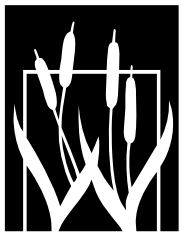


Whitemarsh
Information Systems Corporation

Database Objects Overview

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

1. Object Types

There are generally considered to be five different types of objects: display, process, language objects, database, and business. These five distinct types of objects are required by organizations as they employ modern, cost-effective computing technologies in world-wide heterogeneous environments.

Because current and accurate information is the essential competitive edge, database objects are a key type of object because they are easy to specify, cost effective to implement, and are highly portable among heterogenous computing environments.

All five object types have their proponents and detractors. All object types, except database objects are first and foremost self contained software that executes according to certain fixed rules. Database objects are different because they are instances of a data structure that proceeds through predefined states according to embedded process transformations.

While database objects share some common names and definitions with the other four object types, they are however unique to database. They are identified, designed, implemented, operated through, evolved, or maintained through just one type of data processing facility, an ANSI SQL database management system (DBMS). If the available ANSI/SQL DBMS is an SQL3 compliant then database object definition and use can be direct. Otherwise, database objects can only be achieved through its proprietary facilities.

Database objects are not new. They were started in certain DBMS types in the late 1960s. It was not until the ANSI SQL3 data model expanded beyond the relational model and not until a whole programming language was incorporated into ANSI SQL3 that database objects became possible.

2. Why Database is Essential to Business

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.

[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.

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. Database objects are not new, rather they are an optimization of true concepts from the past. They are now practical to specify, implement and maintain through CASE, code generators and repositories. Finally, database objects significantly optimize traditional systems analysis, design, implementation, testing, and maintenance activities.

The database object concepts were formulated almost 20 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 X3H2, the ANSI Database Languages committee 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, *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 on paper only until the ANSI database languages committee, X3H2, working since early 1993 on the specification of SQL, formulated the essential linguistic components of database objects. These exist under the name of SQL/3. This book presents database objects first as requirements, then as an example, then through an enumeration of the SQL/3 facilities that enable them.

Database objects, but not through SQL were validated almost 10 years ago through a large scale database project for the U.S. Army.

> An Army General in 1985 wanted PECO, a 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 described in this paper has been accomplished many times since 1985 with commercially available software and hardware.

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

3. The Business Case for Database Objects

Distributed, client/server data and processes 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 paper is all about the development of the symphonic scores through which organizations plan, conduct and modify their enterprise (establishment (static) and campaigns (dynamic)). In 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]

---

[2] In this paper, 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

(continued...)

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, 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

[3](...continued)

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.).

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.

4. The Data Processing Case for Database Objects

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 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, and finally client/server. 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.

In the book, *Enterprise Database in a Client/Server Environment*[4], Figure 1 was supplemented with Figure 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.

[4] 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.

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)

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. Data Semantics and Storage Control

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 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 2. Process Semantics and Execution Control

Since 1994, however, through Microsoft's ODBC[5] (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, TopSpeed 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.

Enterprise database environments that consisted of a relatively small set of well controlled main-frames through which business semantics[6] 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 and 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 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.

---

5 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.

6 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).

The database object development environment is accomplished through database object oriented systems analysis and design activities[7] 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 3, which is an amalgamation of Figures 1 and 2, results. An enterprise database environment in which semantic control is extended only to data (Figure 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 3 is not just an update to Figures 1 and 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 database object instances according to the particular style and computing environments of the individual world-wide business units.

5. 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 five fundamentally different classes of objects:

- Process
- Display
- Language
- Database(described above)
- Business

[7] See Chapter 11 of Database Objects, the Foundation Stones of Enterprise Database, Whitemarsh Press, 1997 (www.wiscorp.com).

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 3. Database Objects Semantics and Development Control

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, boxes, and windows on a computer screen. Language objects include mathematic operations such as SUMs, AVGs, SQRT or the trigonometric operation SINE or COSINE have been known the data processing community almost since its inception. Business objects are a conceptual component of a business or enterprise that exhibits certain behavior.

Because there are a number of different types of object classes, listening to persons making object presentations can often result in different definitions and examples with the consequence of confusion and mystery. The analogy is with the two words, work and record. Is the presenter using the words in their noun form or their verb form? Depending on which, the definitions and examples are different. Similarly with objects, it is necessary to understand each different type and to understand which type is being addressed by the speaker. The table over the next pages provides characteristics of the most common types of objects.

Three variations of the word object are essential to understanding the table: object class type, object class, and object. First, object. An object is something real such as a specific instance of a patient, or an automobile traversing the campus. An object class is an abstract representation/definition of all patients or all automobiles. Buick is an object class. A Buick with the VIN number 12320598203984038 is an object of the object class Buick automobile. A object class type is a category of object classes that are dissimilar with respect to their object class attributes. For example, a button on a computer screen doesn't have gas consumption statistics, weight, or is likely to rust. Similarly, an object class automobile is unlikely to have been programmed in the language "C," or exhibit "shadow" when selected.

In addition to these three terms, since the set of object classes under discussion are all related to just computing, three additional columns within the table are of interest: definition environment, execution environment, and portability considerations. The definition environment presents the characteristics/languages and/or computer support required to fully express the object's class. Once the object class is instantiated, the object's execution environment is essential to understand. That is, is it proprietary, commonly available, etc. Finally, when objects finally exist, are they portable from one execution environment to the next, or are they captive to the specific execution environment.

Object Class Type	**Business Definition**	**Example**	**Object Class Definition Environment**	**Object Class Execution or Object use Environment**	**Object Portability Characteristics**
Display object	A button/icon on a computer screen that when activated or pushed performs a predetermined function	Windows 98 Explorer Icon or the Windows 98 Start button	Windows operating system	Windows 98 operating system	Portable only when included in the Windows 98 operating system. There may be similar functions in other operating

Object Class Type	Business Definition	Example	Object Class Definition Environment	Object Class Execution or Object use Environment	Object Portability Characteristics
					systems, but the specific object is not portable across operating systems from different vendors
Process object	A named activity that when invoked accomplishes a certain result	Computation of quantity of beds available for patient intake	Defined through use of a computer language such as COBOL, Cognos' Powerhouse, or a DBMS such as Oracle	The object can exist and operate only within computing environments that accommodate its existence. For example, an object class programmed in a proprietary 4GL can only operate within computers that support it and possibly that are also executing a "run-time" module from the language vendor of the 4GL. Once installed, and only if the object's class conforms to an external specification for inter object access and data exchange can the object "interact" with other objects.	Portability is likely low between environments supporting different 4GLs or operating systems not accommodating the specific 4GL. The data representation component of the object is likely incomplete and not portable.
Language	A formal	The syntax	The definition	SQRT is	The actual

Object Class Type	Business Definition	Example	Object Class Definition Environment	Object Class Execution or Object use Environment	Object Portability Characteristics
object	expression of syntax within a computer language that when used accomplishes a certain result	SQRT, or the mathematical function cosign or tangent. Included are also language commands such as READ, or PRINT.	environment is the language one level lower or more detailed that the language in which the language object is used. For example, the SQRT function might be programmed in assembler and used within the language FORTRAN. Given that its meaning and results are well defined and external to the computing environment (they are!) Then whenever SQRT is invoked, the result is predictable.	executable within every language that offers its expression to the language's programmer. Languages that commonly offer SQRT are all 3GLs, most 4GLs, financial modelling languages, and the like.	assembler code representing the object class is not portable to different operating systems. However, the behavior of the invoked object is portable. Thus, when ever "49" in input, the result "7" must be the only allow result.
Database object	An organization of data structures, embedded processes, control logic, and named states that occur is a well defined order.	Patient and the defined states of patient such as identified, admitted, assessed, treated, discharged	The definition environment is the computer language SQL. Currently a significant amount of the SQL will be vendor proprietary. As SQL/3 becomes standardized more and more of a database	ISO/ANSI standard SQL based DBMSs are required to define and to execute database objects. Once installed, and only if the object's class conforms to an external specification for	Database object class definitions exist as syntax. Database objects exist as value-pairs of syntax and value (Name = Gorman). ASCII files of object class definitions and values are exchangable

Object Class Type	Business Definition	Example	Object Class Definition Environment	Object Class Execution or Object use Environment	Object Portability Characteristics
			object's syntax will become vendor independent.	inter object access and data exchange can the object "interact" with other objects.	between SQL based DBMSs. Additionally, one SQL based application environment can configure and submit an SQL compliant command to another SQL environment to interrogate both the other SQL environment's object class and its objects.
Business object	A conceptual component of a business or enterprise that exhibits certain behavior	Patient within Riverview	A business object class is expressed through traditional requirements analysis and design. Once accomplished, the object class may be syntactically expressed, for example, in any combination of 3GL, 4GL, and SQL DBMS. Once completely defined/expressed, the object class is transformed to an "executable" through traditional data processing	The object class exists as an installable component of a business application. Once installed, and only if the object's class conforms to an external specification for inter object access and data exchange can the object "interact" with other objects.	The objects require the exact operating system and likely DBMS environments within which their object classes were expressed. The actual objects are portable only to the extent that another object class of the same type contains data loading facilities and has been configured to behave in congruent ways.

Object Class Type	Business Definition	Example	Object Class Definition Environment	Object Class Execution or Object use Environment	Object Portability Characteristics
			methods. Once executable, it is made available to end users as a menu item on a screen, or a invocable system, subsystem, or function.		

Business objects are frequently described in the trade press and are described extensively in papers submitted by ANSI/X3H7 as well as the business object management special interest group (BOMSIG) of the Object Management Group (OMG). Because business objects and database objects can be confused, business objects are further described.

6. Business Objects

The Object Management Group (OMG) is its 1995 OMB Business Application Architecture White Paper, Draft 2, describes the business object as 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[8].

[8] 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.

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.

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 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

7. ANSI/X3H2, The Database Languages Committee

When ANSI database standards were started in 1978[9], 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. At an ANSI X3H2 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 X3H2 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 its computing languages clearly 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.

X3H2, 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. 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.

---

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

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/3 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

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/3 are database objects now possible. Database objects represent a technology independent, vendor, computing platform and operating system independent marriage between data and process semantics.

Since 1992, the ANSI and ISO database languages committees have been expanding SQL well beyond its relational strictures. SQL/3 is now powerful enough to handle business's need for objects, hereafter called database objects. SQL/3 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/3 also contains a full data manipulation language for stored procedures, and a full complement of data integrity rules, actions, and procedures. Finally, SQL/3 contains fully developed facilities for transaction management.

SQL/3 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 Top Speed's Clarion for Windows. It is through the vendor proprietary, hardware and operating system specific facilities that SQL/3 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.

8. ANSI X3H2 Database Objects

Database objects "live" entirely within the domain of the ANSI X3H2 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.

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/3, 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/3 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:

- Data 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 data structure segments, the proper computer-based rules governing data structure segment 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[10]. 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.

9. Object Oriented Environments

Knowledge worker environments must be object oriented. That is, they must work in whole activities rather than specialized subsets of activities. Real product workers work in highly specialized environments in order to produce real products such as aircraft, automobiles, and other manufactured products. Real product environments are not required to be object-oriented. Rather, they are just required to be exact and correct for every iteration of the product or subproduct that is manufactured. Knowledge worker environments must be object oriented because they involve whole collections of data and processes, and because the knowledge worker products are abstract and are highly variable either by design or necessity.

The three heralded characteristics of objects are encapsulation, inheritance, and polymorphism. The most critical aspect of an object oriented environment is encapsulation. Encapsulation means that the component (that is, its data and attendant processes) that is within the capsule is accessible only through the well defined rules imposed by the capsule. An EKG machine is an example of complete encapsulation. The machine contains all the materials and processes necessary to capture the EKG signals, record them on paper, and to react to

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

unacceptable signal changes. Encapsulation has been a characteristic of well defined, designed, implemented, and maintained data processing systems and environments since the early 1950s.

Encapsulation existed well before data processing. A business form that contains not only the blank fields for data entry and also all the sets of instructions, look-up tables, etc., is encapsulated. If the data entry person had to rely on external tables and/or external instructions and if the wrong set was employed then there would be two consequences:

- broken encapsulation
- bad data entry

From a formal viewpoint, 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 configured independent of the currency of the money represented by the premium given 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.

All successful computing environments must strive to be object-oriented because this lessens errors, increases the quality of designs, and lowers the costs of application and database development. The knowledge worker framework is depicted on the next page. This knowledge worker framework embraces all organizations, functions, business information systems, databases, interfaces between functions and business information systems, and Riverview's missions. Each component within the framework should be object-oriented. An object oriented component can contain references or make use of "outside" object-oriented components. Those references must however be unambiguous.

Given that the knowledge worker environments are heavily dependent on commercial off-the-shelf (COTS) packages, then these COTS packages must be as object-oriented as possible. Given also that there are large, broad, and integrated databases that are used by the COTS packages, then the business information systems that support the databases must too be object-oriented so as to ensure that each business information system receives consistent data. When multiple business information systems store data then that data must be stored through singly defined, and consistently applied encapsulated business rules.

If during the evolution and maintenance of the knowledge worker environment an object oriented component is changed then the change/evolution must be available to and employed by all referencing object oriented components at the same instant. Otherwise, data and process corruption will occur.

Knowledge Worker Framework								
Viewpoint			**Man-Machine Interface**					
			Machine		**Interface**	**Man**		
Project	**Deliverables**	**Mission**	**Database Object**	**Business Information System**	**Business Event**	**Business Function**	**Organization**	**Primary Responsi-bility**
Specification	**Scope**	List of business missions	List of major business resources	List of business information Systems	List of interface events	List of major business scenarios	List of organizations	**Architect**
	Business	Mission hierarchies	Resource life cycles	Information sequencing and hierarchies	Event sequencing and hierarchies	Business scenario sequencing and hierarchies	Organization charts, jobs and descriptions	
Specification and Implemen-tation	**System**	Database object hierarchies	Database object models	Information system designs	Invocation protocols, input and output data, and messages	Best practices, quality measures and accomplishment assessments	Job roles, responsibilities, and activity schedules	**Architect and Engineer**
Implemen-tation	**Technology**	Policy execution enforcement	Logical DBMS Schemas	Information systems application designs	Presentation layer information system instigators	Activity sequences to accomplish business scenarios	Procedure manuals, task lists, quality measures and assessments	**Engineer**
	Deployment	Installed business policy and procedures	Physical DBMS Schemas	Implemented information systems	Client & server windows and/or batch execution mechanisms	Office policies and procedures to accomplish activities	Daily schedules, shift and personnel assignments	
Operation	**Operations**	Operating business	DBMS Views	Operating information systems	Start, stop, and messages	Detailed procedure based instructions	Daily activity executions, and assessments	

The knowledge worker environment is complex. That is, there will be objects of all object class types. With respect to the Knowledge Worker Framework, business object generally conform to the Business Information System column. Database object conform to the database object column. As a consequence there will be an interaction between the two columns. The most effective object environment will be one where the maximum amount of a business' object class is defined as database objects.

Figure 4 depicts a high level diagram of the models that are involved in a database object environment. There are six distinct technology independent models above the line and three technology dependent models below the line. The technology independent models represent the specification of a database object environment while the models below the line represent an implementation of database objects.

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 appendix. The remaining database object environment models are presented in the book, *Database Objects, The Foundation Stones of Enterprise Database*.

The mission model provides a description of the ultimate aim, goal, or database objective to be served by the database objects. The database objects operate within the scope of their business information systems. 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.

10. Database Object Composition and Execution Paradigm

Figure 5 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 data structure segments.

Each white rectangle in Figure 6 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

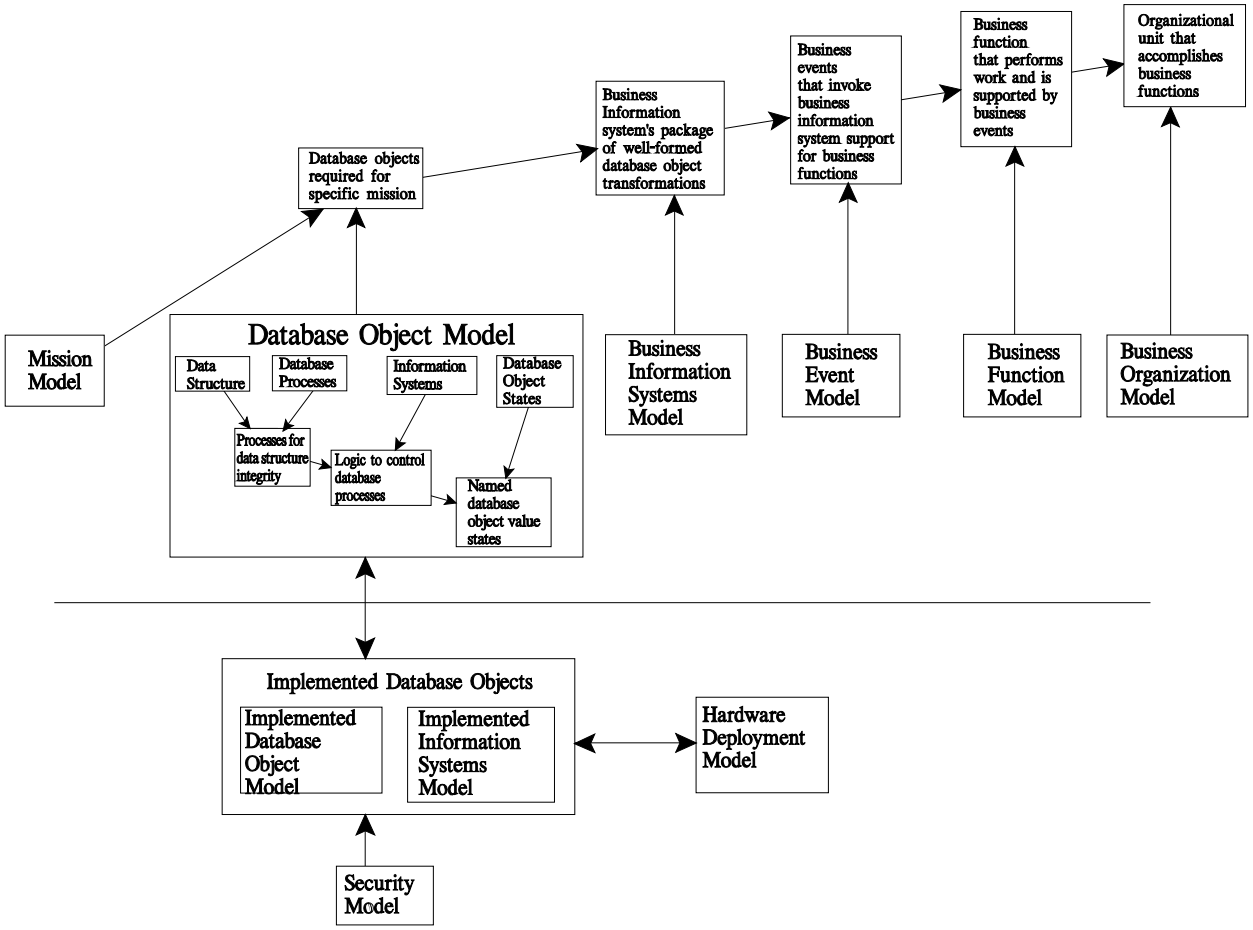
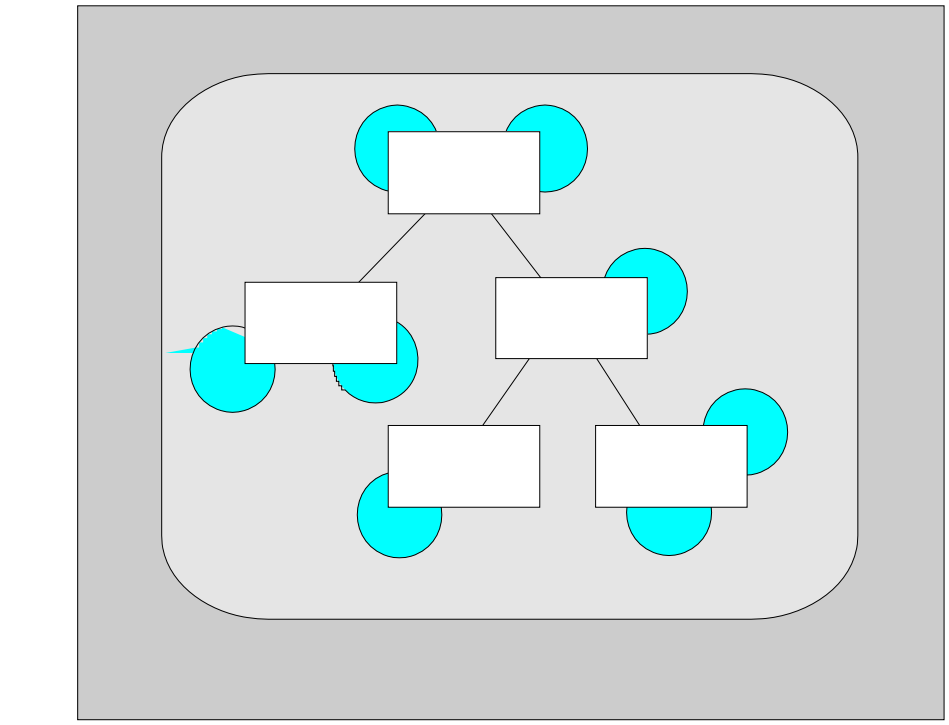


Figure 4. Essential Meta Models Required for Database Object Environments

the database object structure (5 segments) is in the NULL state only the structure exists. A database object in the null state is equivalent to a tree without leaves. As the database object progresses through states, different data structure segments 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 data structure segment is equivalent to an ANSI SQL/3 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. 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



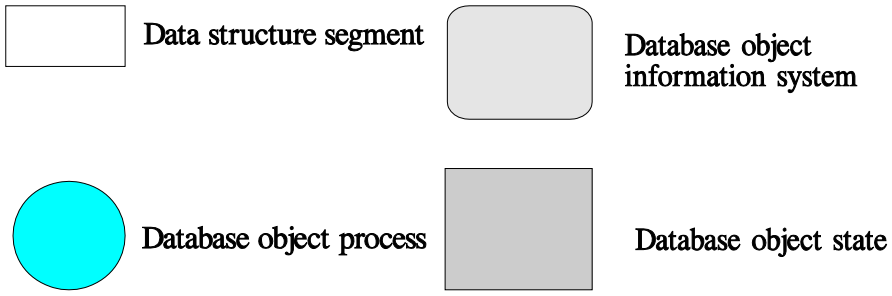


Figure 5. Database Object Composition Paradigm

group that allows a contained repeating group. For example, DEPENDENTS containing HOBBIES.

Intersecting each database segment, represented in Figure 6 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 segment 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.

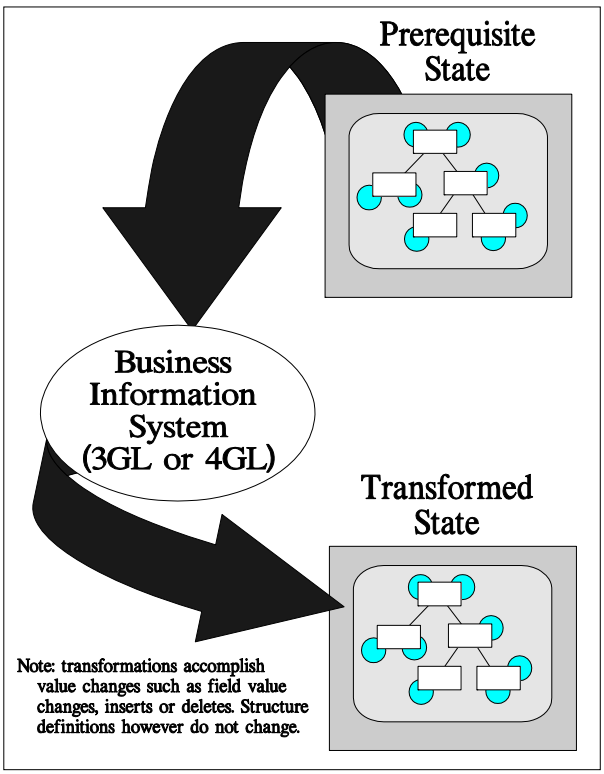
The action type of database object process includes BEFORE, AFTER, and then verb types such as INSERT, DELETE, MODIFY. The action type and verb type enable six different actions that to occur for each database object segment change. The process logic contained in each database object process can be to 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 segment instance was installed through an AFTER INSERT action that would compute the age and then perform a database object segment modification to install the age value.

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

Surrounding the entire database object in Figure 6 is an outer database object information system. In the example of Figure 6 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 segment instances. In this example, there may merely be a single database object segment 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 EMPLOYEE REQUISITION stage, and adds to it a list of persons who represent EMPLOYEE candidates. In this situation, the database object segment 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



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 6. Database Object Execution Paradigm

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 5 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. 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.

The execution paradigm is shown in Figure 6 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 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 segments.

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.

The reason database objects are critical to client/server, heterogeneous, multiple-DBMS, world wide environments is depicted in Figure 7 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 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.

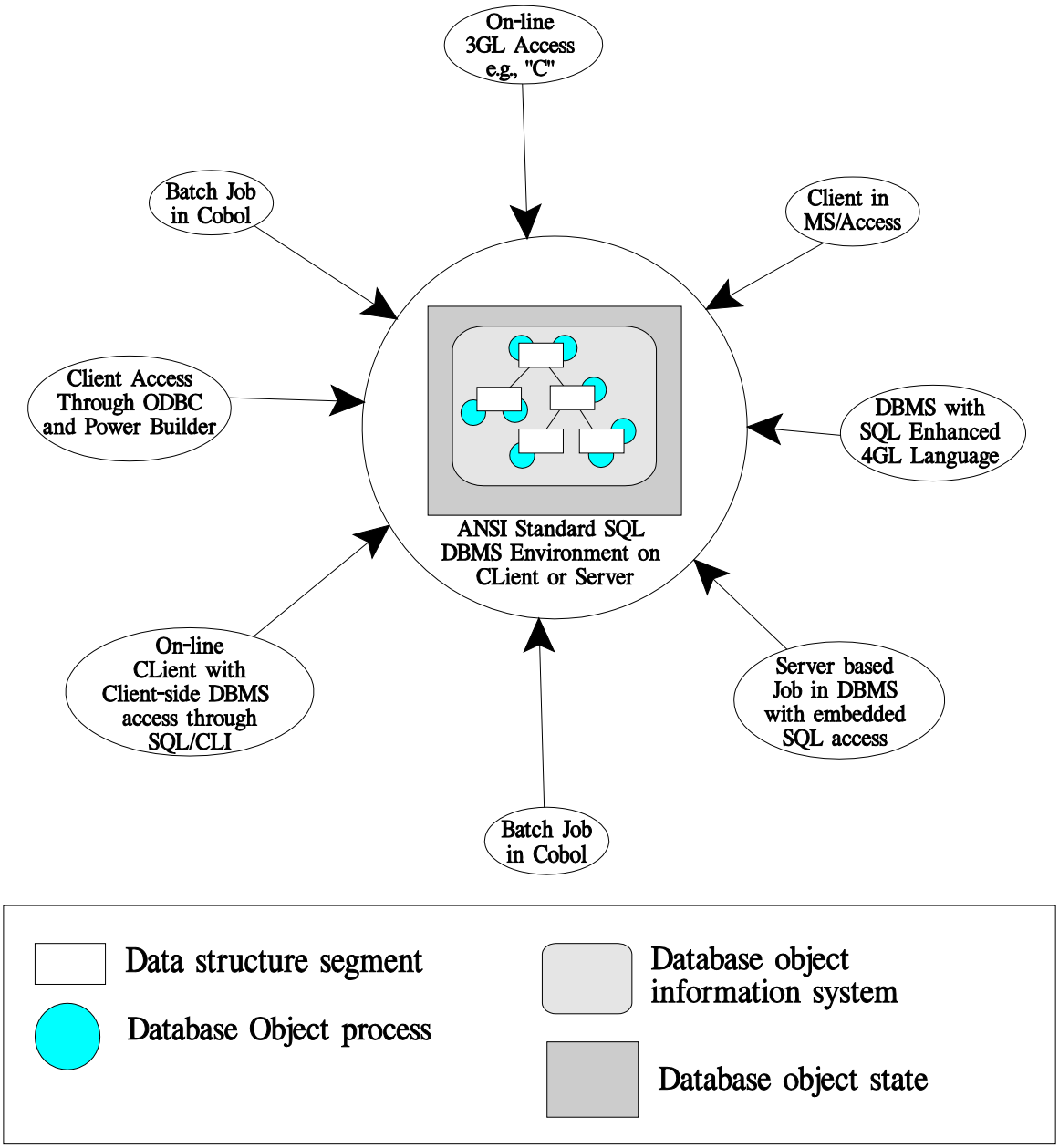


Figure 7. Myriad of Database Object Invoking Agents

There are five key evaluation criteria by which an object approach can be evaluated:

- Does the approach to objects make business applications easier to analyze and design. Are the results are more complete, self contained, and intuitively obvious to understand?

- Is the result of the analysis and design easy to encode in unambiguous syntax within an ANSI standard language?

- Are the object classes that result from compiling the object class syntax easy to instantiate and are the object classes able to be interrogated as to their class definitions?

- Are the object classes and the objects able to easily ported to a large variety of computers from PCs to mainframes across a wide variety of operating systems?

- Are the object classes and objects able to be easily accessed across networks of a large variety of computers from PCs to mainframes and across networks of a wide variety of operating systems?

These evaluation criteria should be assessed against any object class and object instance proposal that is brought before Riverview.

With respect to database objects, currently the main SQL vendors, e.g., Oracle, Sybase, Informix and IBM can all implement the database object paradigm stated in this appendix. The resulting syntax however is not currently ISO/ANSI standard. It is the intention of the ISO and ANSI database language committees to maximize the standard syntax so that all the above evaluation criteria can be answered in the affirmative.

11. 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, 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/3 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 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 environ