Database Change Management

Although a cliché, it is true that change is the only constant in today's complex business environment. An ever-changing market causes businesses to have to continually adapt. Businesses are striving to meet constantly changing customer expectations while trying to sustain revenue growth and profitability at the same time. To keep pace, businesses must constantly update and enhance products and services to meet and exceed the offerings of competitors.

> *Change is the only constant in today's complex business environment.*

Moreover, the individuals within the business usually find it difficult to deal with change. Change usually implies additional roles and responsibilities that almost inevitably make our job more difficult. Our comfortable little status quo no longer exists. So, we have to change, too—either change aspects of our environment or our approach to doing things. There are many different aspects of managing change, particularly with respect to IT. Each of the following comprises a different facet of the "change management" experience.

- The physical environment or workplace changes to accommodate more employees, fewer employees, or perhaps just different employees with new and different skill sets.
- The organization changes such that "things" like processes or methodology have to adapt to facilitate a quicker pace for product and service delivery.
- The network infrastructure changes to provide support for a growing, and perhaps geographically dispersed, workforce.
- Applications and systems change to perform different processes with existing data or to include more or different types of data.
- The type and structure of data changes, requiring modifications to the underlying database schemata to accommodate the new data.

Change is inevitable but necessary for business survival and success. Many factors conspire to force us into changing our database structures, including

- Changes to application programs that require additional or modified data elements
- Performance modifications and tweaks to make database applications run faster
- Regulatory changes that mandate storing new types of data, or the same data for longer periods of time
- Changes to business practices, requiring new types of data
- Technological changes that enable databases to store new types of data and more data than ever before

> *Many factors conspire to force us into changing our database structures.*

Change will never disappear. Therefore, it is imperative that we have solutions to enable us to better manage these inevitable changes

Change Management Requirements

To successfully implement effective change management, understanding a set of basic requirements is essential. To ensure success, the following factors need to be incorporated into your change management discipline: proactivity, intelligence, analyses (planning and impact), automation, standardization, reliability, predictability, and quick and efficient delivery.

- Proactivity. Proactive change, which can eliminate future problems, is an organization's most valuable type of change. The earlier in the development cycle that required changes are identified and implemented, the lower the overall cost of the change will be.
- Intelligence. When implementing a change, every aspect of the change needs to be examined, because it could result in an unanticipated cost to the company. The impact of each change must be examined and incorporated into the change process, because a simple change in one area may cause a complex change in another area. Intelligence in the change management process often requires a thorough analysis that includes an efficient and low- risk implementation plan. True intelligence also requires the development of a contingency plan, should the change or set of changes not perform as projected.
- Planning analysis. Planning maximizes the effectiveness of change. A well-planned change saves time. It is always easier to do it right the first time than to do it again after the first change proves to be less than effective. An effective organization will have a thorough understanding of the impact of each change before allocating resources to implement the change.

A well-planned change saves time.

- Impact analysis. Comprehensive impact and risk analysis allows the organization to examine the entire problem, and the risk involved, to determine the best course of action. A single change usually can be accomplished in many different ways. However, the impact of each change may be considerably different. Some changes involve more risks: failure, undue difficulty, need for additional changes, downtime, and so on. All considerations are important when determining the best approach to implementing change.
- Automation. With limited resources and a growing workload, automating the change process serves to reduce human error and to eliminate more-menial tasks from overburdened staff.
- Standardization of procedure. Attrition, job promotions, and job changes require organizations to standardize processes to meet continued productivity levels. An organized and thoroughly documented approach to completing a task reduces the learning curve, as well as the training time.
- Reliable and predictable process. When creating any deliverable, a business needs to know that none of the invested effort is wasted. Because time is valuable, a high level of predictability will help to ensure continued success and profitability. Reliability and predictability are key factors in producing a consistently high-quality product.
- Availability. Most changes require downtime to implement the change. Applications must come down—the same is true of databases. However, high availability is required of most applications these days, especially for an e-business. This is fast becoming a requirement in the Internet age. Reducing the amount of downtime required to make a change will increase application availability.
- Quick and efficient delivery. Consumers demand quick turnaround for most products and services. Profitability is at its best when a product is first to market. Conversely, the cost of slow or inefficient delivery of products can be enormous. So, when implementing change, faster is better. The shorter the duration of an outage to accomplish the change, the quicker the system can be brought to market.

The Change Management Perspective of the DBA

The DBA is the custodian of database changes. However, the DBA is not usually the one to request a change; a programmer, application owner, or business user typically does that. There are times, though, when the DBA will request changes, for example, to address performance reasons or to utilize new features or technologies. At any rate, regardless of who requests the change, the DBA is charged with carrying out the database changes and ensuring that each change is performed successfully and with no impact on the rest of the database.

The DBA is the custodian of database changes.

To effectively make database changes, the DBA needs to consider each of the items discussed in the previous section: proactivity, intelligence, analyses (planning and impact), automation, standardization, reliability, predictability, and quick and efficient delivery. Without a robust, time- tested process that is designed to effect database changes, the DBA will encounter a very difficult job. Why?

Well, today's major DBMS products do not support fast and efficient database structure changes. Each DBMS provides differing levels of support for making changes to its databases, but none easily supports every type of change that might be required. One quick example: Most DBMSs today do not enable a column to be added easily to the middle of an existing row. To accomplish such a task, the DBA must drop the table and recreate it with the new column in the middle. But what about the data? When the table is dropped, the data is gone unless the DBA was wise enough to first unload the data. But what about the indexes on the table? Well, they too are dropped when the table is dropped, so unless the DBA knows this and recreates the indexes too, performance will suffer. The same is true for database security: When the table is dropped, all security for the table is also dropped. And this is but one example of a simple change that becomes difficult to implement and manage.

Adding to this dilemma is the fact that most organizations have at least two, and sometime more, copies of each database. At the very least, a test and production version will exist. But there may be multiple testing environments—for example, to support simultaneous development, quality assurance, unit testing, and integration testing. Each database change will need to be made to each of these copies, as well as, eventually, to the production copy. Furthermore, most organizations have multiple DBMS products, each with varying levels of support for making changes. So, you can see how database change can quickly monopolize a DBA's time.

Types of Changes

Managing change is a big component of the DBA's job. In fact, if systems and databases could be installed into an environment that never changed, most of the DBA's job would vanish. However, things change. Business changes usually necessitate a change to application code or to database structure. Less obvious business changes also impact the database—for example, when the business grows and more users are added, when additional data is stored, or

when transaction volume grows. Additionally, technological changes such as upgrades to the DBMS and changes to hardware components impact the functionality of database software and therefore require DBA involvement.

Business changes usually necessitate a change to application code or to database structure.

DBMS Software

The DBA must be prepared to manage the migration to new DBMS versions and releases. The complexity involved in moving from one version of a DBMS to another will depends on the new features and functions supported by the new version. Additional complexity will be introduced if features are removed from the DBMS in a later version, because databases and programs may need to change if the removed features were being used. Furthermore, as functionality is added to, and removed from, the DBMS, the DBA must create the proper policies and procedures for the proper use of each new DBMS feature.

Hardware Configuration

The DBMS may require hardware upgrades or configuration changes. The DBA will be expected to work in conjunction with the system programmers and administrators responsible for setting up and maintaining the hardware. At times the DBMS may require a different configuration than is commonly used, thereby requiring the DBA to communicate to the SA the reason why a nonstandard configuration is required.

Conversely, when hardware changes for other reasons, the DBMS configuration may have to change. Perhaps your organization is changing the disk drives in use with your database server hardware, or maybe adding additional memory to the box. Hardware changes such as these may require changes to database structures and the DBMS configuration. The DBA must be actively engaged with the SA team that configures and maintains the hardware used by the DBMS, and, as discussed earlier, may even have to function as an SA in addition to carrying out his DBA duties.

Logical and Physical Design

When the database changes, it is important that the blueprints that define the database also change. This means that you need to keep the conceptual and logical data models synchronized with the physical database. This can be accomplished in several ways.

When the database changes, the blueprints that define the database must also change.

Organizations adept in data administration may choose to make changes at the conceptual and logical levels first, and then migrate the changes into the physical database. Usually such an approach requires data modeling tools that segregate the logical and physical models. Furthermore, the procedures must facilitate the specification of changes at each level while providing the capability to synchronize the different models—both forward from logical to physical, and backward from physical to logical.

In the absence of robust data modeling tools, the typical approach taken to synchronize models is manual. Whenever the physical database is modified, the DBA must manually update the logical data model (and perhaps the conceptual data model). Such an effort requires a strict approach to change propagation. As with any manual change management scenario, this approach is tedious and error prone. However, it is imperative that the logical model be synchronized with the physical database. Failure to do so invalidates the usefulness of the data model as a blueprint for database development.

Applications

Application changes need to be synchronized with database changes; however, this is easier said than done. Whenever changes are made to a physical database structure, application changes usually accompany those changes. For example, simply adding a column to a database table requires application software to populate, modify, and report on the data in the new column.

Application changes need to be synchronized with database changes.

When the database change is migrated to a production environment, the application change must be migrated as well. Failure to do so will render the database change ineffective. Of course, the DBA could allow the database change to be migrated before the application change to ensure that the database structures are correctly specified. After changing the database in the production environment, the DBA would then inspect the database for accuracy and only then allow the application changes to be migrated.

But the relationship between the database change and the application change is valid. If the application change is backed off, the database change should be backed off, as well. And vice versa. Failure to synchronize changes will likely cause an application error or inefficiency. It is imperative that the DBA understands these relationships and monitors the change management process to ensure that the database and application changes happen in step with one another.

Physical Database Structures

The most complicated and time consuming type of change for DBAs is planning, analyzing, and implementing changes to physical database structures. But most databases change over time—indeed, the database that remains static once implemented is very rare. So the DBA must be prepared to make changes to the databases under his care. Some changes will be simple to implement, but others are very complex, error prone, and time consuming. The remainder of this chapter discusses how physical database objects can be changed and the problems that DBAs can expect to encounter in the process.

The most complicated type of change for DBAs is making changes to physical database structures.

Impact of Change on Database Structures

When the data requirements of your organization change, the databases used to store the data must also change. If the data is not reliable and available, the system does not serve the business—rather, it threatens the health of the business. So, we need infallible techniques to manage database changes. But even more, we need techniques that are not just fail-safe but also automated, efficient, and easy to use. Unfortunately, today's database systems do not make managing database change particularly easy.

Relational databases are created using Data Definition Language (DDL) statements. DDL consists of three SQL verbs: CREATE, DROP, and ALTER. The CREATE statement is used to create a database object initially, and the DROP statement is used to remove a database object from the system. The ALTER statement is used to make changes to database objects.

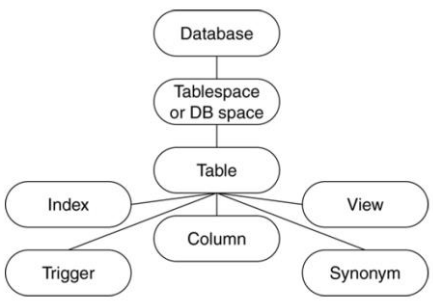
Not every aspect of a database object can be changed by using the ALTER statement. Some types of changes require the database object to be dropped and recreated with the new parameters. The exact specifications for what can, and cannot, be changed using ALTER differs from DBMS to DBMS.

For example, you can add columns to an existing table using the ALTER statement, but only at the end of the table. In other words, you cannot use ALTER to add a column between two existing columns. Additionally, you cannot remove columns from a table. To add a column anywhere but at the end of the column list, or to remove a column from a table, you must first drop the table and then recreate it with the desired changes. Every DBMS has limitations on what can be changed by using the ALTER statement. Furthermore, not just tables, but most database objects have certain aspects that cannot be changed using ALTER.

When making changes to a database requires an object to be dropped and recreated, the DBA must cope with the cascading DROP effect. A cascading DROP refers to the effect that occurs when a higher-level database object is dropped: All lower-level database objects are also dropped. (See Figure for a depiction of the database object hierarchy.) Thus, if you drop a database, all objects

defined in that database are also dropped. The cascading DROP effect complicates the job of changing a database schema.

Figure Database object hierarchy



The cascading DROP effect complicates the job of changing a database schema.

To understand the complexity involved, let's use an example. Suppose you are working with DB2, and you need to change a segmented tablespace to a partitioned tablespace. To do so, you must drop the segmented tablespace and recreate it as a partitioned tablespace. However, when you drop the segmented tablespace, you are also dropping any tables defined in the tablespace, as well as the table's columns and keys, all indexes defined on those tables, any triggers defined on the tables, any synonyms defined for the table, and all views that access the table. Furthermore, when a database object is dropped, the security information and database statistics are deleted from the system. The DBA must be able to capture all this information prior to dropping the tablespace so that it can be recreated after the partitioned tablespace is implemented. Capturing the DDL from the system catalog or dictionary and ensuring that the DDL is submitted correctly after the modification can be a tedious, complex, and error-prone process.

The system catalog or data dictionary stores the metadata about each database object. Metadata includes much more than just the physical characteristics of database objects. Additional information

about database objects such as security authorizations and database statistics are stored along with the metadata. All of the information required to recreate any database object is readily available if you know where to look for it, and part of the DBA's job is knowing where to look.

A final concern regarding database change: What happens to application program specifications? When database objects that are accessed by a program are dropped, the DBMS may invalidate that program. Depending on the DBMS and the type of program, additional steps may be required to rebind the application to the DBMS after the database objects accessed by the program have been recreated.

The Limitations of ALTER

Many types of database object alteration cannot be performed using the basic SQL ALTER statement; as usual, this varies from DBMS to DBMS and, indeed, from version to version of a single DBMS. However, the actions that are most likely to not be supported by ALTER include

- Changing the name of a database object (however, some objects— usually just tables—can be renamed using the RENAME statement)
- Moving a database object to another database
- Changing the number of tablespace partitions or data files
- Removing a partition from a partitioned tablespace or index
- Moving a table from one tablespace to another
- Rearranging the order of columns in a table
- Changing a column's data type and length
- Removing columns from a table
- Changing the definition of a primary key or a foreign key
- Adding a column to a table that cannot be null
- Adding or removing columns from a view
- Changing the SELECT statement on which the view is based
- Changing the columns of an index
- Changing whether an index is unique
- Changing whether an index is clustering
- Changing whether the index is ascending or descending
- Modifying the contents of a trigger
- Changing a hash key

In some limited cases, it is possible to use ALTER to change the length of certain types of columns. For example, in Oracle you can alter a character column to a larger size, but not to a smaller size. DB2 allows the length of a variable-length column to be changed to a larger size, but not, once again, to a smaller size. Additionally, it may be possible to change a column from one numeric data type to another. However, changes to the data type and length of a column usually require the table to be dropped and recreated with the new data type and length.

Making physical changes to actual database objects is merely one aspect of database change. Myriad tasks require the DBA to modify and migrate database structures. One daunting challenge is to keep test databases synchronized and available for application program testing. The DBA must develop robust procedures for creating new test environments by duplicating a master testing structure. Furthermore, the DBA may need to create scripts to set up

the database in a specific way before each test run. Once the scripts are created, they can be turned over to the application developers to run as needed.

> *Making physical changes to actual database objects is merely one aspect of database change.*

Another challenge is recovery from a database change that was improperly specified, or backing off a migration to a prior point in time. These tasks are much more complicated and require knowledge of the database environment both before and after the change or migration.

The preceding discussion justifies the purchase of a database change management tool to streamline and automate database change management. Keep in mind that the list of items above is not exhaustive and that it will differ from DBMS to DBMS. DBA tools exist that manage the change process and enable the DBA to simply point and click to specify a change. The tool then handles all of the details of how to make the change. Such a tool removes from the shoulders of the DBA the burden of ensuring that a change to a database object does not cause other implicit changes. Database change management tools provide:

- A reduction in the amount of time required to specify what needs to change
- A more simple and elegant method of analyzing the impact of database changes
- A reduction in technical knowledge needed to create, alter, and drop database objects
- Ability to track all changes over time
- An increase in application availability by reducing the time it takes to perform changes.

A database change management tool is one of the first tools acquired by most organizations when they implement a database of any size. Such tools reduce the amount of time, effort, and human error involved in managing database changes. The increase in speed and accuracy when using a change management tool provides an immediate return on the investment to the organization.

Database Change Scenarios

A DBA will need to make many different types of changes to a database over its lifetime. Some will be simple and easy to implement, other much more difficult and complex.

As discussed earlier, the SQL ALTER statement can be used to make many types of changes to databases. However, other types of changes may require additional steps to implement. It is the DBA's job to understand the best way to effect any type of database change. Keep in mind that simple changes often become more difficult in the real world. For example, a simple database change is not quite so simple when it needs to be propagated to multiple databases on different servers at multiple locations.

> *A simple change is not quite so simple when it has to be propagated to multiple databases on different servers at multiple locations.*

A single complex change, such as renaming a column, can take hours to implement manually. Changing the name of one column can require hundreds of changes to be scheduled, executed, and verified from development to test to production. Tackling such challenges is the job of the DBA.

Some Database Change Examples

Adding a new column to the end of a table is usually a very simple type of change. All that is required to implement the change is an ALTER statement such as:

```
ALTER TABLE Table_1

ADD COLUMN new column INTEGER NULL
```

> *Adding a new column to the end of a table is usually a simple type of change.*

The change can be accomplished in a straightforward manner by issuing a single SQL statement. It simply adds a new integer column to Table_1 that can be set to null. However, making the change once is easy, but keeping track of the change is a little more complex. Tracking database changes becomes more difficult as the number of database environments increases and the latency required between changes increases. In other words, a simple change that needs to be populated across twenty distinct database environments over a period of three months becomes more complex because the DBA must be able to track which environment has which changes. Furthermore, there will usually be multiple changes that need to be tracked.

A somewhat more difficult change is modifying the amount of free space for a database object. Such a change typically is accomplished by using an ALTER statement, but additional work is required after the ALTER statement has been issued. For example, consider the following ALTER statement:

```
ALTER TABLESPACE TS1 PCTFREE 25
```

Modifying the amount of free space for a database object is a more difficult change.

This statement changes the free space percentage for the tablespace named TS1 to 25% (from whatever value it was before). However, the additional free space does not magically appear after issuing this ALTER statement. In order to reclaim the free space for the tablespace, the DBA will have to reorganize the tablespace after successfully issuing the ALTER statement. Additional work is also required to ensure that sufficient disk space is available for the increased amount of free space. Therefore, the DBA needs to understand how each parameter that can be altered is actually impacted by the ALTER statement. Furthermore, the DBA needs to understand when additional work is required to fully implement the desired change.

Finally, let's examine a very difficult database change: adding a column to the middle of a table. Implementing such a change requires a lot of forethought and planning because it cannot be achieved using a simple ALTER statement. Instead, the table must be dropped and recreated with the new column in the appropriate place. The following steps need to be performed:

1. Retrieve the current definition of the table by querying the system catalog or data dictionary.
2. Retrieve the current definition of any views that specify the table by querying the system catalog or data dictionary.
3. Retrieve the current definition of any indexes defined on the table by querying the system catalog or data dictionary.
4. Retrieve the current definition of any triggers defined on the table by querying the system catalog or data dictionary.
5. Capture all referential constraints for the table and its related tables and determine what their impact will be if the table is dropped (causing all data in the table to be deleted).
6. Retrieve all security authorizations that have been granted for the table by querying the system catalog or data dictionary.
7. Obtain a list of all programs that access the table by using the system catalog, data dictionary, and any other program documentation at your disposal.
8. Unload the data in the table.
9. Drop the table, which in turn drops any views and indexes associated with the table, as well as invalidates any SQL statements against that table in any application programs.
10. Recreate the table with the new column by using the definition obtained from the system catalog.
11. Reload the table, using the unloaded data from step 8.
12. Recreate any referential constraints that may have been dropped.
13. Recreate any triggers, views, and indexes for the table.
14. Recreate the security authorizations captured in step 6.
15. Examine each application program to determine if changes are required for it to continue functioning appropriately.

Adding a column to the middle of a table is a very difficult change.

As you can plainly see, such a complex change requires diligent attention to detail to ensure that it is made correctly. The process is fraught with potential for human error and is very time consuming. In summary, to effectively enact database changes, DBAs must understand all of the intricate relationships between the databases they manage and have a firm understanding of the types of changes supported by the DBMS products they use.

Comparing Database Structures

When managing multiple database environments, the DBA may need to compare one environment to another. Usually changes are made to one database environment, say the test environment, as applications are built and tested. After the changes have been sufficiently tested, they will be promoted to the next environment, perhaps QA, for additional quality assurance testing. In order to appropriately migrate the required changes, the DBA must be able to identify all of the changes that were applied in the test environment.

One approach to change migration is for the DBA to keep records of each change and then duplicate the changes one by one in the new database environment. However, such an approach is likely to be inefficient. The DBA

could analyze a series of changes and condense them into a single change or perhaps a smaller group of changes, but once again, this approach is time consuming and error prone.

An alternative approach is to use a DBA tool to compare database components. All of the differences between the environments can be written to a report, or the tool can automatically replicate the structure of the database environment of record to another database environment. To accomplish this, the tool can compare the physical databases using the system catalog, data dictionary, or DDL scripts. A comparison tool is almost a requirement for a very complex database implementation because it is very difficult to track changes from one environment to the next. And the more environments that exist, the more difficult the change management becomes.

Use a DBA tool to compare database components.

If your organization does not have a database change management tool, be sure to save the DDL scripts used to create databases and keep them up-to-date. Every change made to the database must also be made to the DDL scripts. Bear in mind that subsequent ALTER statements can change the database, but will not change the DDL scripts. The DBA will need to either update the DDL scripts either by appending the ALTER statements to the appropriate DDL scripts or by changing the DDL scripts to reflect the effect of the ALTER statement. Both approaches are not ideal: for the first approach, changes may be required that cannot be implemented using ALTER (requiring you to modify the DDL script), and for the second the likelihood of introducing errors is high because a single change is made twice—once to the actual database and once to the saved DDL script.

If you do not store the DDL scripts for your database objects you will need to learn how to query the system catalog or data dictionary tables to recreate the database DDL manually. Both of these approaches, saving DDL and manually recreating DDL, are error prone and time consuming.

Without some type of comparison functionality the DBA must keep track of every single change and accurately record which environments have been changed and which have not. This too is an error- prone process. If the DBA does not keep accurate records then she will have to tediously examine the entire database structure for each database that may have changed using the system catalog or data dictionary in each database environment. Once again, this is also an error-prone and time consuming process.

Requesting Database Changes

The application development team generally requests changes to databases. The DBA is the custodian of the database, but is not the primary user of the database. Business users accessing data by means of application programs and systems tend to be the primary users of databases.

In order to properly coordinate database changes, the DBA group must institute policies and procedures governing how changes are to be requested and implemented. It is not reasonable to expect database changes to be implemented immediately, or even the same day. However, the DBA group should be held accountable to reasonable deadlines for implementing database changes. The DBA must examine each request to determine its impact on the database and on the applications accessing that database. Only after this information is evaluated can the database change be implemented.

Institute policies governing how changes are to be requested and implemented.

An application developer will request database changes only when those changes are viewed as required. In other words, the application has new data usage needs, and the database needs to be changed to support those needs. Of course, not every request will be implemented exactly as it is received. The DBA may need to modify requests based on his knowledge of the DBMS. Any deviations from the request must be discussed with the development team to ensure that they are still workable within the realm of the application.

Standardized Change Requests

The DBA group should establish standardized forms for implementing database changes. These forms should be customized for each shop, taking into account things such as environment, development expectations, knowledge, DBA experience, production workload, SLAs, platforms, DBMSs, and naming conventions.

Establish standardized forms for implementing database changes.

Standardized change request forms prevent miscommunications from occurring during the change management process and, if possible, should be implemented online. The form should include all pertinent information related to each change including, at a minimum, operating system, database subsystem or instance name, object owner, object name, object type, desired change, and date requested. The form should include sign-off boxes for those

personnel that are required to sign off on the change before it can be implemented. Required sign-offs should include at least the application development team leader and a senior DBA—but could also include a business unit representative, DA, or SA, depending on the nature of the request.

When the database change is completed, the form should be signed off by the DBA implementing the change and then sent back to the originators. The originators return the form to the DBAs with a requested date for implementing the change in production.

Checking the Checklists

Many DBAs develop checklists that they follow for each type of database change. These checklists may be incorporated into an online change request system so that the DBA can walk through changes as required. Additionally, many application development teams utilize checklists to ensure that every step that is required for an application to run correctly is taken and verified.

It is a good practice for the DBA group and the application development teams to share their checklists with each other to verify that each step in the database change and turnover process is completed successfully. Activities performed by DBAs, developers, and technical support personnel often overlap. Allowing the teams to review each other's checklists promotes a better understanding of what steps other units are performing during the change process. Many steps require intricate interaction and communication between the different teams. A formalized review process can correct errors before problems arise.

Sharing database-change checklists promotes database accuracy.

Communication

DBAs must provide education to development organizations on how to request a database change. They should provide guidance on accessing change request forms, instructions for completing the forms, and guidelines on service expectations.

Unrealistic service expectations, often the biggest problem, can be avoided through education. For example, if a requester understands that the DBA team could take up to two days to process his request, he will be able to work that delay into his timeline. Clarifying realistic service expectations on an organizational level prevents the DBA team from being deluged by "change this now" requests. These expectations need to be based on solid ground—performance requirements, availability, 24/7 issues, and so on. The DBA must ensure that reasonable timeframes—based on solid requirements— are built into policies and procedures that are readily available to all requesters. Only in this way does change management become a discipline.

Unrealistic service expectations can be avoided through education.

Summary

Databases are guaranteed to require changes over the course of their lifetime. The DBA is the custodian of the database and is therefore responsible for implementing change in a responsible manner that ensures the structure, integrity, and reliability of the database. The DBA must create and administer a database change management discipline consisting of tools, procedures, and policies to effect database change appropriately and responsibly.