**Flyway DB > Demo use case**

[External]

Cognizant Team:

The below email, which I had said I would send, explains in detail some of the issues with our Flyway approach. Please take a look at this and come up with some recommendations for us.

**Flyway Problem 1:**

The development team creates database SQL scripts that follow an ordered naming convention. This allows Flyway to execute the scripts in the intended order, and to correctly track what scripts in a new code drop have already been applied to the DB and which ones have to be applied. The naming convention currently used by the dev team includes the current date when the script is created (e.g. V20201230\_080\_59931\_01\_3700\_\_USCASA\_CBPH.sql). This means that Flyway will apply scripts in the order that they were created by developers.

Every month, MBP maintenance team releases a monthly maintenance release (MM). For instance, we could have MM4 released Dec 15 and MM5 released Jan 15. The issue comes when downstream teams report an issue in an already-released MM. If on Jan 1 an issue is reported in MM4, a new SQL script will be created with a new date in the naming, e.g. Jan 5. That script will be added to both the Flyway dbloader images for MM4 and for MM5.

This causes the following issue: If a downstream team is doing a cumulative upgrade, e.g. from MM3 to MM6, they should be able to simply take the MM6 image which contains all the cumulative scripts needed to build an MM6 environment up from scratch or to do this upgrade from MM3. The problem here is that the script that was created as a patch for MM4 is in the wrong order. Instead of being applied right after the other MM4 scripts, and before all MM5 scripts, it will be applied after any MM5 scripts that were created before Jan 5. This could cause issues.

To solve this, the CoreFusion team (downstream from Core Development) has taken on the task of renaming Core scripts in order to force a correct order. For instance, Core Development recently released a script V20201230\_080\_59931\_01\_3700\_\_USCASA\_CBPH.sql which CoreFusion renamed to V20210101\_999\_101\_\_V20201230\_080\_59931\_01\_3700\_\_USCASA\_CBPH.sql. But this renaming creates a new handoff and dependency, which causes an unnecessary delay. It also creates a disconnect between the base source code and the code being used downstream. It also means that e-fixes cannot flow straight from core development into downstream environments. It also creates inconsistency in the script naming for CoreFusion-touched components vs those that CoreFusion does not touch.

**What is Needed:** A new approach to SQL script naming that allows Flyway to seamlessly handle scripts in the correct order. This approach must support patches into past MMs as well as rollback.

Flyway Problem 2:

Flyway is designed to maintain an ongoing cumulative set of scripts; as long as the proper naming convention is followed, you can always deploy the latest image and the Flyway framework will correctly determine the current version of the DB and the exact scripts needed to bring it to the desired target version. This allows any environment to be upgraded from any version level to any later version level. These cumulative scripts would span all release boundaries, i.e. it should still work regardless of major vs minor releases.

The MBP development team has adopted the approach that for GA releases (e.g. an upgrade from 3.6 to 3.7), new consolidated upgrade scripts are created that replace the cumulative, Flyway-compliant scripts. This brings some performance efficiencies: for example, in a scenario where one minor release added a column, and then a subsequent minor release removed that column, the new GA script would skip the unnecessary changes that were later reversed. The GA scripts also handle updating account data more efficiently.

The problem with this approach is that it breaks the Flyway model. If we are using Flyway as designed, we could upgrade a client across a GA release boundary (e.g. from 3.6 MM4 to 3.7 MM2) without any custom scripting being needed. Flyway would take care of it. But with our current model, the 3.6 GA script would some GA scripts that would look new to Flyway; but when Flyway executes them, it runs into errors because some of the underlying changes were already implemented by the minor releases leading up to that GA release. This creates a mess of duplicates, errors, etc.

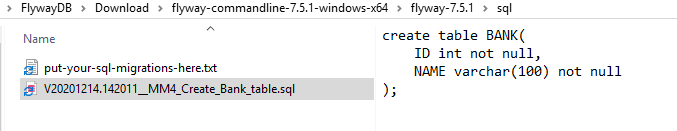
**What is Needed:** A rearchitected approach that looks at the entire process from a production deployment perspective. For instance, maybe we continue to repackage some scripts for GA releases but always supply a migration script of some sort that manipulates the Flyway schema table accordingly.

Thanks

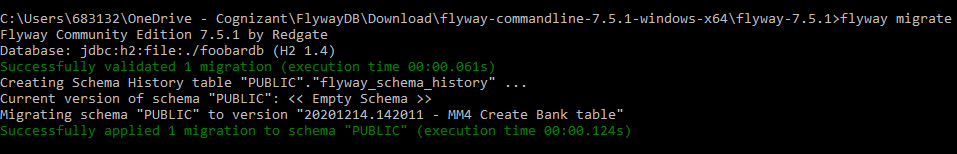
Chris

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

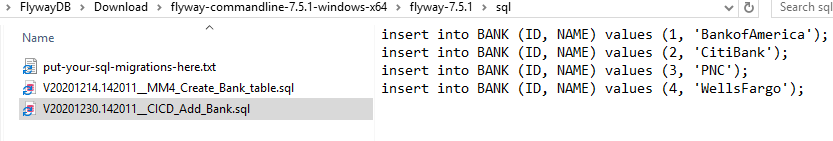
**Step1:** MM4 Release



$ **flyway migrate**

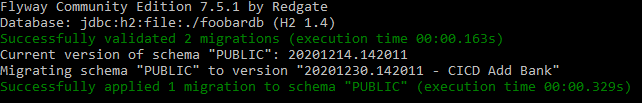


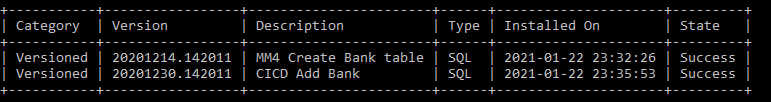
**Step2:** CI CD Release



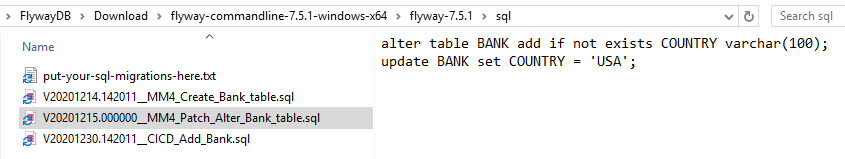
$ **flyway migrate**

**$ flyway info**

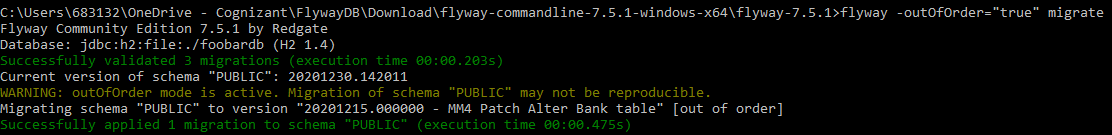




**Step3:** MM4 Patch Fix (missed Country column)



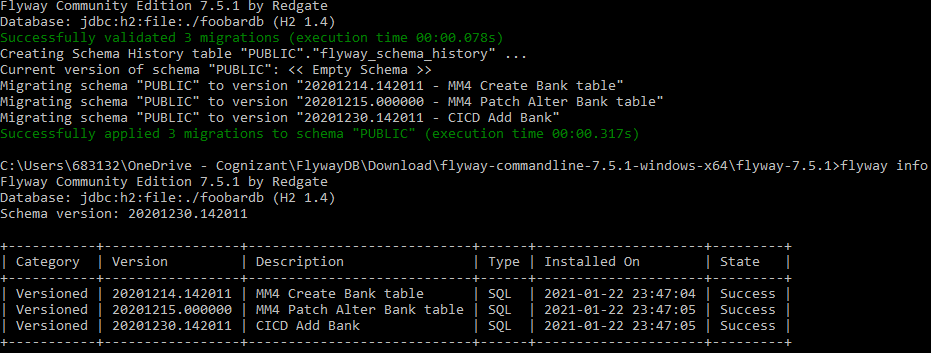
$ **flyway -outOfOrder=”true” migrate**



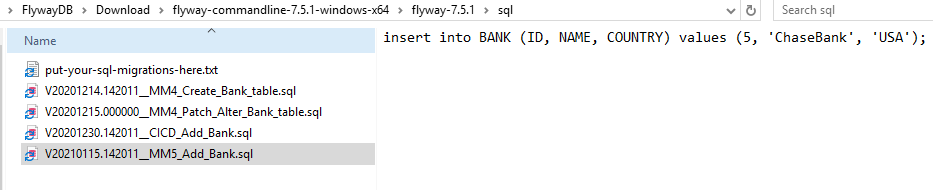
**$flyway clean**

**$flyway migrate**

**$flyway info**

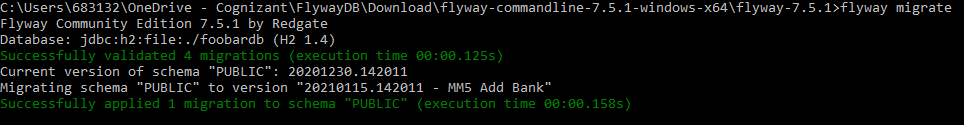


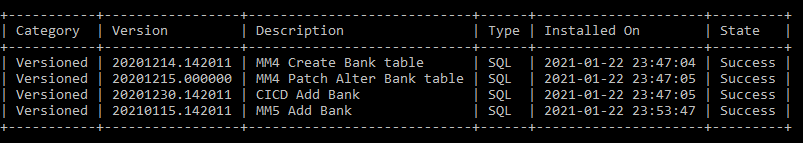
**Step4:** MM5 Release



**$ flyway migrate**

**$ flyway info**

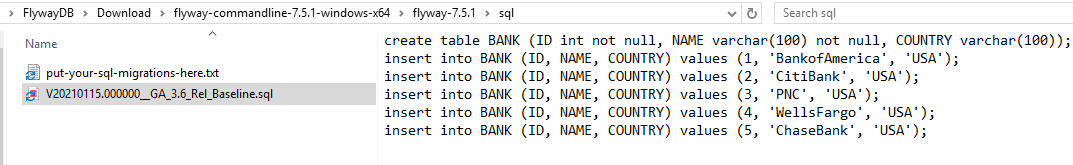




**Done!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

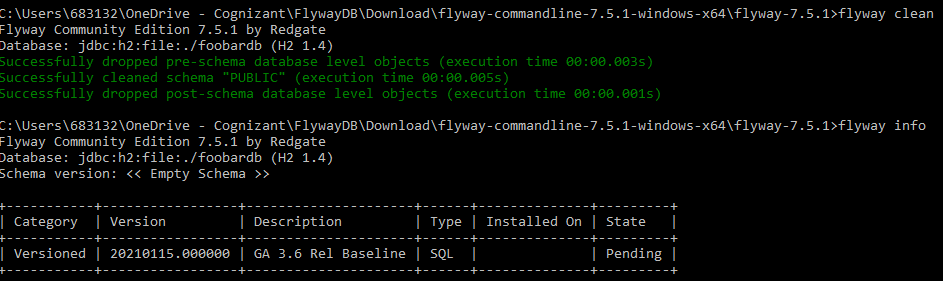
**Step1:** GA Release 3.6 optimized SQL scripts ready

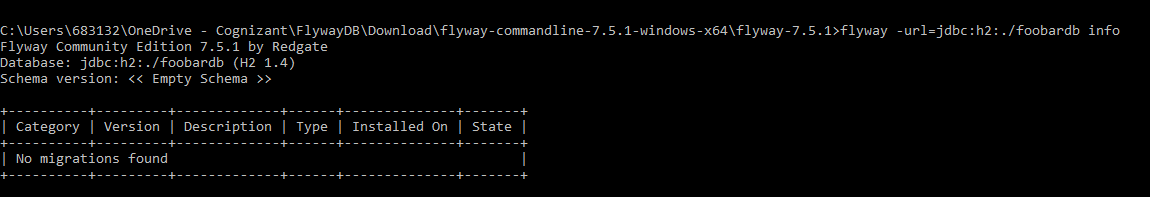


**$ flyway clean**

**$ flyway info**

**$ flyway -url=jdbc:h2:./foobardb.mv.db info**

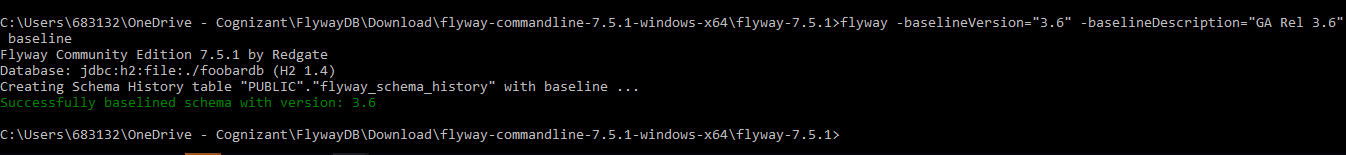


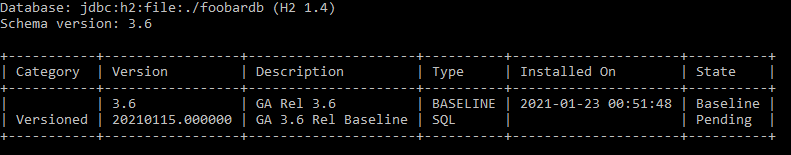


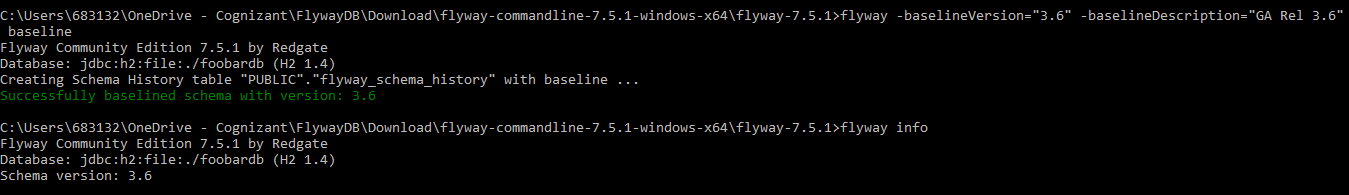
**$** **flyway -baselineVersion="3.6" -baselineDescription="GA Rel 3.6" baseline**

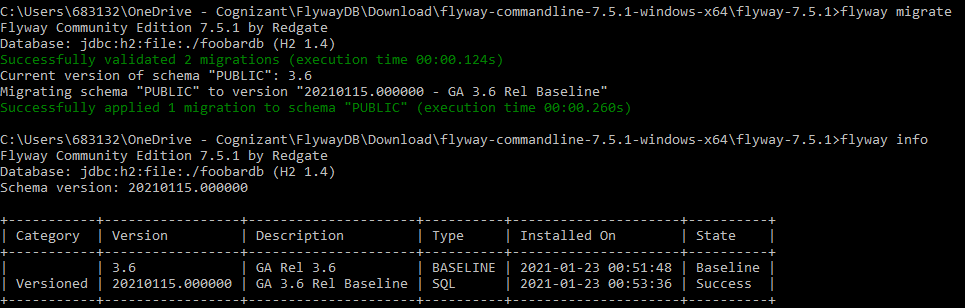
**$ flyway migrate**

**$ flyway info**





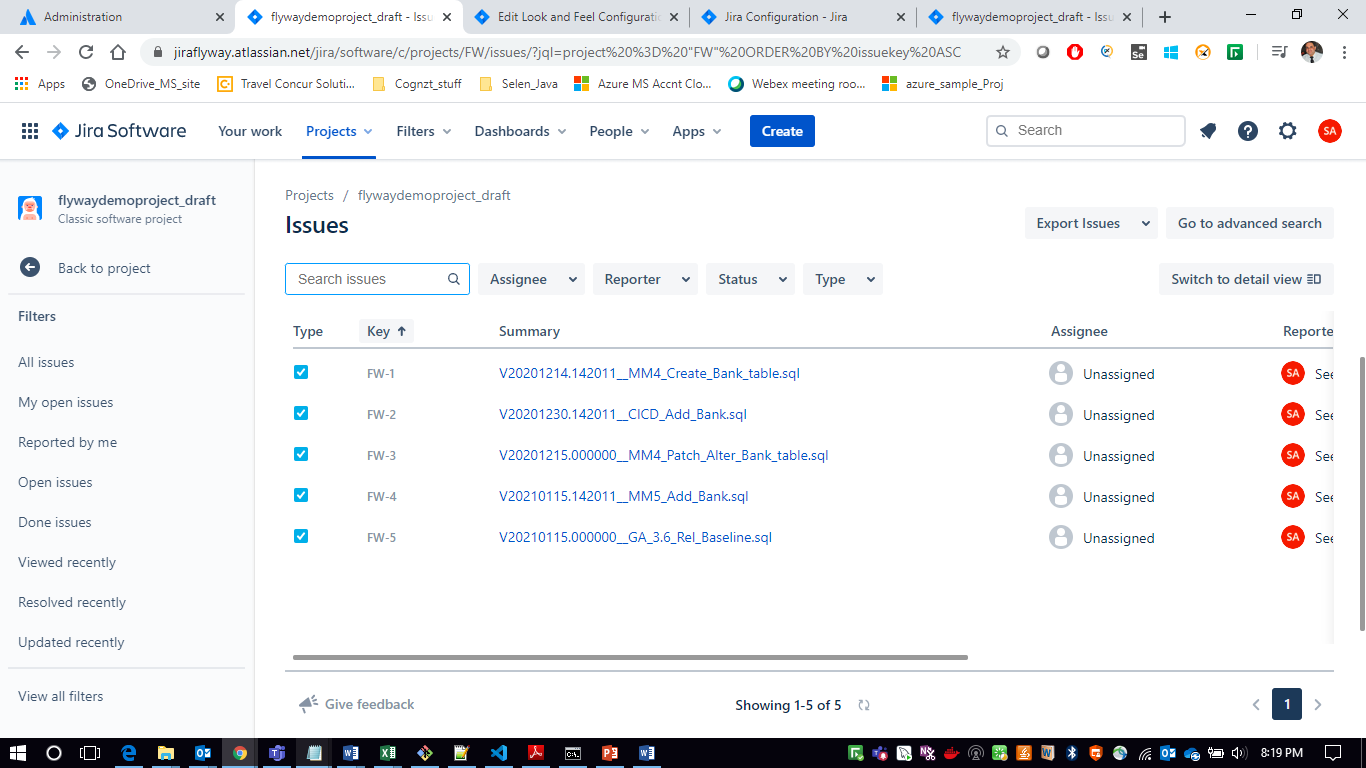




**Done!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jira create tasks /issue



Put the script name in the Summary

Unique (No Duplicates)

Increasing value

Format V\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*