

# MANAGING COMPLEX DATABASE CHANGES

**INTRODUCTION** Development can be a volatile endeavor—especially when sites or applications are database dependant and subject to a high degree of change and modification. While this is especially true of sites or applications involving multiple developers (or even multiple teams of developers working on different branches or features sets), even a small set of changes being added to an existing application can become a dicey task when database changes are involved.

As such, in cases where database changes need to be addressed, database comparison tools—such as Idera’s SQL comparison toolset—are indispensable. Therefore, what follows is a high-level overview of the benefits that database comparison tools provide—along with a list of steps you’ll want to use as a template for establishing your own deployment best practices in order to thoroughly test development changes before pushing them out into production.

## Setting the Stage

To really take advantage of properly pushing changes from actively developed applications or sites into production without causing undue stress you'll need a viable environment. First and foremost in creating this environment is the need to give developers their own copies of the production database that they'll be developing against. Obviously, some security considerations apply here - but we'll look at those in a bit more depth later on. But by giving developers their own copy of the production database to work against, they're free to pursue whatever functional changes they need to address without fear of compromising data or stability in production. In other words, the first step in creating a viable testing environment is to separate developers from production.

Next is the need to thoroughly test or vet developer changes before they're pushed into production. And this, in turn, requires a testing or staging environment that is also completely separate from your production environment (and from your development environment in most cases). In practice, this staging environment should be an almost exact duplicate of the systems you'll be using out in production because the purpose of this environment is to allow Q/A personnel and processes to detect any potential bugs, problems, or failures prior to pushing changes out into production (where they're much more expensive to deal with).

Therefore, to properly take advantage of database comparison tools and most effectively manage application changes involving databases you'll typically want multiple development environments (one per developer or even one per team) along with an additional staging environment—where all of these additional environments mirror production as faithfully as possible in terms of configuration, settings, and architecture.

## Order of Operations

With a functional development, staging, and production environment in place, you're ready to define your own workflows that will make testing and pushing development changes more successful and less traumatic. What follows is a high-level outline of the core concepts and considerations that you'll want to address in creating your own deployment workflows and process. Accordingly, feel free to use the next list of steps as a template, or as the basis, for your own deployment process—but just make sure that you take the time to carefully consider any specific needs your own environment or applications might require.

### 1. START WITH A DEVELOPMENT ZERO STEP

Developers should have an easy way to grab the latest copy of your production database—with all of the schema and relevant application data as it currently is out in production. To achieve this, nightly copies (or full backups) of your production database can be copied from production and put into a secured location for access by developers. With your zero-step backups, make sure to evaluate any security considerations for data at rest, and in some cases, you may even need an intermediate step that 'scrubs' or 'scrambles' sensitive data (such as user, medical, or other sensitive details) out of your zero-step backups after backups are made.

Then by equipping developers with simple scripts (or tools) to overwrite their current development databases with the most recent copy of what's out in production (or a 'scrubbed' version of what's out in production), developers are able to routinely test and retest their changes on an ongoing basis—rather than working for weeks (or even months) before trying to cram a big set of monolithic changes into production.

## **2. ENCOURAGE DEVELOPERS TO ROUTINELY SCRIPT DATABASE CHANGES**

With zero-step backups in play, developers should get into the habit of scripting changes they're making to the database. This way as they make changes, they can save their change scripts, and then routinely run them against a fresh copy of the production database to make sure everything is working correctly. For developers, this is a more agile approach to development—and more closely follows the paradigm of continuous integration. The side benefit of this approach, however, is that it encourages developers to be cognizant of the database changes they're making, rewards them for scripting their changes (instead of solely using the GUI to 'tweak' changes here and there), and helps them protect themselves against the pain of forgetting some minor changes here and there that they had to make in order to get changes or improvements to work. (Note too that I'm not advocating that developers NOT use the GUI—just that if they use it they don't hit the 'Save' button, but that they use the 'Script Changes' button instead.)

## **3. CAREFULLY EVALUATE DEVELOPER CHANGES**

When development changes are ready to be deployed into production, these changes first need to be vetted against your staging environment. Ideally, developers should be able to provide change scripts that cover all of the changes they need—and these can be reviewed (if they haven't already been so) and can serve as a 'map' of potential changes that will be pushed out, eventually, into production. However, in the fast-paced world of development, it's NEVER safe to assume that the only changes your database might encounter will always be faithfully covered by change scripts handed in by developers.

Therefore, by using a database comparison tool, such as Idera's SQL comparison toolset, the best practice is to compare the developer's test environment against the staging environment to get a complete enumeration of any and all changes that need to be pushed forward. Knowledge of these changes will serve as a better way to review code and other database changes and will also help make it easier to spot any potential performance or other problems that may not have been addressed during development.

## **4. SAVE A COPY OF THE CHANGES BETWEEN DEV AND STAGE**

Once you've finished comparing schema, constraints, defaults, indexes, sprocs, and everything else (along with, in many cases, core data in things like lookup tables or other application-specific tables), you'll want to save a copy of the script that your database comparison tool generates for use in synchronizing your staging environment (to make it look like your dev environment). Doing this is a critical step as it will serve as the basis for a number of auditing and verification steps further on.

## **5. PUSH CHANGES TO STAGING**

Once you've had time to compare the changes proposed by your developers with the actual changes defined in steps 3 and 4, you're ready to push any necessary changes into your staging environment. In ideal cases, you can just use the change scripts provided by your developers - but in other cases you need to use the 'synchronization' or change scripts provided by your database comparison tool instead. Either way, the goal of this phase of the change management process is to push the databases changes needed to support new features or functionality up into your staging environment where they can be adequately tested and verified by Q/A personnel and processes before being released into production.

## **6. GENERATE ROLLBACK SCRIPTS**

With your development changes pushed into staging, your production and staging environments are now out of sync—which you can use to your advantage with a database comparison tool in order to generate rollback scripts. In other words, since you've pushed a set of changes into staging in order to support any development changes planned, you can now run a comparison between your production database and the staging database to generate a set of 'synchronization' or change-scripts that can be used to make the staging database look just like the production database. This, in turn, results in a roll-back script that you can use in one of two ways. First of all, if there are any problems with dev changes that prevent them from going live, you can use your rollback script to 'back' the dev changes 'out' of your staging environment. (Just make sure to do an additional comparison between staging and production to make sure they're exactly the same once you're done backing the changes out). Likewise, if a nasty bug or issue somehow makes it past Q/A and testing and needs to be rolled back out of production, you can use your rollback scripts as well (just make sure to validate the logic in these scripts to make sure you won't encounter any side-effects of data-loss). As such, storing a copy of your rollback script in source-control or somewhere else (safe) is always a good idea.

## **7. DOUBLE-CHECK CHANGES PRIOR TO GOING LIVE**

This is where a database comparison tool like Idera's SQL comparison toolset really shines. Rather than merely pushing the same set of changes from dev to staging up from staging into production, you can compare your finalized, validated, staging environment with your production environment prior to pushing any changes. In this way, by comparing staging with production, you can generate a 'synchronization' or change script that you can use to finally push development changes into your production environment. However, by comparing this synchronization script with the synchronization (or change script) you used to push new functionality into your staging environment, you can ensure that no un-tested features, changes, or modifications are accidentally pushed into stage. In other words, if the change script you're going to be using to push from stage into production doesn't exactly mirror the change scripts you used to graduate code changes and features from development into staging, then somehow (somewhere) something was missed and you're going to be pushing untested functionality and code into production - so you'll want to abort, regroup, analyze where things went wrong, and try again. Hopefully you'll never encounter a problem at this stage in the game, but if you do, the ability to pre-empt the deployment of unintended and un-tested changes into production will make the cost of a database comparison tool more than worth it in terms of bypassed trauma and difficulty.

## **8. PUSH CHANGES INTO PRODUCTION**

With your rollback scripts at the ready (and in some cases you may even want to have an additional 'testing' environment or database that the DBA can use to test database changes AND rollbacks), you're now ready to push your development changes into production - which you can typically do by using the synchronization script generated by your database comparison tool. More importantly, if you've done everything correctly, your changes have been adequately tested and vetted within 2 different, viable, environments (that of your developers as well as within your staging environment). The benefit, of course, being that you're much less likely to encounter difficulties during your application or site upgrades. (Note: you may also want to push a copy of your change script, at this point, into source control for versioning and auditing purposes.)

## **9. UPDATE YOUR ZERO-STEP BACKUPS**

At this point, with your changes successfully pushed out to production, your developers will need to start again from ground-zero with any additional changes. So make sure to get a timely zero-step backup into place, and make sure that developers know to update their development environments accordingly. In some cases, recently deployed changes from other developers or teams will 100% break the changes that another developer or team is working on—but that's the goal, as it's much easier to address these problems on a regular (or continuously integrated) basis rather than blindly discovering these breaks days, weeks, or even months down the road.

**CONCLUSION** You only have to use a database comparison tool a few times to realize how powerful it is in terms of herding changes around from one database to another. But when you create a disciplined approach to using the strengths of database comparison tools to allow changes to 'graduate' from one environment to the next, you put in place a system of checks-and-balances that can make managing database changes much easier and much more trouble-free.

---

**HOW CAN IDERA HELP?** Idera's **SQL comparison toolset** is a set of products that perform object and data comparison, as well as synchronization. No need to purchase two separate products...get both in a single toolset! The tools are easy-to-use and can save hours of development time and make object and data comparison and synchronization quick and easy.

**Try SQL comparison toolset for free!**

**Download a free 14-day trial: [www.idera.com/Products/SQL-Server/SQL-comparison-toolset/](http://www.idera.com/Products/SQL-Server/SQL-comparison-toolset/)**

---

**ABOUT THE AUTHOR** Michael K. Campbell ([mike@sqlservervideos.com](mailto:mike@sqlservervideos.com)) is a consultant with years of SQL Server DBA and development experience. He spends most of his time engaged in consulting, technical evangelism, and creating free online SQL Server Videos.

For additional information or to download a 14-day evaluation of any Idera product, please visit: [www.idera.com](http://www.idera.com).

#### ABOUT IDERA

Idera provides tools for Microsoft SQL Server, SharePoint and PowerShell management and administration. Our products provide solutions for performance monitoring, backup and recovery, security and auditing and PowerShell scripting. Headquartered in Houston, Texas, Idera is a Microsoft Gold Partner and has over 5,000 customers worldwide. For more information, or to download a free 14-day full-functional evaluation copy of any of Idera's tools for SQL Server, SharePoint or PowerShell, please visit [www.idera.com](http://www.idera.com).

---

Idera | BBS Technologies | 802 Lovett Blvd Houston, Texas 77006  
PHONE 713.523.4433 | TOLL FREE 877.GO.IDERA (464.3372)  
FAX 713.862.5210 | WEB [www.idera.com](http://www.idera.com)

