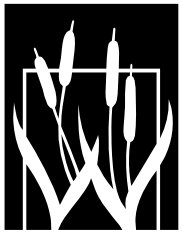


Whitemarsh
Information Systems Corporation

Database Objects Brief Introduction

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. Overview

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 [information system] that accomplishes whole business transactions in a certain manner.

While 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 modules/systems in the form of an executable that behaves according to certain fixed rules.

A database object is none of these. It is its own class. While database objects share some common names and definitions with the other three object classes, that is, encapsulation, inheritance, and polymorphism, database objects are unique to both database and DBMS.

Database objects are identified, designed, implemented, operated through, and evolved, or maintained through just one type of data processing facility, a database management system (DBMS). If the available DBMS is an ANSI SQL3 DBMS (download SQL_BOM from the www.wiscorp.com) 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.

Database objects are essential to the proper understanding, specification, implementation, and maintenance of world-wide heterogeneous databases. Database objects fit within the enterprise's Knowledge Worker Framework (download the Knowledge Worker Framework book from www.wiscorp.com).

Database objects are not new. They were started in certain DBMS types (e.g., IDMS, IDS, GIM, Inquire, and Adabas) in the late 1960s. Relational DBMS such as DB2, Oracle, Informix, and Sybase, however, stopped the march to database objects dead in its tracks. It was not until the ANSI SQL3 data model moved away from relational and not until a whole programming language was incorporated into ANSI SQL3 that the march to database objects restarted. The newest versions of IDMS, DB2, Oracle, Informix, and Sybase have all started to support the data structure and process features essential for database objects. 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.

Database objects were formulated almost 20 years ago by the late Matt Flavin in his 1979 Yourdon Monograph, Fundamentals of Information Modeling. During the Seventies, Matt (who worked for Infodata of Rochester, NY and Fairfax, VA.), 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.

Database objects existed only on paper only until the ANSI database language's committee, X3H2, working since early 1993 on the specification of SQL, formulated the essential linguistic components of database objects.

2. 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 businesses 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 resource's spectrum be satisfied?

Business data needs far exceed today's DBMS's two dimensional table capabilities and simple column-based constraints. 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, operating systems, or DBMS vendors, 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–data is just executed 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.

3. Components of Database Objects

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

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 is the 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 makes 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 non-persistent database objects is 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/99 table structure. With SQL/99, columns can support very complex structures, such as simple values, lists, sets, multi-sets, and abstract data types of arbitrary complexity. 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.

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 table 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/99 such product sales information would require multiple tables with the attendant keys, joins, and computer processing melt-downs. 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-tables contain the same four-part composition:

- Data Structure: The set of data structures (simple and complex collections of tables) 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 columns (simple or complex), 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, accomplishing 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/99 DBMS through the SQL/99 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.

4. Database Object Composition and Execution Paradigm

Figure 1 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 1 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 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/99 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 group that allows a contained repeating group. For example, DEPENDENTS containing HOBBIES.

Intersecting each database segment, represented in Figure 1 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

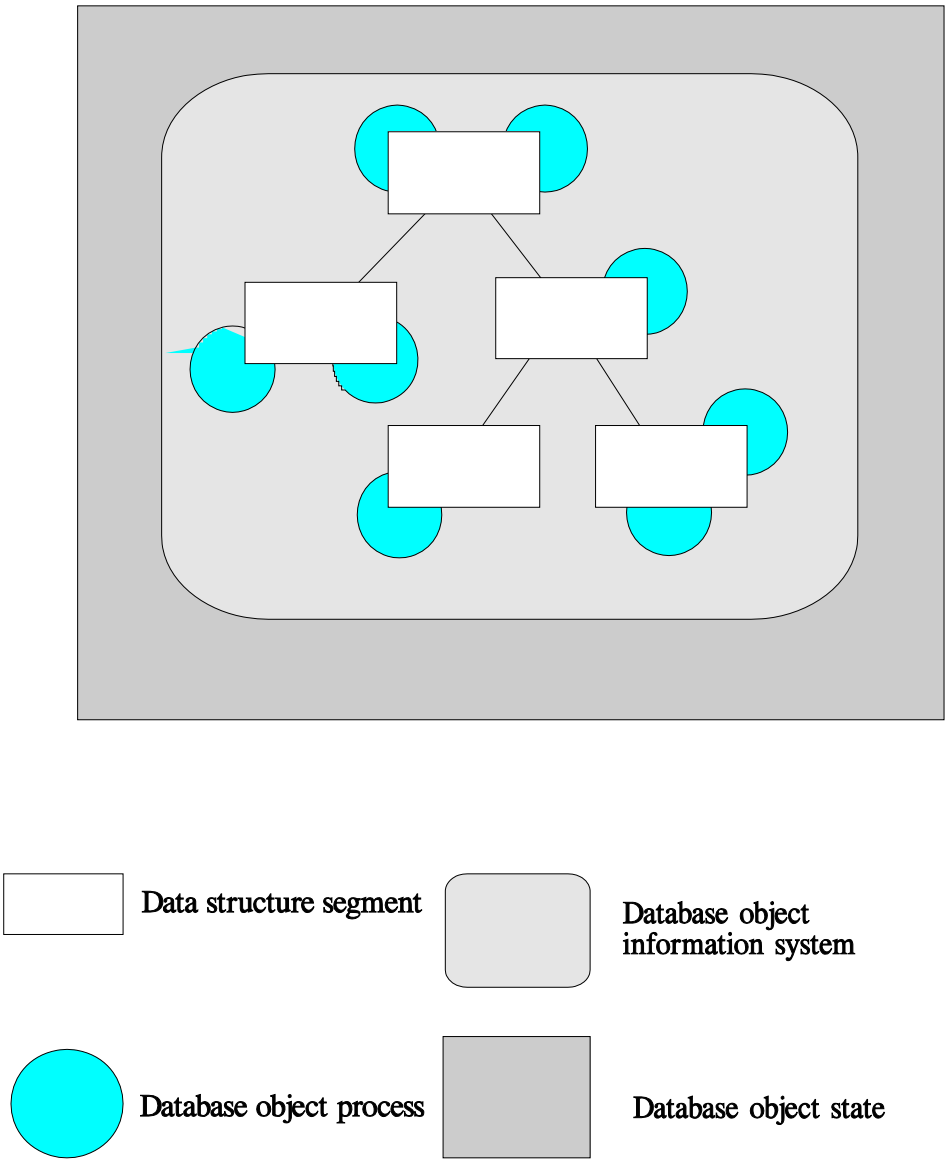


Figure 1. Database Object Composition Paradigm

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.

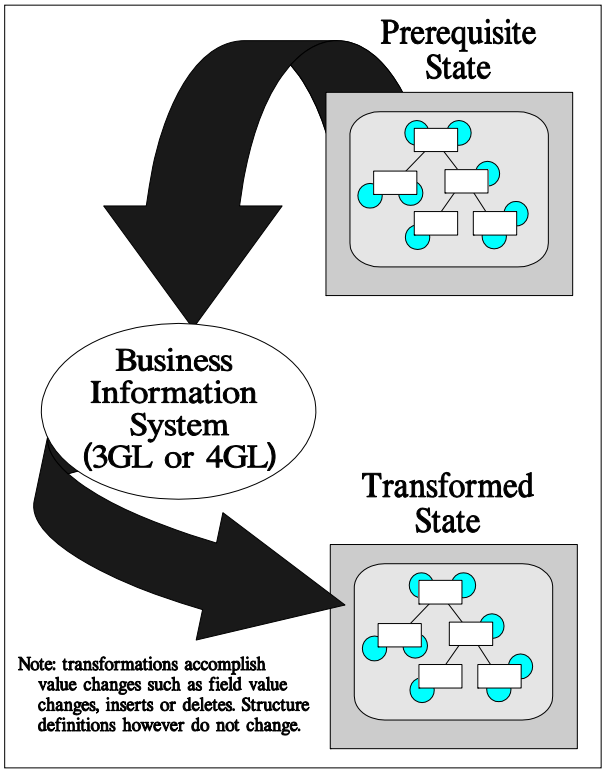
Surrounding the entire database object in Figure 1 is an outer database object information system. In the example of Figure 1 only one database object information system is shown. In reality, many database object information systems may cause database object transformations from one value state to another. 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 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 1 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 2 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



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 2. Database Object Execution Paradigm

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 3 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.

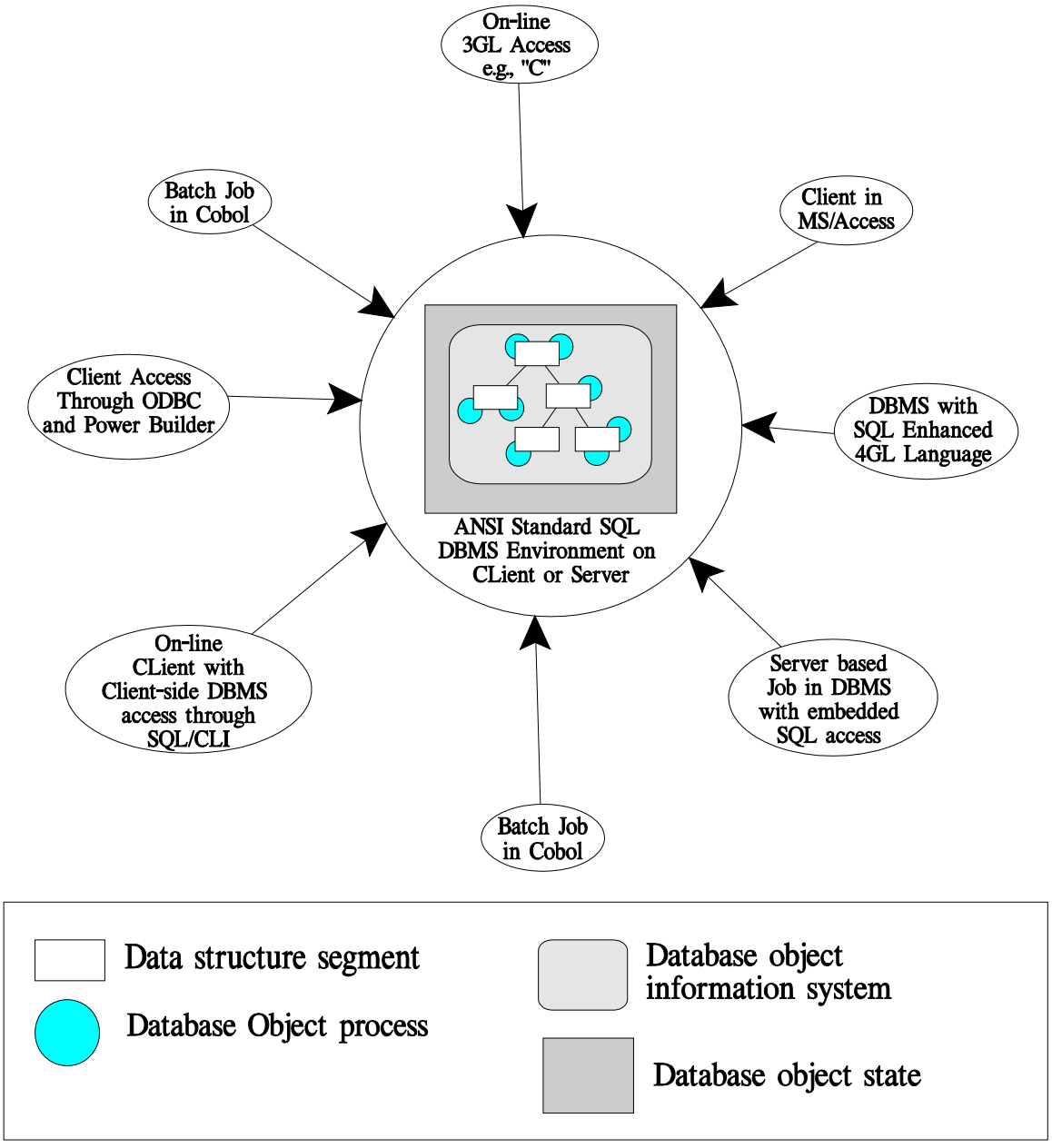


Figure 3. Myriad of Database Object Invoking Agents

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.

5. Database Objects Summary and Benefits

The benefits derived from database objects include:

- Whole containment within SQL/99 DBMS

- Access to both type and instance components

- Complete expression through syntax

- Import and export through ISO/ANSI standard SQL/99 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/99 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.