



WHITE PAPER

The Agile Database: Best Practices



Database Continuous Delivery: Agile Database Best Practices

Competition in the marketplace is getting fiercer every day, particularly in today's application economy where competitors are, literally, only a swipe away. On the level playing field created by the internet, large enterprises are finding that they must figure out a way to compete against young upstarts in this new digital era.

On the one hand, tech startups aren't encumbered by legacy applications or the inertia that's generated over decades of operating experience, and are able to benefit from implementing the newest technologies and practices from the get go. On the other hand, large enterprises have the kind of brand cache that only comes from the formation of deep and lasting relationships with their customer base, and they have considerably more resources to bring to bear in defending their turf.

In the adoption of modern practices like Agile and Continuous Delivery, the large-scale enterprise can enjoy the best of both worlds by tapping into their more considerable resource base to match tactics against younger, more nimble companies. And it looks as though more and more business leaders are, in fact, recognizing this opportunity to bring parity back to their competitive endeavors. In their "8th Annual State of Agile Survey" published in 2014, VersionOne surveyed over 3,500 IT professionals, three-quarters of whom worked in software organizations ranging from 100 to 1,000 employees, and found that adoption of Agile across large scale enterprises is on the rise.

Although the benefits of Agile are numerous, the development methodology presents a unique challenge for the DBA team.

"The number of teams practicing agile at each organization continues to grow, and **this year there is evidence that it's growing fastest at the enterprise level**. 57% of respondents said their companies had adopted agile practices across 5 or more teams. This number has nearly doubled in the last 2 years (48% in 2012 and 33% in 2011). The fastest-growing group is those who practice agile in 10 or more teams (38% of respondents), which is an 8% increase over 2012."¹

Borne out of a business need, Agile is steadily marching towards becoming the development practice of choice among large-scale enterprises. When queried about why they chose to adopt Agile, the VersionOne survey found that the top three reasons were to "accelerate time to market (23%), more easily manage changing priorities (16%), and to better align IT and business objectives (15%)." And Agile has delivered on its promises, with respondents

¹ VersionOne (2014) The 8th Annual State of Agile Survey



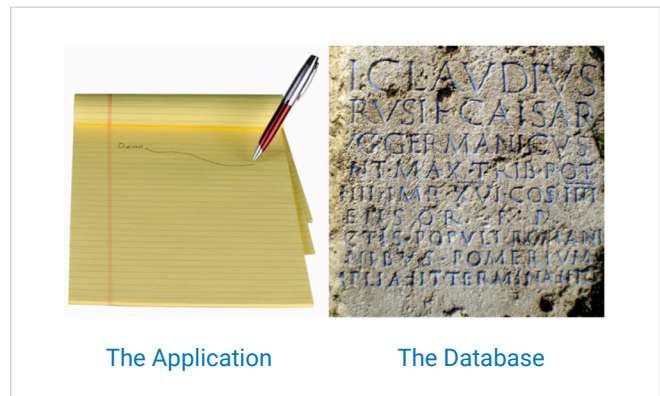
reporting that “the top business benefits were the ability to manage changing priorities (92%), productivity (87%) and project visibility (86%),” in a survey question allowing for multiple responses. Although not in the top three, 83% of respondents reported that they did experience faster time to market, and 82% reported having found better alignment between IT and the objectives of the business.

Agile and the Database: A Unique Challenge

An Unforgiving Platform

Although the benefits of Agile are numerous, the development methodology presents a unique challenge for the DBA team, which stems from the inherent differences between application and database platforms. Relational databases possess a persistent nature that’s due to the precious data stored inside, and cannot simply be wiped out and replaced with new code if a mistake is found along the promotion path towards production.

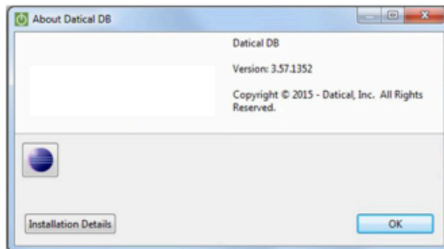
This characteristic makes the database a particularly unforgiving platform, because once you update the production database, that database is set in stone, and you’re very likely to lose data if you try to revert it. Whereas experimentation is encouraged in Agile development, because code is easily replaceable, the database needs to be developed much more deliberately, and with greater caution, owing to the fact that it’s much harder to roll things back in the database in the event of a failure.



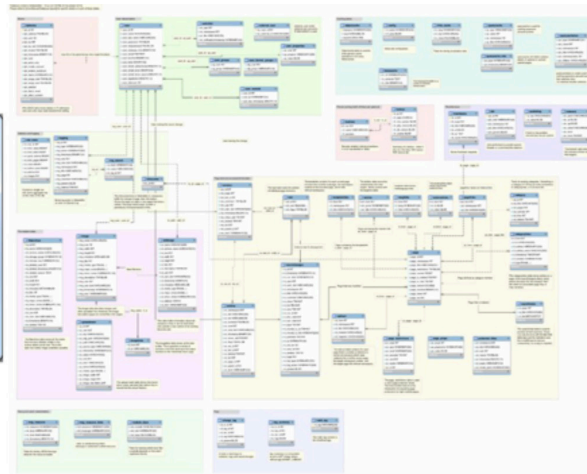
Difficult to Version

Another area of difference between the database and the application which presents a unique challenge to DBA teams is the ability to determine the version of a particular instance. Applications make the version number readily available to help identify their current state, but unless the DBA team is extremely conscientious about how they go about updating the database, the state of the database is much more difficult to ascertain. To do so requires some detective work and manual inspection of database objects to determine what changes were implemented to support the latest application release.

Although tedious, it’s important for DBA teams to have an understanding of what state the database is currently at, so that they can effectively determine what scripts need to be run in order to evolve the database to support the next release, or to test a new feature. The challenge is that although necessary, this manual detective work can significantly impede release velocity in an Agile environment.



The Application



The Database

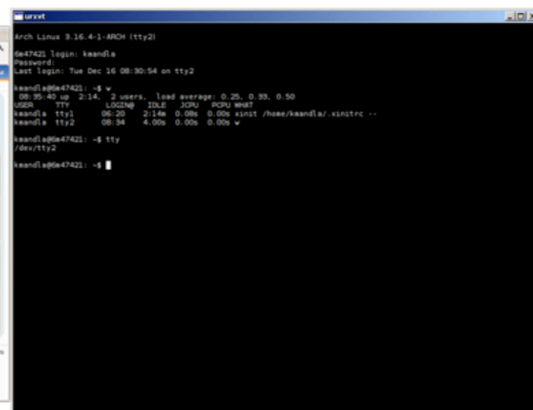
Lack of Safe Automation

Compounding the challenges for DBA teams who support Agile workflows is a general lack of availability of database automation tooling. When we engage with prospects, they often report that, on a scale of 1-10, their application processes are automated to an 8 or 9, while the database is a 2. This isn't a criticism of the DBA team, it's just that there aren't many automation tools available for the database. In the case where a team does use a tool to help with deploying database changes, the tool often amounts to little more than automating the execution of SQL scripts.

There just isn't the same degree of tooling and safe automation available for the database as there is for supporting tasks like static analysis, unit testing, or tying in to release automation for the application code. This state of affairs leaves database deployments still very much decoupled from the process of updating the application, which inevitably introduces risk when deploying database changes.



The Application



The Database



Where Did This Technology Gap Come From?

The adoption of Agile in the early 2000's, primarily by developers, spurred innovation in tooling designed to support Agile workflows. This innovation was first seen in the area of test automation as a way to increase developer productivity, and actually stems from one of the four main values set forth in the Agile Manifesto, which values "working software over comprehensive documentation." This value guides one of the core principles in Agile development, which is that a product or service should always be in a "releasable" state, meaning that there should always be a functioning, stable package of code which can be delivered to the customer.

For Dev, satisfying this value meant one of two paths could be taken. Either the development team could allocate resources to constantly test, fix, and retest new code as it was created, or development could bake testing in from the very beginning.

The former approach is just not economical, and so Dev embraced the latter, which led to test-driven development practices. In today's Agile teams, developers write tests based on requirements first, and only after this is complete do they begin work on the code itself. These tests are stored on local machines or integration servers, and are triggered every time a new build occurs, or in the case of Continuous Integration, whenever new code is committed to the source control repository. When a developer's code "breaks the build," the expectation on an Agile team is that he or she will immediately rectify the situation and recommit the code.

When a development team is first transitioning to Agile, developers frequently push back against this "test first" attitude, feeling that it consumes too much of their time on what they consider to be a low value activity. But the reality is actually that writing tests first is a very high value activity for the team, and by extension the business itself. At the Equity conference in 2007, Barry Boehm, an IT researcher from the University of Southern California, shared a chart which describes the relative cost of fixing software bugs as a release is promoted through each stage along the path to production.² Essentially, what Boehm's research proves is that the earlier an issue is discovered in the SDLC, the less it costs the development team in terms of time to fix the issue, and by extension the less it costs the business which funds the development team's activity.

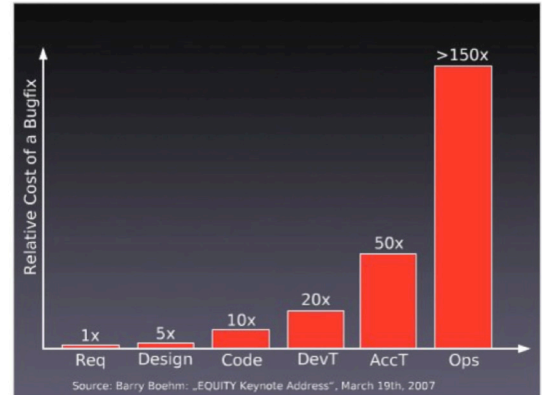
For these economic reasons, and in support of Agile's core values, early Agile teams embraced test-driven development. But testing does take a considerable amount of time, which creates tension between the stated value of "working software" and one of the manifesto's underlying principles, which states, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

² Boehm, B. (2007). EQUITY Keynote Address. Speech presented at the IEEE International Conference on Exploring Quantifiable IT Yields, Amsterdam, The Netherlands.



To reconcile this tension, development started looking for ways to automate the process of testing, which is a tedious and manual task, so they could free up more time to focus on delivering value to the customer, to whom the act of testing is transparent. This activity led to innovation in test automation tooling, which in turn spawned the desire to identify and automate other manual, tedious tasks that are necessary to the delivery of working, stable software. Essentially, automation became a habit in Agile development, and over the intervening years tooling became more and more abstracted, all in the pursuit of automating routine tasks so that Agile teams could focus on delivering value to the customer. Test automation led to code quality tools, which led to continuous build and integration servers, which led to even more abstract concepts like release automation and Continuous Delivery.

Best Practices in Agile Database Development

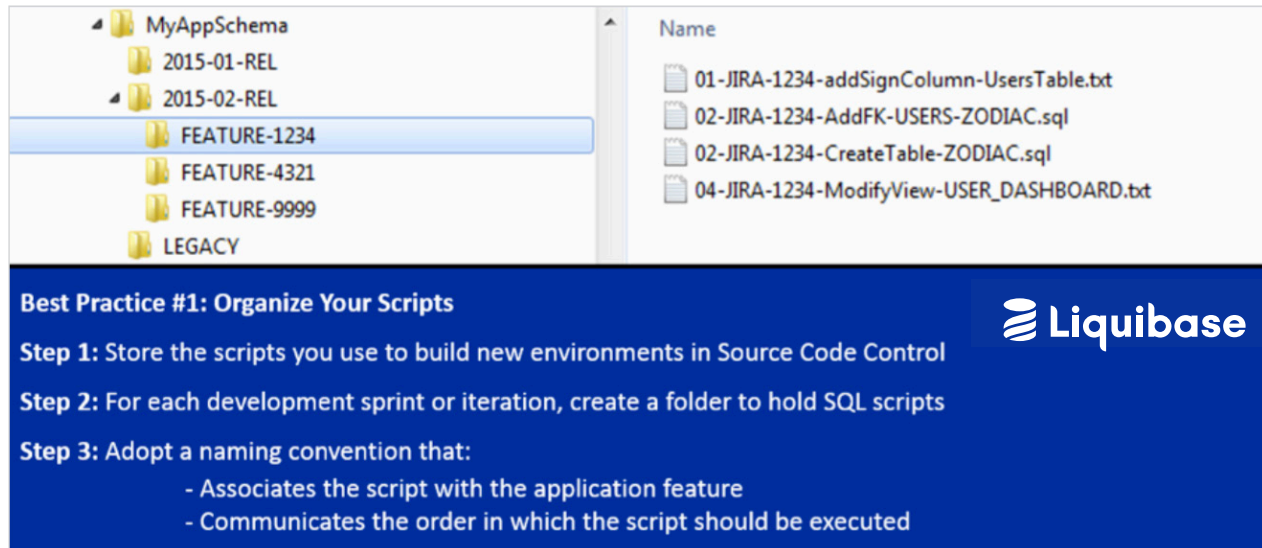


To assist database teams facing the onslaught of changes coming out of Agile development teams, we've compiled a list of best practices in Agile database development to share with you. These practices have been culled from conversations with customers, prospects, and industry experts over the past few years, and are designed to help DBA teams better support Agile workflows.

Best Practice #1: Organize Your Scripts

This one may seem a bit pedestrian, but it's an important concept for database professionals to embrace, as it will allow you to mirror similar practices used in Agile development which organize application changes around specific releases. The first step is to begin committing the scripts you use to build new environments within the same Source Code Control System that development is using. This creates alignment between the changes associated with a particular application release and the changes to the database schema that support those application modifications. Organizing changes in this manner provides granular visibility around the context of change, so that DBAs are quickly able to identify which changes were made when, and for what reasons.

A good rule of thumb when organizing your scripts is to keep them as atomic as possible, meaning that each script should only contain exactly what it needs to effect a single change. For example, if an application change requires a new database table, you should store all of the logic required to create that table (create table, ID primary key, setup index, etc.) in a single script.



The next step is to organize your scripts in a folder structure that mirrors the specific development sprint you're supporting. When storing those scripts, you should follow the same directory structure used for the application, i.e. **Project -> Release -> Feature**, as depicted in the screenshot above.

The last step is to adopt a naming convention for both scripts and folders that associates each script with the application feature being supported, and that clearly communicates the order in which the script should be executed. Doing so not only facilitates the deployment of database changes, but also provides some common ground for the database team as they coordinate and collaborate with Dev, QA, and Ops.

Best Practice #2: Version Your Instances

Organizing your scripts in the manner prescribed above allows you to begin versioning your instances. This is extremely important for the DBA team supporting Agile development, as understanding the current state of the database helps you determine what you need to execute in order to evolve the database to where it needs to go.

Start by creating a table or spreadsheet tracker that marks every script that has passed. Another option is to automate this process by building logic within each script so that it marks its own passing and updates the table accordingly, recording the WHO, WHAT, WHEN, and WHY of each script that is executed against the database.

The next step is to "tag" each script in the database to the release it supports. This facilitates lookup later on when you're trying to determine which changes were part of which release, and makes troubleshooting efforts easier. Once this system is in place, you can begin to compare what changes have already been run in the database against what changes still need to be executed, which allows you to create release packages for database changes on a per instance basis.




Best Practice #3: Automate Your Database Deployments

Best practice #1 and #2 will take some time, effort and patience to enact, and will likely require some reorganization in the way a database team thinks about the way they work. But together they bring a level of sophistication to database development that enables the team to begin automating their database deployments. Automation is the biggest mental leap the team will have to take, and although it can create some trepidation, it is also the step which will provide the team the most benefits, as well as pay the most dividends moving forward.

FILENAME	AUTHOR	DATEEXECUTED	DESCRIPTION	TAG
1 2015-01-Release/01-JIRA-1234-addSignColumn-UsersTable.sql	Pete Pickerill	15-JAN-15 11.56...	addColumn	2015-02-RELEASE
2 2015-01-Release/02-JIRA-1234-CreateTable-ZODIAC.sql	Carey Bengel	07-JAN-15 02.43...	sql	2015-02-RELEASE
3 2015-01-Release/03-JIRA-1234-AddFK-USERS-ZODIAC.sql	Carey Bengel	07-JAN-15 02.43...	sql	2015-02-RELEASE
4 2015-01-Release/04-JIRA-1234-ModifyView-USER_DASHBOARD.sql	Carey Bengel	07-JAN-15 02.43...	sql	2015-02-RELEASE
5 2015-01-Release/01-JIRA-4321-addSeasonalDiscountColumn.sql	Carey Bengel	07-JAN-15 02.43...	sql	2015-02-RELEASE
6 2015-01-Release/01-JIRA-4434-CreateFunction-f_find_sign.sql	Pete	07-JAN-15 02.43...	createFunction	2015-02-RELEASE
7 2015-01-Release/01-JIRA-3323-DropTable-ChristmasWishList.sql	Pete	07-JAN-15 02.43...	dropTable	2015-02-RELEASE

Best Practice #2: Version Your Instances

 **Liquibase**

Every script marks its passing

- Statements added to the end of each script or a Stored Procedure written to record Who, What, When, and Why
- Major releases are “tagged” in the database

Compare what has run to what should be run to determine the release package

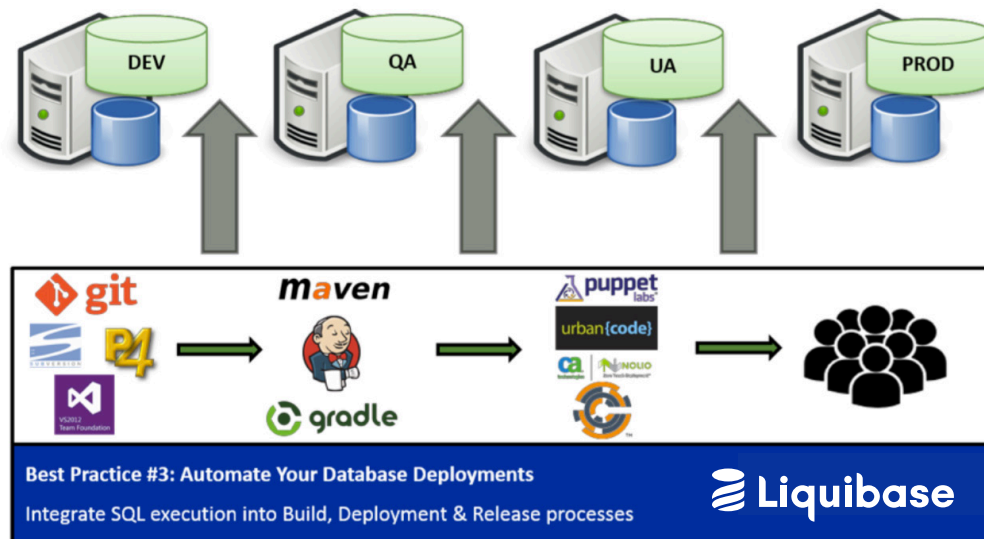
- Find omissions from previous feature releases
- Quickly determine major version on a per instance basis

Once you have your database changes organized, stored in Source Code Control, and have laid out a path to determining what changes are required for each major/minor application release, database deployments become an eligible candidate to include in the same automation process used by the application.

The argument for automation is that, once implemented, it brings a level of standardization and repeatability to your database deployments that will not only make the lives of the database team easier, but will also help to improve the performance of the entire application delivery process. By following a very structured, regimented process for updating the database as it migrates along the path to production, your environments will become more in sync over time, eventually eliminating the configuration drift that comes from making out-of-process changes. When your environments are in sync with each other, you benefit from testing environments in Dev and QA that are much closer to what the production environment looks like. This allows you to identify relevant issues much further left in the deployment process, making them cheaper and easier to resolve, as demonstrated by Boehm’s research.



Liquibase Enterprise was Built for Agile



Liquibase's philosophy is that the best method for mitigating deployment risk, reducing costs, and increasing velocity is to keep database changes in sync with the application code as they are promoted through environments. Built upon Liquibase, the leading open source tool for database source control, Liquibase Enterprise stores your database changesets in the exact same repository where your application code resides. This application-centric approach creates a single source of truth for tracking and managing database changes through your release cycle. It comes fully integrated with Git, Subversion, and Microsoft Team Foundation Server (TFS), and supports extensibility to integrate with other source control systems.

Bringing an existing database under source control is incredibly easy using Liquibase Enterprise. Simply baseline the existing database to bring it under management, and Liquibase's powerful data model will abstract the structure of the database, keeping intact the history of how the database has evolved over time. Then Liquibase Enterprise stores this baseline in your source control system, providing a starting point for tracking and managing changes.

As you continue to evolve the database, Liquibase Enterprise maintains the connection between database changes and application code. When crafting new database changes, the changeset wizard in the Liquibase Enterprise GUI guides you through best practices for source control management, requiring the assignment of an author and changeset ID for each change in addition to allowing for tags and comments each time you commit. This rich history of the evolution of database objects, when combined with Liquibase Enterprise's robust reporting features, eases the strain of database audits by facilitating rapid lookup of changes that need to be investigated and providing detail around who changed what, when, and why.



Liquibase Enterprise provides an advanced status feature that allows you to instantly ascertain the state of the database at any given point in time, so you no longer have to wonder about what changes have already been applied and which still need to be executed. The data model makes Liquibase Enterprise database agnostic, with full support for Oracle, DB2, SQL Server, MySQL, and Post-greSQL, and the capability to extend into other RDBMS, providing the internal logic to automatically migrate changes between disparate databases. To support your unique release cycle, Liquibase Enterprise allows you to define your own workflow for promoting changes through Liquibase's Deployment Plan feature, providing complete visibility into your process via a single dashboard from which you can track and manage your process for updating the database from development to production.

Project organization carries this a step further, allowing you to define Deployment Plans for all of the applications you support, in addition to making it incredibly easy to support branching and merging of database changes to support Agile development practices.

What's more, the advanced feature sets in Liquibase Enterprise allow room to grow as you pursue more advanced techniques like Continuous Delivery, protecting your technology investment and increasing your ROI. Liquibase's Forecast feature simulates proposed changes against an in-memory model of the target database prior to deploying, virtually eliminating the risk of a failed deployment. Taking Forecast a step further, the Liquibase Rules Engine allows you to customize rule sets for validating your database changes, enforcing corporate policy and your organization's best practices. Fully certified integrations with popular Application Release Automation (ARA) suites like IBM UrbanCode Deploy, CA Release Automation, and Serena Deployment Manager enable a fully automated deployment pipeline for your database changes.

About Liquibase

Liquibase accelerates database application development and schema change management across the software development lifecycle for organizations that practice Agile or DevOps. While there has been a proliferation of application lifecycle management tools to help teams develop and deliver application code faster, innovations for the database have lagged behind, creating a bottleneck in the develop-test-deploy process. Liquibase uniquely solves this problem by managing database changes in step with the application code as they are promoted through environments, eliminating the need to track and manage SQL scripts.

Liquibase's flagship product, Liquibase Enterprise, is built upon the leading open source solution for database versioning and source control, extending Liquibase to achieve enterprise level scalability, reliability, traceability, and ease of use. Value added functionality includes support for programmable objects, DBMS specific syntax, documentation for audit compliance, and the ability to model the impact of changes before deploying to an environment. A lightweight architecture and tight integrations with release automation tools lead to smooth implementations, increased adoption and accelerated time-to-value for development organizations. For the business, Liquibase Enterprise delivers outstanding ROI by increasing velocity, reducing cost and risk throughout the release cycle, and accelerating time to market.



www.liquibase.com | info@liquibase.com