DBmaestro
DevOps for Database

DBmaestro
DevOps for Database

How to integrate development and deployment into a single process

The Need

In today's agile world, development teams must respond quickly to ever changing business needs. The traditional practice is to use a task management system, such as JiRA, TeamForge, TFS, RTC, or others. These systems manage the business requirements and break them into tasks for the developer who makes the change, both for the application code as well as the database code.

Many organizations implement Continuous Delivery processes, allowing the IT department to support the high pace required by the organization. The best practice is to have the entire process of build, test and deploy - automated. This means that once a developer checks-in the new code to the source control, this code can be deployed to production.

While these automation best practices are done for the application code (C#, C++, Java etc.) they are generally not implemented for the database code (DDL, DCL and DML). There are many reasons why database code is treated differently than the application code and they will not all be covered here, but the two major impediments database code presents to automation are:

1. Development in a shared environment

In this environment developers introduce different changes which are not related to each other and should not be promoted at the same time.

2. Database deployment

Database deployment is done by generating DDL, DCL or DML scripts, which in turn are executed on the production database which stores the data required to operate the business. We must then confirm repeatedly that this is the correct script we wish to deploy and that it contains only the changes that we want.

As a solution, many organizations have a weekly, bi-weekly or monthly committee meeting to decide which database changes - based on the tasks from the task management system - are approved for deployment, and then manually generate the delta script specifically for the approved tasks.

The Challenge

This method presents a new challenge to the organization; how to build (compile) the application - and include changes related to the approved tasks for the database - while at the same time ignoring changes that were checked-in into the source control repository (but belong to tasks/user stories that were not approved).

Native Code Solution

The leading file-based source control tools today have solved this challenge by providing several methods such as branching the environment and integrating with the task management system. They also provide a method to extract the files based on the selected tasks.

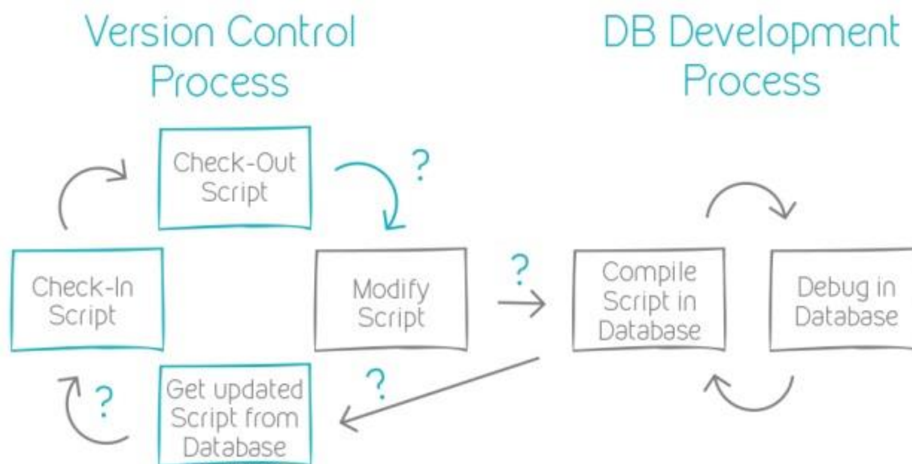
The process of building the application is done on the build server which wipes the workspace and extracts the application code from the source control repository for the approved tasks (or it branches and compiles it). The binaries - output of the compilation - can now be deployed to the next environment on the build server.

This process relies critically on an assumption that the source control repository is the single source of truth for the code. Only changes that were checked-in to the source control repository are part of the build process, changes made locally on the developer PC and not checked-in cannot be promoted.

This may sound trivial, but there is a major technology which is part of the application, and the standard methods of file-based source control don't apply to it. This of course would be the database!

Here are some of the reasons why source control does not work for the database:

1. When manually generating delta scripts for each task, there is no guarantee that the source control script matches the database



2. Even when automatically generating delta scripts by comparing environments and syncing them, the environment contains changes derived from tasks that were not approved - and which should not be included in the delta script.

Let's delve a little deeper into the reasons for these challenges

The Database Challenge

There are several approaches to managing database changes:

1. Development - Save the CREATE and the ALTER scripts in the file-based source control. The CREATE script is used to create the database from scratch and the ALTER script is used when upgrading the application.

The issue here is that there is no guarantee that the file-based source control is the single source of truth as anyone can login to the database and make changes that will not find their way to the file-based source control. QA tests will pass because the changes are in the database.

2. Deployments - Using tools that compare two database environments, or file(s) with a database - and generate the deployment script.

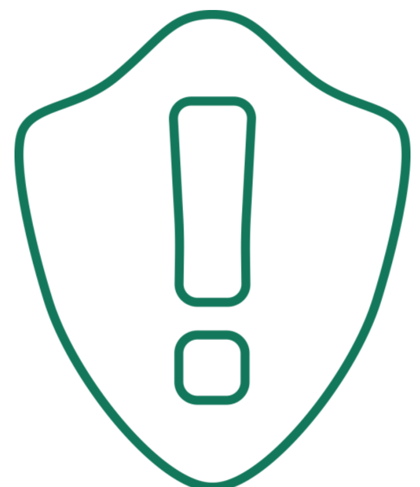
The problem:

This simple compare & sync method is risky because it only compares source to target. It is not aware of source control, and it may generate a script that incudes commands that belongs to tasks that should not be included, or that even revert a critical fix. For example, if there is an index in the target environment and not in the source control repository – these tools are not aware if it was added in production (and thus needs to be protected from outdated revisions from development), or if it was removed from development (and thus needs to be dropped in production).

3. Tasks dependencies – deploying the correct revision of objects which were changed by two or more tasks.

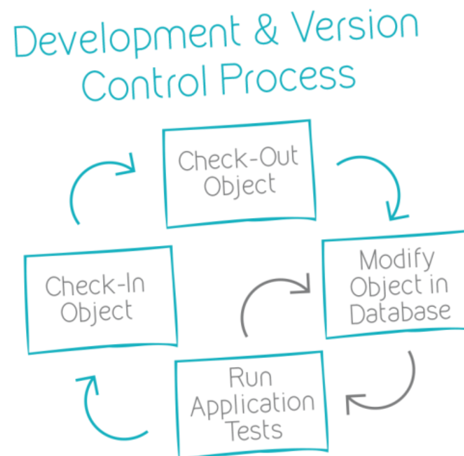
The problem:

By reviewing the revision of the selected task and comparing it to current status, without considering the past deployment, the generated script can revert changes made on objects and were already deployed as part of a previous deployment.



The Solution

The solution is to integrate development and deployment into a single process known as Database Enforced Change Management. This involves ensuring that a source control repository is the single source of truth, by using an enforcement change policy that guarantees every change to the database objects is documented.



The above flow should be combined with a sophisticated impact analysis which uses baseline aware analysis algorithms and is aware of the nature of the change, and can ignore changes which were done in a different branch, parallel environment or that conflict with a critical fix.

Simple Compare & Sync

Source vs. Target	Action
=	No Action
≠	?

Baseline Aware Deployment

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

This process works very well for native code (Java, C#, C++ etc.) but it is not implemented for the database code (table structure, procedures, lookup content and so on).

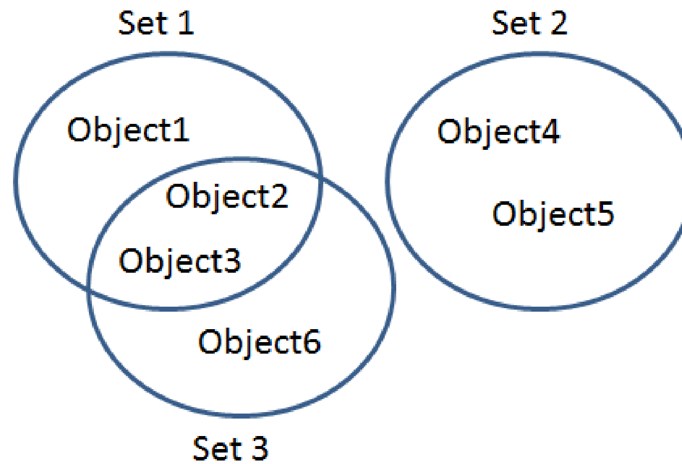
The reason for this is that organizations don't want changes to be made in the deployed database in the same way as their code. They want to control which changes in the database will be released and when. Saving just the object definition in the Check-In is not enough. Information regarding the reason for the change (task, user story, requirement, epic etc.) is as critical as saving the object definition.

We need to think differently in order to meet the demands of the database.

The Theory

The concept is to deploy database changes between environments by keeping the reason for the change (task) as part of the metadata of the check-in. Using simple set theory, we can always add a new set to the one we promoted previously with each set representing the changes for a relevant task.

First we generate the build (delta) script for set 1. Next we generate the build (delta) script for set 1 and 2 and so on. Any new set may add new objects or new revisions to existing objects.



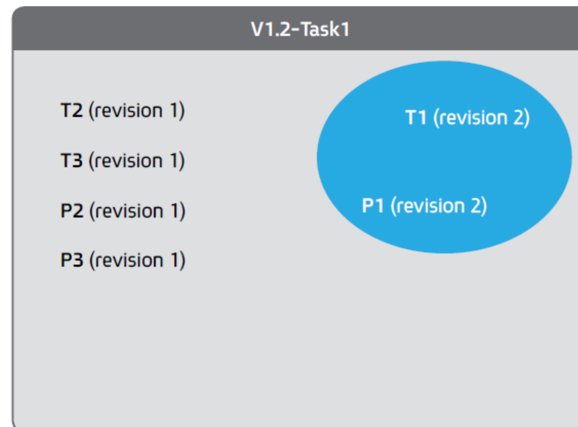
Practice

Let's assume the following scenario:

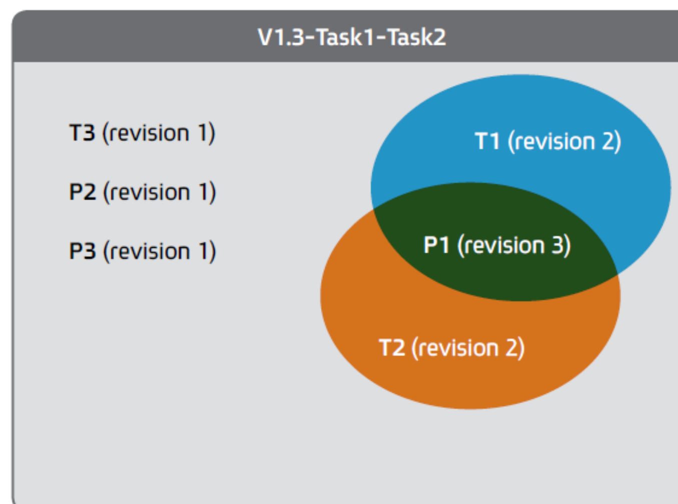
Our database contains Three tables (T1 [c11 int, c12 varchar], T2 [c21 date, c22 int] and T3 [c31 int, c32 date]), and 3 procedures (P1, P2 and P3). We'll name the current structure V1.1.

Starting point
T1 (revision 1)
T2 (revision 1)
T3 (revision 1)
P1 (revision 1)
P2 (revision 1)
P3 (revision 1)

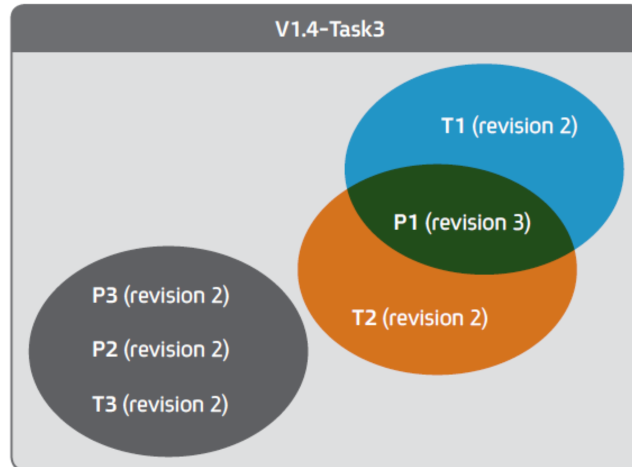
1. Developer A introduces the following changes to **T1** (add new column – C13 date) and altered procedure **P1** as part of Task1. We'll name this structure V1.2-Task1



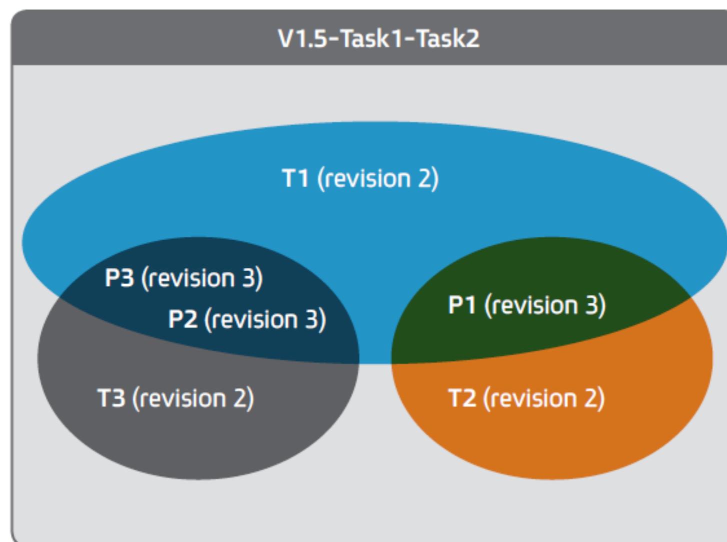
2. Developer B introduces the following changes to **T2** (add new column – C23 varchar) and altered procedure **P1** (which includes the change made by developer A) as part of Task2. We'll name this structure V1.3-Task1-Task2. As you can see, Task2 is dependent on Task1



3. Developer C introduces changes to **T3** (add new column C33 varchar), altered procedure **P2** and altered procedure **P3** as part of Task3. We'll name this structure V1.4-Task3



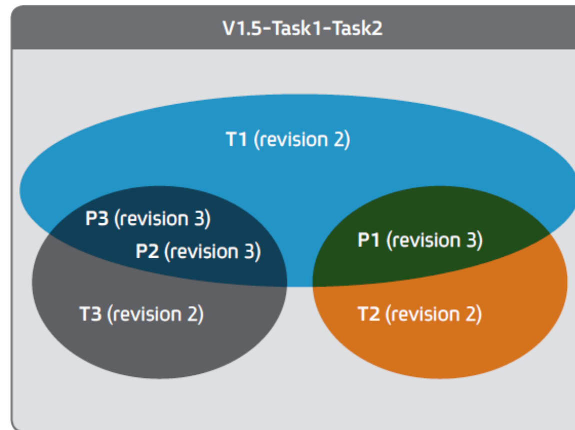
4. Developer C introduces changes to **T1** (change C13 from date to int) and altered procedure **P3** as part of Task1. We'll name this structure V1.5-Task1-Task2



Now there is only a need to promote changes that are related to Task1 and Task3.

Let's review the current status:

Object	Object Revision
T1	3
T2	2
T3	2
P1	3
P2	3
P3	3



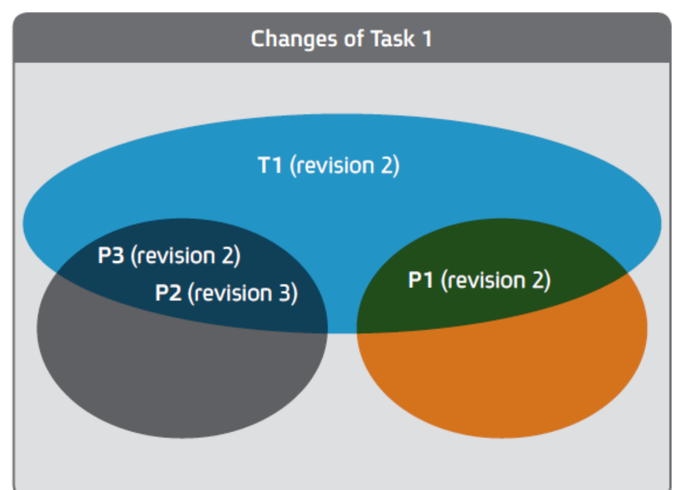
The challenge now is to understand which objects and which revision should be considered when generating the delta script for Task1 & Task3.

Let's review the total history of changes

Sequence	Object	Change	Object Revision	Created for
1	T1	New column C13	2	Task1
2	P1	Altered	2	Task1
3	P3	Altered	2	Task1
4	T2	New column C23	2	Task2
5	P1	Altered	3	Task2
6	T3	New column C33	2	Task3
7	P2	Altered	2	Task3
8	P3	Altered	3	Task3
9	T1	Modify C13	3	Task1
10	P2	Altered	3	Task1

Collecting changes made for Task1 returns the following

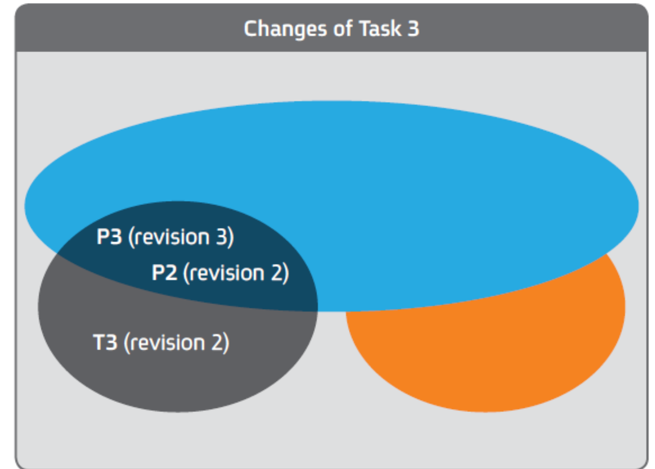
Object	Object Revision
T1	3
P1	2
P2	3
P3	2



DBmaestro DevOps for Database

Collecting changes made for Taks3 returns the following

Object	Object Revision
T3	2
P2	2
P3	3



As you can see, if we execute the script of changes for Task1 followed with the script of changes for Task3 or, if we first execute the script of changes for Task3 followed by the script of changes for Task1 we will not get the desired structure. This is because there are objects which were affected by different tasks and the final revision of the objects is not in the same task.

Object	Expected Object Revision	Executing changes Task1 Followed by Task3	Executing Changes Task3 Followed by Task1
T1	3	3	3
T3	2	2	2
P1	2	2	2
P2	3	2 - Wrong	3
P3	3	3	2 – Wrong

By using the union of sets we need to collect changes of Task1 and then collect changes of Task1 & Task3 together. If Task7 was also approved, we collect changes of Task1, Task1 + Task3, and Task1 + Task3 + Task7.

DBmaestro
DevOps for Database

Summary

Based on the set theory concept that the latest revision of the object returns in the union operation, we can promote the changes of all tasks (A and B). By promoting changes of task A, followed by promoting changes of task A union task B, will result in promoting only task B changes.

This requires (1) to rely on our source control as a single source of truth, (2) having the reason (task, user-story, epic, requirement etc.) included in the metadata of the check-in (3) and that the analysis that generates the delta script should utilize the previous requirements in order to use the correct object definition from the source control - based on the selected tasks and previous deployments.

With this practice, development teams now have the ability to decide on tasks, based on business needs rather than on their technology dependency. They can now respond quickly to the ever changing business requirements in today's agile world.

Database enforced change management means that development flexibility can be drastically increased, allowing for mid-course changes and corrections. Huge resources can be saved by automatically developing, testing and responding to scope changes for database upgrades and rollback scripts.

With the application code and the database code using the same method across the entire lifecycle and by using this system of selecting changes based on tasks, only approved code changes are promoted to production.

Database Continuous Delivery should follow the proven best practices of change management, enforcing a single change process over the database, and enabling efficient resolution of deployment conflicts to eliminate the risk of code overrides, cross updates and merges of code, while plugging into the rest of the release process. This adds up to the ability to correlate changes and business requirements.

Headquarters 300 Baker Avenue, suite 300, Concord, MA 01742, USA +1 978.405.3368
EMEA Headquarters 21 Yagiya Kapaim st. Petach Tikva 4900101, Israel +972.3.9248558
Copyright © 2016 DBmaestro. All rights reserved.

Follow Us: