level structures.

The procedures are named, and their data actions are associated with specific subsets of the named data structure. The names of the procedure sets represent data structure transformations from one recognizable state to another. Each state represents a determined value set within the business. Procedure examples include: establishing an employee requisition, accomplishing employee hiring, and performing employee assignment.

Traditional computer programming languages do not contain sufficient data modeling, access and processing facilities to fully handle business objects. Further, computer programming languages encapsulate data to such an extent that the defined and contained business objects are truly captive of the programming language within which they are defined, captured, stored, and manipulated. Simply, traditional programming languages are “data poor.”

Business objects, as described by the Object Management Group, are software products that live within traditional computing environments. OMG's business objects are defined, employed, and are manipulated by traditional object oriented languages such as C++, Smalltalk, 4GLs such as Sybase's Power Builder, Oracle Forms, Clarion for Windows, or Microsoft's Visual Basic, and are stored either in traditional file structures, relational database management systems, or hybrid object-relational database management systems. Because all these language environments operate differently on different computing environments, OMG's Business Objects cannot satisfy the demands of business environments for database objects which:

- Are easy to specify, implement, use and maintain
- Operate on world-wide, heterogeneous hardware and operating system environments, and
- Behave consistently regardless of their host computing hardware environment

Database was always intended to be representations of coherent structures of data that matched well defined areas of policy and the supporting behavior to match well known business event states. Consequently,

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

A database object is an instance of a data structure that proceeds through predefined states according to embedded process transformations.

Database objects, regardless of persistence and regardless of whether single or multi-table contain the same four-part composition:

- Database Object Structure: the set of data structures that map onto the different value sets for real world database objects such as an auto accident, vehicle and emergency medicine incident.

- Database Object Process: the set of database object processes that enforce the integrity of data structure fields, references between database objects and actions among contained database object tables, the proper computer-based rules governing database object table insertion, modification, and deletion. For example, the proper and complete storage of an auto accident.

- Database Object Information System: the set of specifications that control, sequence, and iterate the execution of various database object processes that cause changes in database object states to achieve specific value-based states in conformance to the requirements of business policies. For example, the reception and database posting of data from business information system activities (screens, data edits, storage, interim reports, etc.) that accomplish entry of the auto accident information.

- Database Object State: The value states of a database object that represent the after-state of the successful accomplishment of one or more recognizable business events. Examples of business events are auto accident initiation, involved vehicle entry, involved person entry, and auto accident DUI (driving under the influence of alcohol/drugs) involvement. Database object state changes are initiated through named business events that are contained in business functions. The business function, auto accident investigation includes the business event, auto-accident-incident initiation, which in turn causes the incident initiation database object information system to execute, which in turn causes several database object processes to cause the auto accident incident to be materialized in the database.

The database object benefits include:

- Whole containment within SQL DBMS
- Access to both type and instance components
- Complete expression through syntax
- Import and export through ISO/ANSI standard SQL facilities
- Ability to be distributed and consistent operations via all SQL compliant DBMSs
- Independence from presentation-layer and operating-system bindings

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

11.2 The Promise of Database

Database is the application of quality organization, planning, and management. Central to these organizational characteristics are carefully crafted policies and procedures. Designed well, business policies and procedures become database objects that can be deployed throughout the organization in a client/server fashion to maximize sharing and consistency while minimizing data hoarding and irregularity.

Database objects are the foundation stones for enterprise database. Organizations not pursuing their specification, implementation and evolution are condemned to complicated, redundant specifications, expensive implementation and difficult operation, evolution and maintenance.

Because database objects are wholly contained within SQL DBMS they can be centrally accessed and manipulated regardless of the end-user environment, that is, batch, on-line, stand-alone "fat" clients, thin clients, Internet, or traditional client/server.

SQL DBMS databases contain both type and instance data. Type-data is metadata. Instance data is traditional data. For an employee database, type data are the table, column, integrity definitions, stored procedures and the like. Instance data are the actual employee records.

Type data is critical for distributed data and process databases because it can be queried to then determine the exact semantics required for database access operations. For example, if there is a currency exchange data object on a server, a query can determine the arguments and data types of the required inputs. Transactions can then be formulated and successfully accomplished.

Database objects are completely expressible as syntax enabling SQL compliant DBMSs to receive new database object syntax for inclusion, requirements to delete database objects from the type and instance data, and command and data strings that cause updates to standard reference data that can act as dynamic integrity constraints.

A fundamental requirement of any compliant SQL DBMS is that it be able to import and export both type and instance data through standard SQL commands. Applications are able to export or import data through standard character set strings. When a new computer site is established, a central server can be activated to down load command strings of syntax and standard reference data. Once down loaded and stored within the SQL:1999 DBMS, the database objects are immediately operational regardless of the DBMS brand, operating systems type, hardware vendor, or 3rd or 4th generation language tools that access and manipulate the newly installed database objects.

Because of ISO and ANSI standards, database objects operate consistently regardless of the SQL DBMS, operating system, and vagaries of the different presentation layer facilities. Enterprises are able to then have centralized semantics that control the fundamental operations of the business objects that are essential to world wide, heterogeneous computing environments.

Finally, database objects are independent from presentation-layer and operating-system bindings. This enables use of local language conventions within the confines of standard policy essentials. For example, regardless of their local abbreviations and local names, there are only two sexes. Additionally, local vendors may provide their own presentation layer facilities, report writers, formats, paper sizes, screen formats, and the like. Given database objects, these localized peculiarities can be accommodated. And, because the database object environment is DBMS based, additional and localized database objects can be easily created and deployed with

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

automatic integration into the standard database object environments essential for effective world-wide, heterogeneous environments.

11.3 Successful Computing Environments

Successful computing environments are those in which you can:

- Develop and maintain sophisticated and complex systems that access data and perform appropriate processes
- Accomplish information system development and maintenance *within* the life span of a modern business problem
- Develop a greatly increased quantity of information system solutions to business problems at a lower per unit cost
- Create applications that are both relevant to the individual and applicable to the enterprise as a whole

The evolution of computing environments can be seen through the evolution of: computer languages, information systems, hardware and operating systems, and the staff required to bring all the components together.

- The computer language evolution has brought us computer languages that are clearly more powerful to the extent that high level specifications can be created quickly out of reusable templates that, in turn, can cause the complete creation of whole business information systems. Because of these developments, there are a greater set of alternatives for defining and implementing IT systems that carry out enterprise policy in an integrated, non-conflicting, and non-redundant manner. There therefore needs to be one executing environment through which database objects are both specified and executed so that there is both consistency and truth to our data. Hence the need for DBMS-based database objects.

- The information systems evolution has taken us one full circle and another half circle from main-frame dominated to PCs on the desk top back to robust servers, and even main-frames that are executing may virtual servers. There is now a real push to completely separate transaction creation from transaction processing. This is the essence of service oriented architectures. Achieving success in this area however requires a much greater effort in defining transactions and also including in transactions all the necessary information for the service oriented environments to be able to find just the right transaction processors. Database objects are even more important that before because all the business rules must be removed from the transaction creation systems and placed in the transaction processing systems.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

- The hardware and operating systems evolution has taken us from a ratio where the hardware was many times the cost of the staff to where the staff is many times the cost of the hardware. The impact here is that there would naturally be an increase in the quantity of sources for transactions that are to be processed. Hence each such device has to have a smaller footprint in terms of both hardware and application software. Database objects content is therefore naturally increased and important.

- The people-ware evolution has taken us from where staff had many, many months to understand a business problem before attempting its automation to the point where solutions have to be learned, design, built, and deployed in no more than a few months. While this is caused mainly by the increased velocity of business changes, it is exacerbated by th high-level staff hour cost. The role of database objects is increased here because if database object classes are fully defined in CASE/Repositories that are in turn coupled with code generators then people-ware will be more efficiently and effectively employed.

- The outsourcing scourge or salvation issue is one that seems to have come full circle. From centralized staff to outsourced staff or contractors to then outsourced staff located at off-shore organizations. Many organizations are now finding that such arrangements are not as business wise as first thought because of all the time and cost overhead associated with management, review, and administration. For reasons similar to those previously provided, it's critical that enterprise policies and procedures are fully defined within database object classes and be stored and evolved in CASE/Repositories that are owned by the enterprise and not by our-source organizations.

- Finally, as the world becomes more "flat" in that everyone can be in contact with everyone else through the Internet, that the natural time-delays of data collection, storage and processing and then transmission to a "higher" level of database is becoming almost instantaneous. Thus, from a communications and interchange point of view, the world is slowly shrinking to just a "point." Database objects become critical here because their contained policy and procedure definition would be commonly and instantly employed by all the residents of our "flat earth."

The reason database objects are critical to client/server, heterogeneous, multiple-DBMS, world wide environments is because of all the different types of change agents that surround the database objects. If the critical semantics of a business is stored both in its data and its computer programs, AND if there are many different types of language (C, COBOL, 4GL, MS/Access, Sybase's Power Builder, etc.), then the probability of having consistent semantics across this programming language environment is ZERO. Either a business must decide on one and only one DBMS, one and only one operating system, and one and only one 4GL programming language, or it must move the semantics of its essential business policies and procedures inside the "firewall" of the DBMS. But since it ranges from impractical to impossible to demand that a business operate only one brand of DBMS, these critical database object must be moved inside the ANSI SQL language. It is only through the ANSI SQL language that business can ensure that database object semantics are protected.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

11.4 Summary

Modern enterprise-wide information systems can only succeed despite the crushing demands of more in less time and for less resources through the use of database objects which are:

- Data structures that define the enterprise policy.
- Database processes that define universally applicable procedures.
- Information systems that define the collection of valid database object transformations.
- Business events that define and represent the triggering of the valid state changes for each database objects.

When database objects are developed through CASE, stored in repositories and built through code generators real savings in both time and dollars are a certainty.

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

INDEX

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved

Copyright 2006, Whitemarsh Information Systems Corporation
Proprietary Data, All Rights Reserved