

The Data Administration Newsletter (TDAN.com)

[Robert S. Seiner - Publisher](#)

[Main Menu](#)

[New Articles](#)

[Article Archive](#)

[Special Features](#)

[What's New](#)

[Contact the Publisher](#)

ITERATIONS OF DATABASE DESIGN

Written by Michael Gorman - Whitemarsh Information Systems Corp.

[\[Return to the Article Archive\]](#)

Published in TDAN.com October 2003

This paper presents a high-level strategy for developing database designs. The strategy begins with a definition of database, describes the preliminary steps for arriving at a design and enumerates and briefly describes the other initial database design iteration cycles. There are three major cycles of database design, and within each, minor cycles. The major cycles are: conceptual, logical, and physical. The conceptual cycle causes a database design to be created that accurately reflects the in-place business policy of the enterprise. The logical database design starts with the conceptual design and then incorporates the requirements of the particular DBMS through which the DBMS operates. The final cycle, physical begins with the logical database and folds-in the requirements of the operational environment. These environments range from monolithic mainframe through client/server to single user PC. These stages, conceptual to logical to physical are not just transformations. They exist independently and may take on a different table structure. They must, however all be mapped one to the other.

During the entire database design process the database design from each cycle is maintained in a repository. Done properly, each major cycle transformation (that is, conceptual to logical to physical) is expressible through SQL syntax changes. This way the re-transformation can always be accomplished.

The iterations of *conceptual database design* are:

- Business Policy
- Historical
- Audit
- Security
- Database administrator data versus business data
- Generalized versus specialized structures

The iterations of the *logical database design* are:

- 1st cut performance
- DBMS DDL effects
- Business Keys versus DBkeys
- DBMS physical database effects
- Interrogation analysis effects

The iterations of the ***physical database design*** are:

- System control effects
- Client/server effects

2. CONCEPTUAL DATABASE DESIGN ITERATIONS

A database is the storage, interrelationships among, retrieval, and update mechanism of retained executed corporate policy. Data is executed corporate policy. For example, the data that results when an employee is hired, forms are filled out, processes occur, and the person reports to work. What do we know of that set of policy executions? The data that's in the database. So, database design must follow clear and cogent policy definition. A database's quality is a direct reflection of the organization's. Once these iterations are complete, the database's design is said to be conceptual. Once specified, the design changes only when business policy changes. The iterations of conceptual database design are:

- Business Policy
- Historical
- Audit
- Security
- Database administrator data versus business data
- Generalized versus specialized structures

The first, the business-policy stage, causes a third normal form database design, which consists of tables of incorporated business data elements (now called columns), primary keys to ensure unique instance selection, and foreign keys with referential action rules to implement the critical/essential business relationships.

All tables must either be the data structure component representation of a database object or be a data structure subset of a database object. A database object is the collection of all data structures that correspond to an essential business policy. For example the set of policy surrounding employees, contracts, fixed assets, and the like. Every database object should be essential to the business. If it isn't, its existence should be questioned. For example, corporate recreational programs might be in the "nice" or "should" categories but are certainly not in the "essential" category.

Every table must contain a business data element based primary key. The table's primary key, which represents a named collection of columns, when valued ensures that only one row of the table is identified.

In the business policy stage of database design, all primary keys must be based on business data elements to ensure rigorous application of business policy analysis. So, for an employee database the primary key might be the employee-identifier, that is, the employee's <full-name><birthdate>. If the organization has a long stated policy of creating an employee number, then employee-number could be employed as the primary key providing there is the assurance that the employees and the employee numbers are in a one-to-one relationship.

Whenever a table is not the primary/root table of a database object, it must be related to the root table or next higher table in the collection of database object tables through foreign keys.

A foreign key is a named collection of columns within a table that is actually the primary key (collection of columns) from another table. The purpose of a foreign key is to act as a selector of rows in one table based on the primary key of another table. The foreign key definition acts as a relationship between the two tables. The basis of the relationship must be defined and be firmly based on business policy. For example, the relationship between an employee and their dependents, that is the employee primary key resident in the dependent table must be an indicator of the legal relationship between the employee and their dependents (natural, adoption, etc.). The foreign key's name within the dependent table should reflect the basis of the relationship.

In addition to the relationship declarations, referential action rules must also be specified. A referential action rule is an action that occurs whenever a referential declaration is violated. For example, an attempt to load a dependent with a nonexistent employee identifier. The referential action rule, now violated would state the consequences of the violation: REJECT or SET_NULL. The first action rejects the attempt to load the dependent. The SET_NULL accepts the dependent but sets the employee identifier to be NULL (that is, attached to the "don't know" employee).

Because database objects can be interrelated, a table must exist that has as its primary key the concatenation of the two primary keys of the related database objects. In addition, there are two foreign key subsets to the respective primary keys. Any additional columns to the database objects associator serves as a value based rationalization for the interrelationship between instances of the database objects.

The business policy stage database design should:

- Update in place (no historical data)
- Have only simple, primitive columns

That means that whenever there is a row update, existing data is changed rather than making a copy of the existing row, changing it, and then storing both the old and new versions. Finally, all data should be atomic and primitive. Atomic data means that columns are simple and single purpose. There are no compound columns (i.e., <fullname>, which is really <last name>, <first name> and <middle initial>). Secondly, primitive means that there are no derived columns such as <total department salary> which would be computed by totaling up all the salaries from the employees known to be in the department.

The business-policy stage design presumes an infinite size and infinite speed computer. After the conceptual database design is completed, it is thoroughly prototyped. That is, it is logical through a

code-generator based DBMS such as Oracle/CASE with generators to thoroughly evaluate the correctness of the business policy represented by the database.

Once the database design is iterated through prototyping, usually 3 to 5 cycles, the database design is ready to proceed to the next design iteration.

2.2 Historical Data

The historical iteration causes the entire database design to be evaluated with respect to the retention of historical data. For example, are all changes of addresses to be kept, or just the first five? Are entire rows of data from a table to be kept? Or, are structures to be established to retain only multiple occurrences of a specific column. Only business and legal policy analysis indicates the answer. And the answer affects the granularity of the primary key, corresponding foreign keys, or just the addition of new structures.

The audit iteration causes the incorporation of a "who-done-it" set of columns in the table. The critical issue is whether its on the basis of a column or the entire row. The impact is similar to that of historical data. If by row, then the date_&_time of last update also has to be part of the primary & foreign keys. In addition, there might need be a column for the reason of update.

2.4 Security Control

The security iteration is centered on whether there is to be column or row level security. Depending on the answer, and depending on the DBMS chosen, very different strategies have to be taken. No strategy is without significant cost both in terms of ultimate machine resources used and also the required bureaucracy.

Databases consist of two types of data: reference data and fact data. Reference data is a term applied to the set of tables of valid values such as State codes, product codes, product unit prices, and discount rates. Fact data refers to the actual business data that makes up a contract, an employee, or a fixed asset description.

A critical component of any well organized and controlled database are the data tables that control and restrict valid values. Every column of every table must be examined to determine if its values is restricted to a reasonable number, say, 100 or so. In these cases, and if the columns are used for selection or ordering, then building a reference data table that restricts the values in the fact data table may be in order.

For each reference data table, additional computing resources have to be expended for table definition, referential integrity and action definition, data loading and update, and other facilities for backup, recovery, and the like.

A specialized database table is in third normal form. If a database table is in second normal form, then the meaning of one column's value depends on the value from another column. Such indirection is useful whenever the meaning of the column may be unstable, or when entire structures of columns may be unstable. For example, if there were sets of tables for each type of legal instrument, say contracts, purchase orders, liens, mortgages, and the like, then whenever a new legal instrument is created then a new table would have to be created. A way to avoid continuous database reorganization is to create a generalized table structure that allows the type of instrument to be one of the columns and then other types of generalized columns to store the data.

In general, generalized structures should be avoided as they inhibit the use of query languages, and the ease of precise updating. When, however, the nature of the table and the set of columns in the

table are not yet stable then the creation of a generalized structure table may be the only way the make real progress with the project rather than spending all the time redefining and reorganizing.

Once the database has proceeded through its basic design iterations and has been accepted as a fair representation of the necessary business policy, the database is ready to be implemented. During the implementation process, the DBMS is chosen and its requirements are reflected in the logical database's design. The logical database design is an exact transformation of the business policy represented in the conceptual design. Any violation of the business policy representation is an error. The DBMS design effect iterations are:

- 1st cut performance
- DBMS DDL effects
- Business keys versus DBkeys
- DBMS physical database effects
- Interrogation analysis effects

Part of the analysis required to build a complete database design is to capture the types and kinds of queries and reports that are common or predominate. Included in the analysis capture is a pseudo-code navigation of the third normal form database. This quantifies the table accesses. Given average computer performance, the quantity of I/O cycles and thus elapsed time can be computed. These statistics, coupled with hourly and daily reporting frequencies can build the reporting load profile that can be contrasted with the ideal data model.

If the performance requirements imposed on the data model produces unacceptable statistics, then database redesign strategies must be started. These range from having to create derived columns to the creation of entirely new data structures.

The database definition languages of ANSI standard SQL DBMSs are becoming richer and richer. This means that more and more of the application functionality can be defined in the database's DBMS. For example, if all dates for contracts must be greater than TODAY, then a function can be stored in the database's definition to ensure that constraint. This way, if contracts are entered through several different application programs and/or different languages, the DBMS-based integrity rule will always and automatically be enforced.

Broadly, the classes of data integrity rules that can be installed into the DBMS' DDL are:

- Column constraints
- Table constraints
- Triggers
- Assertion
- Referential Action Rules

Each class of data integrity rule must be evaluated for applicability in concert with each application to ensure that the applied-all-the-time rules should be universally applied.

Another significant class of DDL effects is the SQL view. A view is a predefined set of database navigations (nested selects) that ultimately produce a single row of data for use by an application.

Collectively, DDL-based data integrity clauses and views can have a profound effect on the complexity of application programs, and on the actual choice of language employed for the application program. Every effort should be made to maximize the use of these two facilities.

The product of this database design iteration is a new product in a new language. Previously, all the database design product should be stored in a repository like one mandated by FIPS-156, or a non-FIPS compliant repository like ORACLE/CASE. In either situation, the database design iteration tool should amply support the migration of the database's design from a pure business-policy-based design to one that fully accommodates all the various types of data integrity rules. The product of this design iteration should be a computer file that contains the ASCII strings of an ANSI standard SQL database data definition language. This file should be "readable" and compilable by any ANSI standard SQL DBMS.

Up to this point in the database's design, all keys have been sets of columns that represent business data elements. For example, the primary key of EMPLOYEE, the <employee identifier> actually consists of the columns <last name><first name><middle initial><birthdate>. While from a business policy database design point of view that is a perfectly reasonable primary key, it has two major things wrong with it: update ability, and propagation.

A foreign key is the primary key from the "referenced from" table. Thus the primary key's value is replicated where ever it is employed as a foreign key. For example, for EMPLOYEE, <employee identifier>, that is, the columns <last name><first name><middle initial><birthdate> must be included in the employee's ADDRESSES, EDUCATION, PRIOR JOBS, and SKILLS tables.

Further, if the EDUCATION table has a dependent table, COURSES TAKEN, then the EDUCATION table's primary key <education identifier>, that is, <employee identifier><school name><year of graduation> is included in each COURSES TAKEN row.

Suppose further that an employee is involved in a project and that PROJECT has TASKS, and each of those tasks consumes time that is reported on a weekly time card that allows capture of time by project, task, and by day, then these rows too have <employee identifier>.

Even ignoring the amount of space required, consider the key value updating effort when an employee's name changes, or if the employee's birthdate is wrong. The quantity of rows that might have to change could well number in the tens of thousands.

To prevent unnecessary propagation and to prevent unnecessary resource consumption for updating relationships just because of erroneous data, many database design teams have substituted these information bearing primary keys with DBKEYs. A DBKEY is a DBMS controlled, arbitrary number (usually a double word integer) that is employed solely for the basis of installing uniqueness and for relating tables, one to another. While the positive effects are obvious, there are costs. For example, <employee identifier>, which is really <last name><first name><middle initial><birthdate> is now not in the COURSES row. To find the actual employee's name, additional selects have to be executed, first for EDUCATION, and then for EMPLOYEE. Because of this extra effort, the following guidelines are commonly followed:

- Do use DBKEYS when they serve only as the basis for relationships and when these relationships extend beyond one level.

- Do NOT use DBKEYS when they are replacements for valid value lookups.
- Always use DBKEYS when the DBMS's physical structure REQUIRES the row to be deleted and re-added whenever the primary key or part of the primary key value is changed

Every database management system (DBMS) vendor regardless of data model and regardless of adherence to an ANSI standard (such as SQL) has invented its own internal storage structures for dictionary, indexes, relationships and data. In addition, every DBMS vendor has invented their own sets of access strategies, data loading utilities, and data update performance reactions. The DBMS-bound database design from the previous cycle is the source database design for this redesign cycle. In addition, the prototypes generated during the business policy based stage are upgraded to act as test cases for determining the performance characteristics of the specific DBMS on the specific platforms for client, server, or both portions. Specially configured prototype runs are created and executed to determine critical performance statistics. These statistics are then used to determine components of the application suite that perform sub optimally. Each sub optimal application area is examined to determine the type of improvements that can be made. These include the following:

- Buffer sizes and configuration for various storage structure components
- File size and table allocation to files and contained file pages
- File allocation to disk drives
- File allocation to disk channels
- Index design including multiple-column indexes, etc.

If all these types of changes fail to alleviate the performance problems, database design changes are in order. The database design change types are described above in the 1st Cut Performance section.

The final database design cycle relates to the environment on which the database is placed. The first set of iterations related to determining and incorporating basic database policy requirements. The second related to folding in the requirements for a specifically chosen DBMS, such as Oracle, Sybase, or DB/2. The third cycle of iterations relates to the installed database, that is, its physical design. As with the logical database design, there cannot be a change in the policy represented through the conceptual design. Further, there can be no compromise on the requirements of the DBMS's logical design. The physical database design cycles are:

- System control effects
- Client/server effects

System control is a term that collectively identifies the following aspects of database applications:

- Reorganization
- Backup And Recovery
- Multiple-Database Processing
- Concurrent Operations

Up until the advent of client/server computing database applications were run on a single computer and on-line access was provided through dumb terminals such as IBM's 3270s. Client/server computing, in contrast provide four alternatives for on-line access. These are:

- Dumb terminal access
- Down loading
- Down and Up Loading
- Cooperative Processing

These four types of client/server interaction require an increasing sophistication in end-user computing. The first, dumb terminal access merely requires a client that is able to access the server through a PC product such as CrossTalk, ProCom, or through a WAN/LAN telecommunications setup, or through the inter/ intra/net.

The second environment requires that the client be able to formulate queries and transmit them to the server for processing and then subsequent down-loading of the query results. The processing sophistication on the client might be restricted to a very simple ANSI/SQL DBMS with a client-to-server connection. Once the data is downloaded, it could be further processed through an on-line DBMS-based report writer.

The third environment is like the second except that data is able to be captured on the client through application logic and then updated to the server. The processing sophistication on the client has to be significant as screens need to be formatted, data acquired and edited, and then formulated into batch streams for uploading. The server then has to be accessed, connection made, and then the data presented to the server for processing against the main database. Once this is complete, the second part, down loading can occur.

The fourth environment, cooperative processing is like the third except that there is no batch processing. In actuality, programs do not execute, formulating batch-loadable files of data for uploading. Rather, a program starts execution; it connects to the server; and a dialogue ensues between the client, the client's end-user, and the server. This dialogue enables data to be entered, digested and/or edited on the client, transmitted to the server, wait until further response is provided by the server (that is, down-loads, or messages). Because of all this sophisticated interaction, client with server and vice versa, the computing environments on both sides need to be quite sophisticated.

5. DATABASE DESIGN ITERATION SUMMARY

The three major cycles of database design are: conceptual, logical, and physical. The conceptual cycle causes a database design to be created to accurately reflect the in-place business policy of the enterprise. The logical database design starts with the conceptual design and then incorporates the requirements of the particular DBMS through which the DBMS operates. The final cycle, physical begins with the logical database and incorporates the requirements of the physical environment. These environments range from monolithic mainframe through client/server to single user PC.

During the entire database design process the database design from each cycle is maintained in a repository. Done properly, each major cycle transformation (that is, conceptual to logical to physical) is expressible through SQL syntax changes. This way the re-transformation can always be accomplished.

The value of this process is obvious: never again will an organization have to re-gather and re-analyze requirements to deduce a database's design. Done once, done forever, providing that fundamental business policy doesn't change. Result? A saving of time and money, and a lowering of risk. Finally, because the designs are maintained in a repository, the effort to deploy another platform that may have a different DBMS and/or different operating system is much easier and simpler. Again, that saves time, money, and lowers risk.

*Michael M. Gorman, President of Whitemarsh Information Systems Corporation, has been involved in database and DBMS for almost **35** years. Mr. Gorman has been the Secretary of the ANSI Database Languages Committee, X3H2 for **25** years. X3H2 standardizes SQL. A full list of Whitemarsh's clients and products can be found on the web site, www.wiscorp.com. The goal of the web site, WisWeb, is to make data management books, courses, methodologies, software, and metrics available to the database community through electronic publishing and downloading. WisWeb memberships are very reasonable and are designed for the individual, the ISD organization, universities/colleges, and professional training organizations.*

[\[Return to the Article Archive\]](#)

The Data Administration Newsletter (TDAN.com)

Robert S. Seiner - Publisher - rseiner@tdan.com

© Copyright 1997-2003 - The Data Administration Newsletter (TDAN.com) - [Disclaimer](#)