Ready, steady, Git: Einführung eines Versionskontrollsystems

Carolin Hagemann, Pascal Thiel
Veolia Umweltservice GmbH, Trivadis GmbH
Hamburg

Schlüsselworte
GIT, Versionskontrolle, Datenbankentwicklung, Forms, Oracle APEX

Einleitung / Abstract

Ein System zur Versionierung der Programmdateien ist in vielen höheren Programmiersprachen wie Java gar nicht mehr wegzudenken. Im Bereich der Datenbankentwicklung ist das Thema im Vergleich allerdings noch nicht so weit verbreitet.
Bei der Etablierung des vorweg evaluierten Versionierungstools GIT und der dazugehörenden Entwicklungsprozesse sind viele Fragen und Unsicherheiten im Entwicklerteam aufgetreten.
Im Rahmen dieses Vortrags wird zu Beginn auf den Nutzen und die allgemeine Funktion eines Versionierungstools eingegangen. Im Anschluss werden die eingeführten und geänderten Entwicklungsprozesse sowie die Erfahrungswerte in Kombination mit der Oracle-Datenbankentwicklung näher beleuchtet.

Was ist eine Versionskontrolle?
Der Einsatz von Versionskontrollsystemen ist in höheren Programmiersprachen ein etablierter Standard.
Eine Versionskontrolle dient hauptsächlich der Verwaltung von Quellcode. Dieser Verwendungszweck untergliedert sich selbst in weitere Unteraufgaben.
Durch die Speicherung und Ablage von Quellcode können die getätigten Änderungen stets nachvollzogen werden, da auch ein direkter Vergleich mit der Vorgängerversion der Datei möglich ist.

Dies hat folgende Vorteile:
Es können Änderungen einfacher und schneller nachvollzogen werden (z.B. wann eine Änderung durch wen durchgeführt wurde, um Rückfragen zu stellen oder Korrekturen einzufordern).
In einem kritischen Fehlerfall in der Produktionsumgebung kann relativ schnell der Ursprungszustand wiederhergestellt werden, da einerseits eine Übersicht der geänderten Quellcode-Dateien vorhanden ist und andererseits ein Rollbackscript durch die mitgelieferten Funktionalitäten einer Versionsverwaltung schnell erstellt ist. Auch gibt es hier diverse Erweiterungsmöglichkeiten durch andere Technologien, die allerdings erst später behandelt werden.
Zusätzlich zur Wiederherstellung von alten Versionen ist gewährleistet, dass jede Änderung des Programmcodes und jeder Stand, der jemals in dem Projekt abgespeichert wurde, archiviert ist. So kann nicht nur sichergestellt werden, dass auch alte Programmversionen immer installiert werden können, sondern bietet auch die Möglichkeit für Audits o.ä. Informationen zur Verfügung zu können.
Insbesondere in größeren Projektteams können sich die Entwicklungsarbeiten überschneiden. Dabei werden bspw. die gleichen Programmdateien bearbeitet. Mithilfe einer Versionsverwaltung kann in diesen Fällen gewährleistet werden, dass die jeweiligen Änderungen automatisch in eine Ergebnisdatei zusammengeführt werden.
Müssen in einem Projekt Änderungen für verschiedene Releasezeiträume oder bspw. Bugs während der Entwicklung einer neuen Programmversion durchgeführt werden, kann mithilfe einer Versionsverwaltung gewährleistet werden, dass die Entwicklungen nicht miteinander kollidieren und anschließend zusammengeführt werden können. Die Entwickler verfügen hierbei um einen eigenen Bereich (Branch) in dem sie ihre Änderungen speichern können.

Warum Versionskontrolle in der Datenbankentwicklung?
Die oben beschriebenen vorteilhaften Funktionalitäten können in jedem Softwareprojekt verwendet werden. Während der Datenbankentwicklung können Änderungen an Packages, Tabellen u.ä. schnell umgesetzt und anschließend gesammelt in die anderen Umgebungen übertragen werden.
Auch wenn zusätzlich zur Datenbankentwicklung Forms- oder APEX-Applikationen entwickelt werden, können diese in eine Versionskontrolle abgelegt und fast wie jede andere Datei verwendet werden.
Das bedeutet, dass alle Vorteile einer Versionskontrolle in der Datenbankentwicklung verwendet werden können. Dieser Umstand erleichtert die Arbeit der Entwickler und macht die Änderungen zu jedem Zeitpunkt transparent.

GIT und Datenbankentwicklung
Bei der Datenbankentwicklung dreht sich alles um Tabellen und wie man sie befüllen kann. Hierfür wird eine Reihe von Packages mit diversen Prozeduren und Funktionen angelegt, um die Geschäftslogik abzubilden. Meist werden diese Entwicklungen direkt in einer Datenbank angelegt, damit der entstandene Quellcode auch getestet und optimiert werden kann.

Wie bereits erwähnt wurde, kann es dabei zu Konflikten kommen. Im Team wird ggf. das gleiche Package bearbeitet oder es kann nicht mehr sicher reproduziert werden welche Abschnitte geändert wurden und in das Installationsskript übernommen werden müssen. Natürlich gibt es für solche Probleme diverse Lösungsmöglichkeiten. Mit GIT als Versionsverwaltung lassen sich viele Probleme fast schon von selbst lösen.
Anders als im zuvor beschriebenen Prozess kann hierbei jedoch nicht mehr ausschließlich in der Datenbank entwickelt werden. GIT liegt nicht in der Datenbank und daher müssen Skripts und Packages in einer eigenen Struktur gehalten werden. Dies geht mit einem lokalen Verzeichnis, in welches sämtlicher Quellcode abgelegt wird. Dieses Verzeichnis wird in GIT integriert und ab diesem Punkt werden sämtliche Änderungen durch GIT indiziert. Zumindest fast. GIT speichert Änderungen immer in sogenannten Commits ab. Beliebig viele modifizierte Dateien können in einen Commit übernommen werden. Zu einem späteren Zeitpunkt kann der alte Projektstand betrachtet, Änderungen herauskristallisiert und ggf. wiederhergestellt werden. Werden die Scripts in der Datenbank direkt ausgeführt, unterscheidet sich die Entwicklungsarbeit im besten Fall nicht von dem Ergebnis der alten Arbeitsweise.
Am einfachsten geht dies mit den vier Befehlen „status“, „add“ und „commit“ und „push“. Der aktuelle Zustand des Projektes kann mit „status“ angezeigt werden. Welche Dateien wurden verändert? Welche sind neu hinzugekommen oder wurden gelöscht. Ist der Entwickler soweit, dass der Zustand in die Versionierung aufgenommen werden kann, können die Befehle „add“ und „commit“ in die neue Version überführt werden. GIT merkt sich hierbei in welchen Zeilen der Quellcode-Dateien Änderungen stattgefunden haben und kann so auch bei Bedarf die Unterschiede zwischen zwei Versionen darstellen.
Bei einem Entwicklerteam mit mehr als einer Person kommen noch zwei weitere Befehle hinzu. Damit im Team alle Zugriff auf die Versionsverwaltung haben, ist neben dem lokalen eigenen Verzeichnis ein „Server“-Verzeichnis notwendig. Dies kann zum Beispiel in einem Netzlaufwerk oder auch auf einem speziellen GIT-Server erstellt werden. Der bekannteste öffentliche GIT-Server ist der Dienst „Github“. Hier kann sich jeder registrieren und Projekte erstellen. Um nun mit dem Server-Verzeichnis zu kommunizieren werden die Befehle „push“, zum Senden der eigenen Änderungen, und „pull“, zum Empfangen von Änderungen vom dem Server, verwendet. Hat ein Teammitglied seine Änderungen an den Server überspielt und ein anderes Mitglied möchte diese in seiner Entwicklung berücksichtigen, fügt GIT die Änderungen nach dem „ull“ direkt in sein Verzeichnis ein. Solange die Entwickler nicht direkt in denselben Zeilen einer Datei gearbeitet haben, kann GIT die Änderungen automatisch miteinander vereinen.
Möchte man Änderungen eines anderen nicht direkt in seinen Quellcode einarbeiten, kann auch mit sogenannten Branches gearbeitet werden. Dies sind Zweigentwicklungen. Jeder Zweig kann unabhängig vom anderen weiterentwickelt werden. Alle Entwickler können auf diese Zweige zugreifen und erweitern; sind jedoch nicht gezwungen die Änderungen zusammenzuführen. Dies ermöglicht es an diversen Softwareänderungen zu arbeiten ohne in Abhängigkeiten zu geraten. Alle Zweige können auf dem Produktivstand aufbauen und sobald entschieden wurde welche Zweige in ein Update übernommen werden, können diese Zweige zusammengeführt werden. Auch hier wurde festgestellt, dass GIT in den meisten Fällen den Quellcode problemlos selbst zusammenführen kann.
Das Zusammenführen werden in einen „Release“-Branch übernommen. Dieser enthält dann sämtliche Changes und Neuerungen des geplanten Updates. GIT ermöglicht es auch hier jederzeit auszugeben welche Dateien und welche Stellen im Quellcode sich im Vergleich zum Produktivstand geändert haben. Dadurch lassen sich Risiken leichter abschätzen und genau bestimmen welche Programmteile vom aktuellen Update betroffen sind.
Um alles im Blick zu haben, gibt es diverse Tools die sowohl das eigene, wie auch das Serververzeichnis analysieren. Manche bieten Vorteile, wenn es beim Zusammenführen von Branches zu Problemen führt, andere bieten Übersichten über alle Zweige im aktuellen Projekt.

Kontaktadresse:
Carolin Hagemann
Veolia Umweltservice GmbH
Hammerbrookstraße 69
D-20097 Hamburg

Pascal Thiel
Trivadis GmbH
Paul-Dessau-Straße 6
D-22761 Hamburg