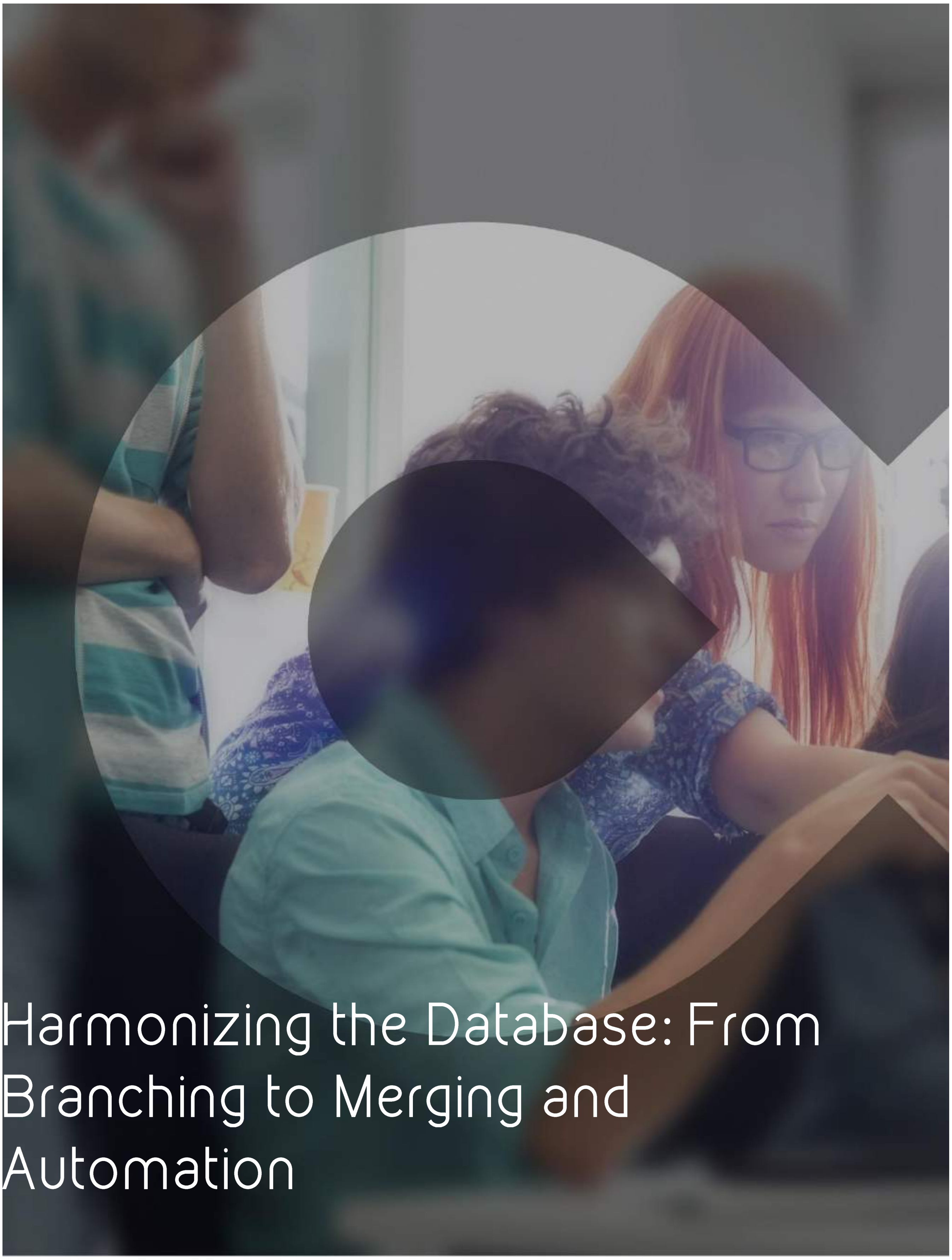


Harmonizing the Database: From Branching to Merging and Automation

DBmaestro
DevOps for Database

Executive Summary- Methodology and Merges

Agile development is a major facet of an increasing number of modern organizations. Consumer demand for instant gratification and top-notch service means developers are under constant pressure to release bug-free applications and make continuous updates and improvements. Development teams are constantly juggling multiple projects. With these demands comes the need for a logical framework to organize and manage various development efforts, whether for multiple projects or for various components of the same application. Enter branch methodologies.

This white paper discusses the various branch methodologies and merge scenarios used in development, and the importance of using the same methodology for both application and database development. Using mixed methodologies between application development and database development creates unnecessary challenges, adding both time and complexity to the already-rushed, already-complex development lifecycle. In today's climate, the ability to automate development, and merge and deploy changes rapidly – without introducing serious mistakes – relies on using the same branch methodology for database development as you use for application development.

Understanding Application Development Branches

Source control plays a vital role in software development; so vital that you will rarely encounter a software project not relying on source control for the code. It quickly became a de facto basic standard in the industry. Today, with Agile and DevOps teams often spread across several continents, application releases and deliveries are expected to be faster than ever before. These teams therefore need a way to support their parallel development and maintain source control.

The Role of Branching

Without the capability to work on two or more distinct lines of development at the same time, it is next to impossible to successfully release maintenance updates and bug fixes for version 1.0 without inadvertently releasing developments-in-progress to your yet-unreleased version 2.0. It is critical, therefore, that both of these processes can be maintained, managed, and developed without the risk of interference from one another.

The primary method for development teams to simultaneously work on different elements of an application, or on the same elements for different versions, is known as branching. The most complex aspect of branching, however, is not keeping development tracks separate; rather, it is merging changes originating from different branches. Source control tools provide this crucial capability. As part of the merge process, source control technology can analyze the objects being saved (text file, image, document, presentation, database object, etc.) and determine if identified differences should be merged into a new baseline source code or not (as illustrated here https://www.flickr.com/photos/jawspeak/5153884916).

Branching Methodologies Explained

There are multiple methodologies for conceptualizing how a particular project will be branched. Atlassian, a successful enterprise software company, describes three primary branching methodologies used by Agile development teams:

- **Release Branching** – Each development release is contained entirely within its own branch. This methodology is an important part of supporting versioned software which is already on the market, in order to differentiate maintenance for each version in its own distinct line of development.

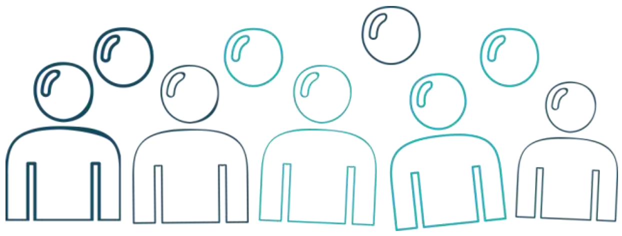
- **Feature Branching** – All the code changes for a specific feature are isolated in a distinct branch, for the purposes of long-term development. Considered a more flexible methodology by some developers, this method ensures that each change to a given feature is fully developed, tested, and validated separately before being merged with a master branch or trunk. Among the benefits of Feature Branching is the ability to selectively enable or disable specific features at any point in the development and deployment cycle.

- **Task/Bug/hotfix Branching** – Also known as Issue Branching, each software issue or coding bug is addressed on its own branch. Identifying branch files by the issue they address adds a level of transparency and facilitates team communication, making it easy to determine which issues are at what stage.

There are also other branching methodologies:

- **Trunk Based Development** – Containing elements of both Release and Feature Branching, each release is recognized as its own branch and each branch contains a replica of the trunk. However, only specific personnel are authorized to merge changes into the release branches.

- **Team Branching** – Distinct branches are created for each development team, especially for teams that have multiple projects, feature groups, or distinct areas of functionality. While the team branch is maintained by multiple developers, it may be divided further into feature or task branches, for example, creating a multi-tiered hub-and-spoke design to your source control.



These five methodologies are not all-inclusive, nor is there a “right” branching methodology. However, some methods are, in fact, better suited for certain development situations or companies. A small company that makes changes infrequently and has a small development team, for example, may find less need for release branches than a major enterprise managing multiple versions and changes several times a week.

The most critical (and nerve-wracking) part of branching is actually the merge event. Therefore, as each branching methodology has its own variation of merging and rebasing, a key consideration must be how features and changes will be best merged and deployed down the line.

Branching Methodology	When the Branch is Created	When to Merge	Rebasing (why and when)
Release Branching	1) When a version is being released 2) When an issue needs to be fixed in a previous release	When the issue was fixed	There is no need to synchronize the most up-to-date code to this branch
Feature Branching	Upon beginning the development of a feature	When a feature is ready for tests	When there is a need to merge changes made in different branches. The database rebase is completed while the application code is being rebased.
Task/Bug/Hotfix Branching	Upon beginning a development task	When a task is ready for tests	When there is a need to merge changes made in different branches. The database rebase is completed while the application code is being rebased.
Trunk Based Development	When a new release is created	As determined by Merge & Build Manager	When there is a need to merge changes made in different branches, because of application dependencies. The database rebase is completed while the application code is being rebased.
Team Branching	Upon the beginning of the development phase	When the development is ready for tests	When there is a need to merge changes made in different teams' branches, because of application dependencies. The database rebase is completed while the application code is being rebased.

How Database Development Fits into the Picture

It's not uncommon for database development teams to work today without branching, which can completely derail application releases. If the application development team uses, say, a Feature Branching methodology, but the database development team doesn't branch at all, then determining which database changes must be merged with which new application feature or bug fix becomes a daunting, time-consuming and error-prone task. It will be problematic for end users and new features won't work properly, if at all.

It simply doesn't make sense.

So why would an organization develop their database without a branch methodology?

- ***Creating the branch*** – Branching databases is not as simple as branching application code. While branching application code is as simple as copying files into a new directory, database branching may require additional storage, new schemas, and cloning several instances or full databases.

- ***Merging the branches*** – Once a database branch environment is created, you need to be 100% confident of its eventual merge into the application. However, file-based source control tools work with scripts, and they lack the ability to directly read the metadata of the merged environment (usually the trunk) from the database. This makes a merge process using the all-important baseline analysis unpredictable.

Yet, since you need to update the database structure to support the application, it is preferable to develop the database in parallel with the application code. For example, you cannot complete application development that accesses a new column without first adding the column in the database. This strong linkage requires both facets of development to use the same branching methodology.

Don't Rely on Mixed Methodologies

Using mixed branching methodologies in one application is a recipe for disaster, as there are distinct merge procedures for each of the five possible development methodologies. Therefore, in order to ensure that a complete and coherent release reaches your end users, the application and database branchings need to be congruent. Ignoring that limitation would be like building a house mixing inches and centimeters, or baking a cake using pounds and kilograms – a recipe for disaster.

For example, if you use the Task Branching methodology, then you must have a parallel task branch for both the application and the database; otherwise, determining where database changes should be merged becomes a time-consuming, tedious and risky effort. Likewise, using Trunk Branching for the database when application development relies on Release Branching means that database and application code changes do not coincide, for an unnecessarily messy, cumbersome, and undependable merge process.

Database Branching and Baseline-Aware Merging

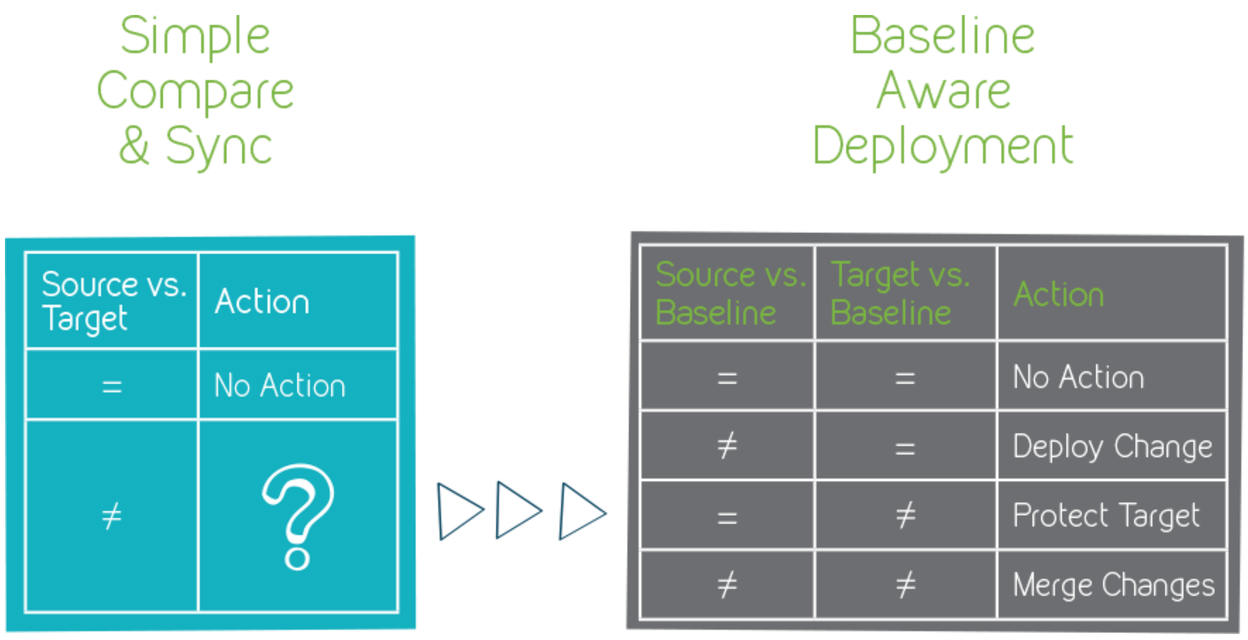
Using the same framework for database development as for application development streamlines your merge process, makes it easy to detect changes that should not be merged, facilitates a seamless workflow, and enables better collaboration between application and database development teams. With that in mind, the method chosen to branch a database development environment and merge database changes should be carefully chosen.

There are several ways to branch a database environment:

- Create an empty schema/database and use an appropriate source control solution to create the objects and fill the static data.
- Create a backup or snapshot of the database for use as the starting point for the branch.
- Use a storage feature to quickly clone database pages.
- Use commercial tools such as Delphix and Actifio to create the branches.

Regarding the merging process, there is one question the infrastructure should address: What is the origin of identified coding or database differences? If that cannot be determined, then you may check in the wrong code or generate errors in the merged environment.

The best method of identifying the source of development changes is baseline-aware analysis. For this kind of analysis, the source control tool compares the content of a given object in three revisions: the current branch revision; the trunk revision; and the branch baseline revision. Then, the tool can recommend an action - merge, ignore, or manually resolve an environment conflict.



Source vs. Target	Action
=	No Action
≠	?

Source vs. Baseline	Target vs. Baseline	Action
=	=	No Action
≠	=	Deploy Change
=	≠	Protect Target
≠	≠	Merge Changes

In contrast, standard compare-and-sync tools fall short of identifying the source of differences between the source and the target environments. This capability, however, is the key ingredient necessary for true deployment automation. As a result, 70% of respondents in the 'Database Development and Deployment Risks Survey Report' using compare-and-sync tools said they have to manually review and correct sync results, as their tools cannot be trusted to automatically deploy correctly.

Baseline-aware analysis provides the critical insights needed to facilitate trusted deployment automation.

Merging Using the Key Branching Methodologies

With the insights of baseline-aware analysis, you can turn to the process of merging database environments. For this purpose, you need:

- A database environment into which you can merge the changes. This database environment is required not only for source control, but also for running integration tests.
- An environment for the branch (this can be a label on the database source control).
- A way to support analysis for selective objects (to support a feature branch).
- A solution that can directly access the metadata from source control and from the database.
- A baseline-aware analysis, rather than a simple compare-and-sync.

Once you have those requirements you can merge the database environments just as you merge the application code. For example:

Release Branching – Changes made in the release branch support a specific release and are merged as needed. Baseline-aware analysis will identify conflicts and ignore objects that were updated for separate releases.

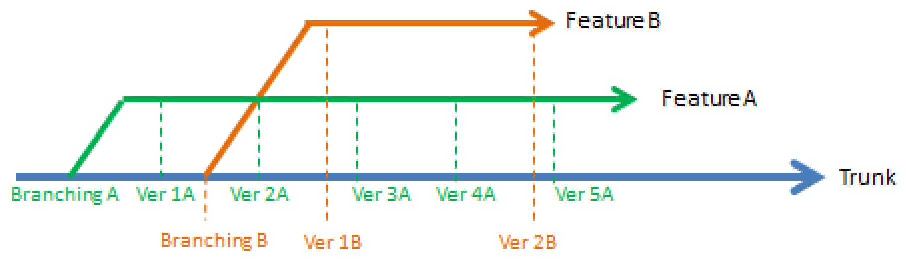
Feature Branching – Changes made in the feature branch can be merged into the trunk environment, while changes in other feature branches will be protected through a baseline-aware analysis.

Task/Bug/Hotfix Branching – The analyses will be performed only for objects that were assigned to the selected tasks. Baseline-aware analysis will ignore changes made in the context of other tasks and will alert developers of conflicts that require manual intervention.

Trunk Based Development – Changes made to a trunk branch are incorporated in a release branch only with the approval of personnel with the proper permissions, such as a Merge and Build Manager, and after baseline-aware analysis. This can include bug fixes, specific elements or enhancements, depending on the organization.

Team Branching – Changes made in a single team branch can be merged into the trunk environment, while changes in other team branches will be protected due to the baseline-aware analysis.

Ideally, every merge process should have its own baseline, representing the last time the environment was merged. Using the wrong baseline can result in incorrect recommendations. The following diagram illustrates how two feature branches should be merged to the trunk.



Every time a feature is merged into the trunk, the label becomes the baseline for the next merge. The first baseline is the label used to create the branch.

Feature Branch	Version	Baseline Version
Feature A	Ver 1A	Branching A
Feature A	Ver 2A	Ver 1A
Feature B	Ver 1B	Branching 1B
Feature A	Ver 3A	Ver 2A
Feature A	Ver 4A	Ver 3A
Feature B	Ver 2B	Ver 1B
Feature A	Ver 5A	Ver 4A

When should the database branch environment be refreshed? Whenever you refresh the relevant code files. The baseline of the last merge will define the refresh, differentiating between changes made after the merge and changes made in the trunk (as the other branches merged in), with the relevant changes merged into your refreshed branch.

DBmaestro
DevOps for Database

Getting the Best Database Deployment Outcomes

As development teams grow larger and more complex, and the pressure to issue updates and releases continues to grow, the various branching methodologies become increasingly valuable as viable solutions for organizing development efforts across organizations, teams and projects.

While branch methodology eases the development process, merging changes continues to present a challenge for development teams. Compare-and-sync tools offer some valuable feedback by detecting changes between two environments, but these tools fail to identify the origin of such changes. They therefore provide little value in determining which changes are safe to be deployed and cannot be effectively automated. As a result, developers relying only on compare-and-sync solutions may end up overriding critical changes introduced by another team or for another function in a different branch.

The solution for streamlining development collaboration and making true automation possible for the database, is baseline-aware analysis. By comparing not only the source and the target environments, but also the baseline, tools offering baseline-aware analysis are the critical ingredient for modern development teams working with any branching methodology. They also make seamless, simultaneous database and application development possible, for the best possible deployment outcomes.