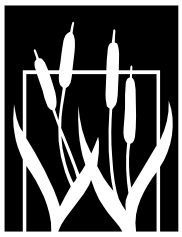

Whitemarsh
Information Systems Corporation

Data Is Executed Policy

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

Table of Contents

1. Objective

The objective of this ***Whitemarsh Short Paper*** is to present an approach to *data as executed policy* including the processes for constructing the procedures that capture and transform data from one value state to another. ***Whitemarsh Short Papers*** are available from the Whitemarsh website at:

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

An ancillary objective of this paper is to show how enterprise[1] policies are reflected in database object classes which, in turn, are allocated to enterprise resource life cycle nodes. This enables integration, nonredundancy, conflict resolution, and efficient and effective information systems plans.

2. Topics Covered

The topics is this paper include:

- Policy and Procedures
- Foundation for Quality Database
- Database Object Classes
- SQL DBMS Implementation of Database Object Classes
- DBMS as the Appropriate Container
- Database Object Class Discovery

3. Policy and Procedures

Data is the "what" that remains after *policy* is executed. Thus, *Data is Executed Policy*. When an organization creates policies, it necessarily creates as its companion, procedures. Together, policy and procedures are set into place to the run the business. As the business runs, data, the consequence of policy execution, is created and stored in databases. These databases become the persistent memory of the organization.

"Data" specifications are thus policy definitions. Similarly, process specifications are procedure definitions. Consequently, all data (i.e., policy) specifications are metadata. All process (i.e., procedure) specifications are also metadata. A metadata database (e.g., Whitemarsh Metabase) is a database for all Policy and Procedure Specifications.

[1]. In this paper, “enterprise” implies an organization units. As such an enterprise may be small or very large. It may relate to the business as a whole, a division, region, or some other business unit.

Any policies and procedures not specified or transformed to metadata and which do not result in database data are not only just anecdotes, they also cannot lead to enterprise persistent memory.

Finally, whoever specifies data and processes are the true specifiers of policy and procedures. Who are they in the enterprise? Senior executives, policy makers, or possibly systems analysts, database designers and programmers. The choice should hopefully be obvious.

4. Foundation for Quality Database

Quality-database represents a technological expression of organization, clarity, and precision. It may or may not be computerized. If it is, it may exist on a desktop, a server, or a large mainframe. Finally, a database may or may not be centralized. Notwithstanding the mode, the mechanism, or the form of database implementation and operation, the codification of and adherence to data semantics, which are the rules for meaning, validity, and usage, are prerequisite to successful database. Quality database can only follow from quality policies and procedures.

When a database is on a computer, it represents the automation of the knowledge component of a business, which is manifest through the business's quality operation, planning and management. With quality-database, business management can research the past, organize the present, and plan for the future. To have quality-database is to have defined policy because:

- Each database object class's data structure within the database is the data representation of policy definition.

- The database object class's processes through which rows are added, deleted, and modified are the mechanisms necessary for policy execution, that is, the policy's procedures through which database objects are transformed from one valid state to another.

- A fully defined policy includes both its complete definition and its necessary steps for coherent execution.

- Interrelated collections of rows across multiple tables of one database object class and across multiple database object classes form more comprehensive policies.

When data is seen as executed policy, and is realized through database object classes, then quality databases support the following within the enterprise:

- Business information systems that are a coherent union of the policy that, in turn, support the execution of procedures that represent the accomplishment of the policy.

- Consistent collection and/or modification of policy instances through the life cycle of the policy.

- Consistent execution of policies whenever, wherever, and however deployed as the essence of the policy and the totality of its critical procedures are encapsulated within the database object class itself

- Minimized redundancy and consistent policy implementations across distributed environments as the database object class can be distributed through encapsulated strategies

- Comparable instances of deployed policies that are independent of hardware architectures and operating systems

What forms the basis of a 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, in this case, 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 valid, predefined state to another through well defined rules. The minimum value states are null and valued. The internal behavior of a database object class as it transforms database objects from one state to another is immaterial to its user. Database object classes conform to the requirements of business rather than the converse. Database objects are the corporate memory of the enterprise. All the rest is anecdote.

Policies and procedures, that is, database object classes, 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's policy is to be profitable, then its balance statement, produced by processing all the general and subsidiary journals are the measure of adherence to the policy. If policy is met, the enterprise must be profitable.

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. Procedures include, for example: establishing an employee requisition, accomplishing employee hiring, and performing employee assignment.

Enterprise-database is an organizational operating condition in which there are 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 class specification, implementation, and evolution 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 begins first with quality-database object classes founded upon the policies and procedures surrounding their specification, implementation and evolution.

Value proceeds not first from the database object, but from the database object class. The consequences of quality-database object classes are "real" database objects. The information technology assets of the enterprise are both its database objects and its database object classes.

For example, if only database objects are needed and valued, then only musical notes would be needed for a great symphonic score. Performances are differentiated however, from the grade-school band to a 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.

Finally, an enterprise is interoperable only to the extent to which it's metadata is nonredundant, not conflicting, and integrated. To believe otherwise or to believe, for example, that the only thing that has to happen to database objects is that they are transformed to XML in order for an enterprise to be interoperable flies in the face of common sense and simple logic. Is it "record" or "record?" Is it seam or seem? Is it 3.79 MPH or 3.79 KPH? Is it rounded or precise? Is it a running total or an amount at that instant? Data alone doesn't provide the answer. Data and semantics do, but only in context in an integrated, non conflicting, and non redundant manner.

5. Database Object Classes

When executed, policy consequence, that is, created data, must be stored in a very business-coherent manner. Traditionally, however, databases are comprised of "tables," and the data for those tables are stored in rows. A simple, single valued column set within a table is generally insufficient to represent the data required for an entire policy. Collections of related tables are thus required. Whitemarsh calls such a collection of tables a database object class. Within the Whitemarsh Metabase, database object classes are independently defined. Tables associated with these database base object classes are defined within the Implemented Data Model.

Database object classes naturally exist within database applications. They are easily "seen" in entity relationship diagrams as clusters of entities that seem to have a common root. Customer is a ready example. There would be a main customer table, then tables for contacts, contacts, orders, locations, classifications, customer groups, and sales statistics. The main table would likely have a single column primary key. Subordinate tables would have multiple column

primary keys and most often the customer's primary key column would be the first column in this column set.

Database object classes consist of more than just data. They also contain database object table processes, states, and overarching database object classes processes. The database object table processes are required to ensure that every database object table row is properly created, changed, or deleted.

The states are required to reflect the complete execution of a policy. Multiple states are required because database objects commonly have a life cycle. That is, from the null state through a series of valued states and finally back to a null state.

The overarching database object class processes, also called database object information systems, are required to ensure that the data received from an application system is properly applied to all database object tables. If not properly applied, then the entire database object state change is considered failed, and must be rolled back to the prior state. These four components provide positive control over the overall states of database objects that comprise the totality of data in the database.

Database objects range from the trivial to the complex.

A trivial database object is:

1. A simple data structure (a set of single value columns),

2. 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 collection of well defined, third-normal formed, interrelated tables about a single policy,

2. Instantiated through collections of database processes that are

3. Part of one or more collections of complex information systems, and

4. 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.).

6. SQL DBMS Implementation of Database Object Classes

SQL DBMSs that conform to the ANSI SQL 2003 standard are generally able to implement database objects. The SQL 2003 standard however does not contain a schema object, Database Object Class. Consequently, in SQL 2003 compliant DBMSs, a database object cannot be defined as a collection of tables. Rather, from the definition of a complex database object class cited above, the SQL 2003 standard creates database object classes through complex data structures defined within the scope of tables.

Within SQL 2003 tables there can be single-valued columns, arrays, groups, repeating groups and nested repeating groups. While not ideal for all the different requirements for database object classes, these structures are generally adequate. Also within tables there can be column and table constraints and triggered processes. Finally there can be fully defined methods. While the SQL 2003 standard does not formally address states, these can be achieved through the SQL View facility coupled with constraints, triggers, and stored procedures.

The external specification of a database object is independent of its DBMS vendor's internal implementation. DBMS vendors are free to implement the ANSI SQL2003 specifications as they like just so long as two conditions are true: the database object specifications are portable from one SQL2003/DBMS to another, and the behavior of the same database object is the same, from the user point of view, even though the database object is implemented by different SQL2003/DBMS vendors.

7. DBMS as the Appropriate Container

Database management systems are the appropriate containers for the persistent, enterprise-wide definition of policy and its companion procedures through database objects for the following reasons:

- DBMSs are inherently multi-user
- DBMSs are application information system independent
- DBMSs are independent from natural, report writer, and "fourth generation languages (4GL)"

DBMSs are innately intended to address the definition, capture, update, and reporting of data across collections of users within the enterprise. This has been one of DBMSs' key goals and characteristics ever since DBMSs were first brought into the computing environment in the early 1960s. Since then, the range of applications served by DBMSs has greatly expanded. So too has the range of users from operational to strategic. It follows then that the proper place for ultimately capturing and storing policy executions, that is, data, are databases. Databases managed by DBMSs have the greatest chance of being consistent and managed in an acceptably rigorous manner.

DBMSs, by definition, are application information system independent. This level of independence has been expanding since the early 1960s. Today, DBMSs are able to be independent of primitive data storage, data transport, the modes and means of data capture, and finally of data presentation to/from the end user. These independencies are critical because of multiple computer hardware architectures, different operating systems, and different presentation layers. With DBMSs, service oriented architectures are possible. DBMSs are able to hold all the critical metadata, all the data, and then all the network infrastructure data in a standard, efficient, and effective manner.

Finally, because DBMSs are the natural data definition, storage and access mechanisms for so many different computing languages, it is essential that process specifications employed commonly across all these different language forms be executed within the domain of the DBMS.

While business rules may be independently defined in order to support rules that span business domains, DBMSs and databases, the actual binding and execution mechanism of the business rules is most logically within the DBMS as it is the most common layer to all the different language-based processes.

8. Database Object Class Discovery

Discovering database objects is straight forward. First the enterprise's missions are defined. Then from each mission leaf, database domains are defined. Then, the entities that are readily apparent within the database domain statements are examined to determine if an entity is a unitary business fact (e.g, Salary), or a class of facts (e.g, Customer Sales Statistics), or a collection of classes of facts (e.g., Customer with locations, sales statistics, classifications, and the like). This last category is the database object class.

Discovery, while important is not sufficient. Database object classes must be placed within enterprise resource life cycles. By assigning database object classes to the nodes in the resource life cycles, their enterprise-wide deployment and use can be fully known. For example, the customer database object class may be important to all asset determination, personnel hiring and allocation, facility requirements and the like. Allocation of database object classes to resource life cycle nodes enables cross referencing to corporate policy specifications, to data uses across the resource life cycle nodes, and to application information systems. These will surface redundancy, or worse, data value creation conflicts.

Policy execution errors are probably the greatest source of conflicts in decision making. Errors exist because of conflicting granularity, precision, and time-synchronization. These errors can now be identified and resolved.

9. Conclusions

The practical application of the points made in this paper include:

- Policy is critical to specify in a coherent manner to have consistent, repeatable behavior across the enterprise.
- Procedures are the "hand and glove" companion to policy.
- Policy, when executed, results in data, which when stored in broad, integrated but nonredundant databases become the enterprise's persistent memory.
- Data specifications are the maximal boundaries of policy within the enterprise.
- Coherent policy leads to coherent data. The inverse is also true.
- Policy specifications, that is, data specifications are best represented as database object classes which, in turn consist of four parts: data structure, database object table processes, states, and database object information systems.
- Database management systems are the best and most natural place for the technology definition of database object classes.
- SQL 2003 standards' compliant DBMSs are an acceptable but not ideal method of defining, instantiating, and employing database objects within enterprise decision making.
- Database objects, set within the context of resource life cycle nodes enable enterprises to inventory their application information systems, determine "holes," and most importantly determine conflicts and redundancy.
- Application information system redundancy, if unexamined can lead to policy execution conflicts, which, in turn lead to reductions in business efficiency and effectiveness.
- Enterprise policies, reflected in database object classes which if allocated to enterprise resource life cycle nodes enables integration, non redundancy, conflict resolution, and information systems plans.

10. References

The following documents are available free from the Whitemarsh website:

Free Whitemarsh Material	
Paper	**URL**
• Knowledge Worker Framework Book • Enterprise Database Whitepaper • Metabase Overview • WRAD Conference Talk • Data Standardization Talk (DAMA)	http://www.wiscorp.com/DatabaseDesignInformation.html
• Resource Life Cycle Analysis • Database Objects	http://www.wiscorp.com/tdan.html

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

Members Only Whitemarsh Materials	
Paper	**URL**
• Enterprise Database Overview • Information Systems Planning (book, courses, and papers) • Knowledge Worker Framework (book, courses, and papers) • Management Challenge • Managing Database: Four Critical Factors	http://www.wiscorp.com/wwmembr/mbr_products_edb.html
Database objects (book, courses, and papers)	http://www.wiscorp.com/wwmembr/mbr_products_do.html
• Data Integrity Rules • Data Is Executed Policy (paper and courses) • Database Project Work Plan Templates • Information Systems Development (book, course, and papers) • [Whitemarsh] Methodology Phases 1 and 2 • Work Breakdown Structures	http://www.wiscorp.com/wwmembr/mbr_products_dp.html

Members Only Whitemarsh Materials	
Paper	**URL**
• Metabase Overview • Metabase Rational Knowledge Worker Framework and Meta Models • Resource Life Cycle Analysis (papers)	http://www.wiscorp.com/wwmembr/mbr_products_mb.html
• Achieving Data Standardization (book and courses) • The Data Standardization Problem	http://www.wiscorp.com/wwmembr/mbr_products_dq.html