Hoe databaseveranderingen te koppelen met SCM-projecten

Database Change Management

Ben Suurmeijer

Sinds tientallen jaren halen veel organisaties voordeel uit het gebruik van geaccepteerde en bewezen Software Change Management (SCM) principes. Processen zoals 'file locking', 'check in/check out', 'compare and merge' en 'roll-backs' stellen organisaties in staat om op een productieve en gecontroleerde manier wijzigingen aan te brengen in hun applicaties. Echter, binnen de databasewereld werkt het hele principe van SCM niet bijzonder goed.

De belangrijkste reden is dat databases totaal anders zijn gestructureerd dan traditionele programmacode (zoals Java en .NET), en om die reden kunnen traditionele SCM-producten, zoals bijvoorbeeld IBM Rational of Subversion, niet gebruikt worden in de databasewereld. Het gevolg is een enorme kloof tussen SCM en database ontwikkelactiviteiten.
Een database is, in tegenstelling tot een softwareprogramma bijvoorbeeld, geen verzameling bestanden, en door het filesysteem wordt een database meestal gezien als één groot bestand. Bij veranderingen in programmacode wordt er een kopie van het programmabestand gemaakt en vindt de ontwikkeling van code plaats op deze lokale kopie. Een database echter is een centraal middel waar iedereen toegang tot heeft en waar ontwikkelaars tegelijkertijd wijzigingen in aan kunnen brengen. Er worden zelden kopieën gemaakt voor elke individuele ontwikkelaar. In situaties waar dit wel gebeurt, moet er nagedacht worden over hoe deze verschillende kopieën van de database voordurend gesynchroniseerd kunnen worden om databaseconsistentie te garanderen; een lastig probleem. Verder moet er rekening gehouden worden met versiebeheer, en bestaat er geen zogenaamde debug-omgeving.
Ook een gestructureerd proces van ontwikkeling, waarbij aanpassingen van de ontwikkelafdeling door worden gegeven aan QA (Quality Assurance), integratie en vervolgens productie, stuit bij database ontwikkeling op problemen. Bij de ontwikkeling van programmacode omvat dit proces weinig meer dan het simpelweg kopiëren en registreren van individuele bestanden. Echter, het verplaatsten van database objecten is niet zo eenvoudig. Het is niet zonder meer mogelijk om een gewijzigde tabel vanuit ontwikkeling in productie te nemen, omdat dan alle productiedata verloren gaan.

In tegenstelling tot een software ontwikkelaar, die gebruik maakt van een SCM-oplossing waarbij een gedetailleerde registratie van alle veranderingen wordt bijgehouden, moeten database ontwikkelaars vaak handmatig een lijst bijhouden met alle wijzigingen. Geloof het of niet, maar ik heb wel eens een database ontwikkelaar gezien die alle modificaties bijhield op gele memobriefjes die verspreid waren over zijn bureau en beeldscherm.
Met andere woorden, database ontwikkeling is een totaal andere tak van sport dan traditionele programmacode ontwikkeling.
Als gevolg van de enorme verschillen tussen software ontwikkeling en database ontwikkeling, is een geautomatiseerd proces voor Database Change management (DCM) op de manier van SCM nooit echt van de grond gekomen, en wordt er noodgedwongen gebruik gemaakt van een breed scala van handmatige processen en work-arounds.

DDL export

De meest bekende work-around is de database definition language (DDL) te exporteren om een databasestructuur te definiëren, en deze vervolgens als bestanden veilig te stellen binnen een traditionele SCM-oplossing (zie afbeelding 1). Echter, dit proces is verre van waterdicht. Niemand kan namelijk verwachten dat ontwikkelaars daadwerkelijk consequent dit proces volgen voor iedere verandering die ze aanbrengen in een database. En wat gebeurt er als iemand alleen maar de database aanpast, en niet de bijbehorende SCM-bestanden? Hoe wordt er omgegaan met deze inhoudelijke veranderingen? Een veranderde waarde van een parameter (in bijvoorbeeld een lookup- of metadatatabel), kan invloed hebben op het gedrag van een applicatieprogramma en moet goed beheerd worden. De vraag is alleen hoe? In dat geval moet er handmatig code geschreven worden om de wijzigingen in de databasestructuur mogelijk te maken zonder verlies van data. Dit is niet alleen erg tijdrovend, maar ook er foutgevoelig. Bovendien is deze handmatig aangebrachte code geen deel van een zogenaamd application lifecycle management (ALM) proces, en volgt hierdoor niet dezelfde regels en voorwaarden zoals bij normale programmacode.

Eerste generatie

De eerste generatie Database Change Management producten betreft in principe geautomatiseerde 'vergelijk en synchroniseer' oplossingen die het mogelijk maken om verschillende database-

omgevingen met elkaar te vergelijken. Dit type oplossingen genereert rapporten die de verschillen aangeven, waarna er automatisch code gegenereerd kan worden om de verschillen te compenseren. Deze oplossingen bieden een belangrijke verbetering in het Database Change Management proces. Door het geautomatiseerde karakter wordt het aantal fouten als gevolg van handmatig handelen in belangrijke mate gereduceerd.

Echter, veel change management problemen worden met de eerste generatie van Database Change Management producten nog steeds niet geadresseerd:

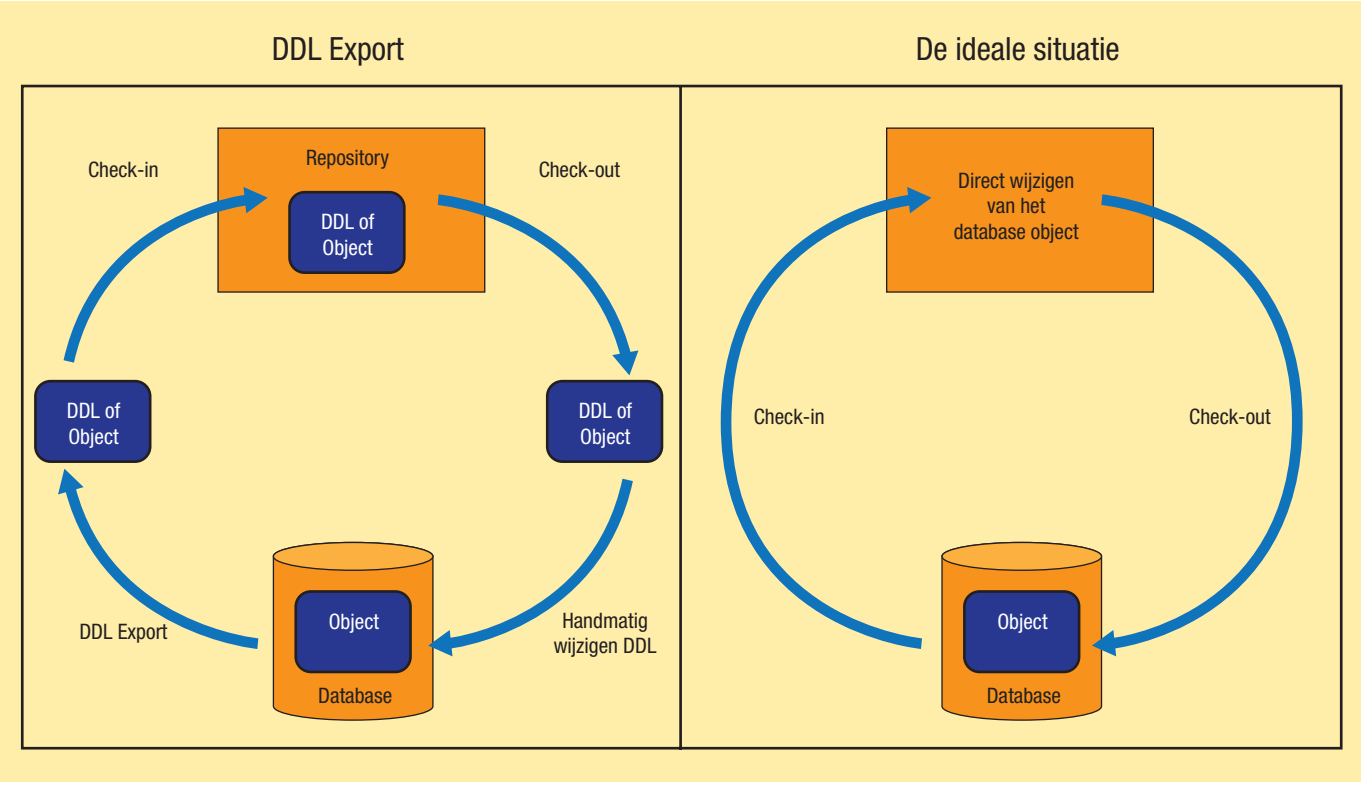
- Wat gebeurt er bijvoorbeeld als er veranderingen in de test-omgeving worden aangebracht? Dan kan het gebeuren dat de referentieomgeving niet langer identiek is, en wijzigingen kunnen verloren gaan;
- Een andere vraag is wat er precies gebeurt tijdens een ontwikkelcyclus. En welke stappen in deze cyclus genomen zijn. Er was geen informatie beschikbaar over hoe het uiteindelijke resultaat is bereikt;
- Kan iedereen die toegang heeft tot een database wijzigingen aanbrengen in de database;
- Welke veranderingen zijn door wie en wanneer gemaakt;
- Is database ontwikkeling effectief gesynchroniseerd met de software ontwikkeling;
- Hoe te handelen bij gedeeltelijke distributie van de veranderingen, bijvoorbeeld de noodzaak om een applicatie-aanpassing te distribueren zonder een andere, minder stabiele aanpassing?

In vergelijking tot moderne SCM-oplossingen biedt de eerste generatie Database Change Management producten weinig meer dan 'vergelijk en synchroniseer'. Ze bieden zeker niet de complete functionaliteit van moderne SCM-oplossingen, zoals bijvoorbeeld de mogelijkheid om veranderingen terug te draaien (rollback), check in/check out, file locking, baseline settings, code merging en veranderingsrapporten voor auditing en regelgeving. Omdat deze eerste generatie Database Change Management producten nog steeds niet in staat is om naar tevredenheid veranderingen te managen, is het wellicht beter deze producten te categoriseren als database development management in plaats van Database Change Management producten.

Nieuwe generatie

Vandaag de dag zijn database ontwikkelaars meer en meer op zoek naar een oplossing die hen in staat stelt om het volledige proces van databaseveranderingen te beheren, in principe een oplossing die dezelfde functionaliteit kan bieden als traditionele SCM-oplossingen:

1. Het afdwingen van formele, gedisciplineerde Database Change Management processen en workflows;
2. Het werk van verschillende database ontwikkelteams synchroniseren en coördineren;
3. Het ondersteunen van de volledige database ontwikkelcyclus, vanaf ontwikkeling via Q&A, naar integratie en tenslotte productie;
4. Beheren en volgen van veranderingen;
5. Het bieden van een goede, stabiele centrale opslag (repository)



Afbeelding 1: DDL export als work-around.

voor het opslaan van een veranderingshistorie voor auditing-doeleinden;
6. Het koppelen van databaseveranderingen met traditionele SCM-projecten;
7. Databaseveranderingen distribueren door middel van SCM-taken.

Om dit te realiseren moet de nieuwe generatie Database Change Management oplossingen het complete palet van Software Change Management functionaliteit in een databasewereld gaan ondersteunen. In tegenstelling tot de eerste generatie oplossingen, welke zoals we hebben gezien niet meer biedt dan een 'vergelijk en synchroniseer' product, moeten moderne Database Change Management oplossingen vandaag de dag een uitgebreid scala aan functionaliteiten kunnen bieden waaronder object locking, check in/check out, baseline setting, samenvoegen van code en een rollback van veranderingen.
In plaats van het foutgevoelige en handmatige proces waarvan database ontwikkelaars vandaag de dag noodgedwongen gebruik maken om aanpassingen in de database aan te brengen, hebben ontwikkelaars een dwingende behoefte aan een manier om wijzigingen direct en effectief aan te kunnen brengen in de structuur van de database.

> Database ontwikkelaars moeten vaak handmatig een lijst bijhouden met alle wijzigingen

Maar, ook de nieuwe generatie Database Change Management oplossingen is niet geheel zonder uitdagingen. Er moet bijvoorbeeld een nieuw type centrale opslag (repository) worden gecreëerd, dat zowel speciale attributen, database objecten als ook database inhoud kan opslaan. Verder is er het probleem van het beheren en coördineren van de centrale resources en de gekopieerde resources. Een veranderingsproces moet worden gedefinieerd en worden afgedwongen, en de activiteiten van de diverse ontwikkelteams moeten worden gesynchroniseerd. Database ontwikkeling moet worden gekoppeld met traditionele SCM-componenten zoals versiebeheer, verandering-sets (change sets) of activiteiten.

Maar ondanks deze uitdagingen biedt een succesvolle implementatie van de nieuwe generatie Database Change Management oplossingen enorme voordelen. De crux is om de juiste oplossing te vinden en de focus te houden op risicobeheersing. Een goede Database Change Management oplossing van de nieuwe generatie moet aan de volgende eisen voldoen:

- Procedures moeten worden afgedwongen, zodat wijzigingen alleen via een SCM workflow doorgevoerd kunnen worden;
- Database ontwikkeling moet zodanig worden gedaan dat botsingen en conflicten voorkomen kunnen worden;
- Standaarden moeten gedefinieerd kunnen worden en afgedwongen via check in/check out en labels;
- Een compleet audit trail van alle veranderingen moet kunnen worden vastgelegd;
- Een automatische rollback van veranderingen, zowel in ontwikkel- als ook in productieomgevingen moet mogelijk zijn;
- De uitrol van verschillende databases moet volledige geautomatiseerd plaats kunnen vinden, met three-way comparison en analyse met gebruik maken van het samenvoegen van databasecode.

dbMaestro TeamWork

Teamwork van dbMaestro maakt het mogelijk om van database ontwikkeling een integraal onderdeel van een gedisciplineerd SCM-proces te maken in plaats van een handmatig proces of, op zijn best, semigeautomatiseerd proces. TeamWork voldoet in alle opzichten aan de eisen voor een nieuwe generatie Database Change Management oplossingen, en vergroot in belangrijke mate de voordelen voor organisaties die een SCM-proces volgen voor zowel software ontwikkeling als ook database ontwikkeling. Veranderingen in databases kunnen nu op eenzelfde manier gevolgd worden als veranderingen in de softwarecode, waardoor het complete ontwikkelproject, inclusief databaseveranderingen, op een gecontroleerde, geregistreerde en gestandaardiseerde wijze uitgerold kan worden. Teamwork integreert met een oplossing als IBM Rational.

Conclusie

Programmacode ontwikkelaars hebben meer dan twintig jaar de voordelen ervaren van het gebruik van SCM-oplossingen. Het is nu tijd om een Database Change Management proces naar hetzelfde niveau te tillen als een SCM-proces, en database ontwikkelaars en DBA's dezelfde mate van eenvoud en procesmatige beveiliging te bieden bij database ontwikkeling. Integratie met bestaande, traditionele SCM-producten biedt een homogeen ontwikkelproces, identiek voor software ontwikkeling, alsook voor de ontwikkeling van database objecten.
De database developer community heeft behoefte aan oplossingen met voordelen als object-locking en auditing, die een betere controle bieden over de ontwikkelprocessen binnen een team. Deze benadering biedt tevens de mogelijkheid tot het distribueren van veranderingen en versiebeheer van database objecten en schema's.
Een innovatieve volgende generatie Database Change Management oplossing, zoals dbMaestro TeamWork, biedt al de hiervoor beschreven mogelijkheden, plus heel veel meer.

Ben Suurmeijer (ben.suurmeijer@go-esi.com) is als software consultant werkzaam bij go-ESI.