Edition Based Redefinition - database-applicaties aanpassen zonder downtime voor de eindgebruikers

Hidden Gems in Oracle - deel 1

23 dec 2014 - John van den Hurk

Volgens velen is EBR (Edition Based Redefinition) een van de belangrijkste nieuwe features in de Oracle 11gR2 Database release. Reden genoeg om in dit eerste artikel in de serie Hidden Gems eens te nader te beschouwen wat deze gratis optie van de Oracle database te bieden heeft.

Volgens velen is EBR (Edition Based Redefinition) een van de belangrijkste nieuwe features in de Oracle 11gR2 Database release. Reden genoeg om in dit eerste artikel in de serie Hidden Gems eens te nader te beschouwen wat deze gratis optie van de Oracle database te bieden heeft.

Het wordt vandaag de dag steeds belangrijker dat systemen up-to-date blijven en tegelijkertijd 24x7 beschikbaar zijn. In het geval van een applicatie die in de Oracle database naast data ook gebruik maakt van logica (in de vorm van PL/SQL, views, triggers etc.) was het tot voor kort vrijwel onmogelijk om wijzigingen aan te brengen zonder dat de applicatie hiervoor gestopt moest worden. Het aanbrengen van wijzigingen leidde voorheen onherroepelijk tot het niet beschikbaar zijn van de applicatie voor de gehele duur van de installatie.

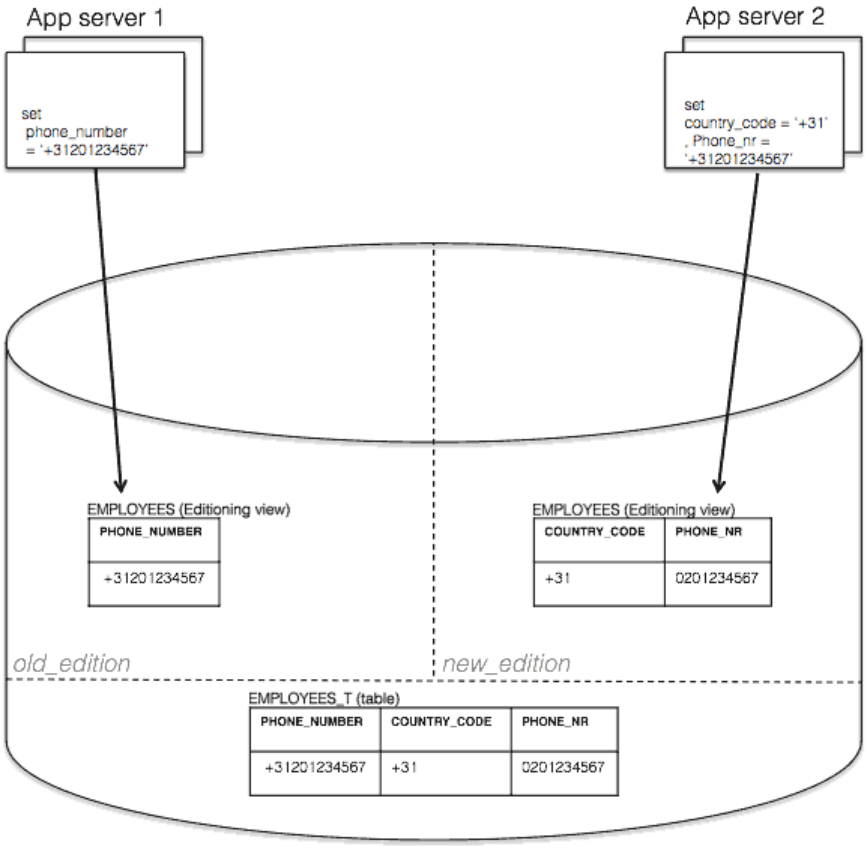
Online deployments

Met de komst van EBR wordt het mogelijk om een groot deel van de wijzigingen in een reguliere upgrade of patch van een applicatie door te voeren, terwijl de applicatie gewoon beschikbaar blijft voor de gebruiker (zogeheten online deployments). Voor specifieke soorten wijzigingen blijft het wel noodzakelijk dat de applicatie gestopt wordt. Verderop in dit artikel volgt meer uitleg over welke wijzigingen wel en niet online uitgevoerd kunnen worden.

Naast online deployments kan EBR ook ingezet worden om fallback en gefaseerde invoering mogelijk te maken. Het wordt namelijk mogelijk om meerdere versies van database-objecten gelijktijdig in een database te hebben. De nieuwe versies van deze objecten zijn actief binnen een zogenaamde editie. Zo kunnen bij installatie van een software-update alle wijzigingen in zo'n editie geïnstalleerd worden, terwijl de oude versies niet geraakt worden.

Dit laatste is belangrijk omdat het betekent dat het vrij eenvoudig is om terug te vallen naar de vorige editie. Het is zelfs mogelijk om meerdere versies actief te gebruiken. Dit geeft de mogelijkheid om een selecte groep gebruikers de nieuwe versie aan te bieden terwijl de rest van de gebruikers nog de oude versie gebruikt.

Hoe werkt het?



De werking van EBR is het eenvoudigst uit te leggen door gebruik te maken van een voorbeeld. Stel dat we een applicatie, gebaseerd op het standaard HR-schema, aan willen passen.

De wijziging moet aan de volgende requirements voldoen:

- Het bestaande telefoonnummer in de employee tabel moet gesplitst worden in twee velden (landcode en lokaal nummer) dus +31201234567 wordt +31 en 0201234567.
- Het installeren van de wijzigingen moet gedaan kunnen worden terwijl de applicatie in gebruik is.
- Een fallback naar de vorige versie moet mogelijk zijn zonder schemawijzigingen en datamanipulaties.

Voor het gemak gaan we uit van de volgende aannames:

- Alle voorbereidingen om gebruik te maken van editioning zijn reeds uitgevoerd. Deze worden later in dit artikel besproken.
- De applicatie zelf draait op twee applicatieservers, waarbij één server de oude en één server de nieuwe versie van de applicatie zal draaien.

Om te beginnen moet er een editie aangemaakt worden. Een editie vormt een nieuwe dimensie binnen de database waarin gebruik gemaakt wordt van *inheritance*. De gecreëerde editie is een *child* van de bestaande *parent*, wat betekent dat deze identiek is aan de parent en dat er wijzigingen gemaakt kunnen worden zonder dat deze invloed hebben op de parent. Op het moment dat er een *editionable* object (synonym, view en alle PL/SQL objecten)

binnen de nieuwe editie aangepast wordt, wordt deze *actualized* en bestaat het ook daadwerkelijk binnen de editie. Zonder deze aanpassing zou de versie van het object gebruikt worden zoals die is gedefinieerd binnen de parent.

Een tabel valt niet onder de eerder genoemde *editionable* objecten, daarmee is data ook niet geversioneerd. Om een tabel als het ware toch te kunnen versioneren is er binnen 11gR2 een nieuw type view geïntroduceerd, namelijk de editioning view.

Een editioning view vormt een soort buffer tussen de applicatiecode en het fysieke schema zelf. Een editioning view kan geen join of where clause bevatten en heeft alleen als functie om kolommen in een tabel wel of niet beschikbaar te maken binnen een editie.

Terugkijkend naar de requirements willen we het bestaande telefoonnummerveld splitsen in twee delen. Aangezien een tabel en de data niet geëditioneerd zijn kunnen we de bestaande kolom niet aanpassen, omdat dit ook impact zou hebben op de bestaande editie.

We zullen dus twee nieuwe kolommen aan de tabel toevoegen voor de landcode en het lokale nummer. Vervolgens passen we de editioning view aan in de nieuwe editie om de nieuwe kolommen beschikbaar en de oude kolommen onbeschikbaar te maken voor de applicatie.

Als onderdeel van de installatie zal uiteraard ook een DML-script opgeleverd moeten worden dat er voor zorgt dat de bestaande data ook beschikbaar is in de nieuwe kolommen. Dit kan uiteraard met een update statement gedaan worden, maar dit zou resulteren in een table-lock op de tabel, waardoor de installatie niet meer echt online is omdat alle updates die een gebruiker zou doen moeten wachten tot het update-script klaar is. In productiesituaties met honderden miljoenen records of nog meer is dit uiteraard onacceptabel en zal dit opgedeeld moeten worden in kleine sets. Bij een van onze klanten hebben we gebruik gemaakt van de standaard built-in DMBS_PARALLEL_EXECUTE package. Dit maakt het mogelijk om eenvoudig, snel en goed onderhoudbaar een dataset op te delen in meerdere stukken voor DML operaties.

Er voor het gemak van uitgaande dat de nieuwe kolommen niet in bestaande stored procedures, packages, triggers etc. gebruikt worden, zouden we nu gebruik kunnen maken van de nieuwe editie. Dit zou echter wel tot gevolg hebben dat wanneer een gebruiker die nog verbonden was met de oude editie, zijn wijzigingen enkel in de oude kolom kan doorvoeren. En wanneer gebruikers in de nieuwe editie bijvoorbeeld een nieuwe *employee* opvoeren, zou deze data onzichtbaar zijn voor gebruikers van de oude editie.

Cross-edition triggers

Om deze problemen op te lossen is er naast 'Editions' en 'Editioning views' nog een laatste type object toegevoegd, namelijk de 'CrossEdition Trigger'. Zo'n crossedition is er zowel in een 'forward' als een 'backward' uitvoering.

In ons geval hebben we in verband met de requirement voor fallback een backward editioning trigger nodig en voor de online requirement ook een forward editioning trigger. In de backward editioning trigger zorgen we ervoor dat bij het inserten of updaten van de gegevens van een employee de data uit de landcode en het lokale nummer minus de '0' aan elkaar geplakt en in de oude kolom geplaatst worden. In de forward editioning trigger splitsen we juist de in een kolom ingevoerde waarde in de twee velden met toevoeging van een voorloop '0'.

Met het creëren van de editioning triggers is de database installatie afgerond. Nu moet er nog wel gezorgd worden dat zowel de applicatieserver met de oude en de nieuwe versie aan de juiste editie verbinden.

Hiervoor is een aantal mogelijkheden:

- De editie per sessie aanpassen met behulp van het alter session commando. Dit zou bijvoorbeeld mogelijk zijn in de connection pool settings van de meeste applicatieservers of door het commando op te nemen in de applicatiecode. Dit laatste zou ik overigens afraden omdat ik van mening ben dat de applicatie zo min mogelijk versie-specifieke code moet bevatten, als alternatief kan er bijvoorbeeld gebruik gemaakt worden van separate configuratie files, LDAP of een andere soortgelijke optie.

- De default editie veranderen met behulp van het alter database commando. Dit betekent dat alle nieuwe connecties automatisch naar de nieuwe editie gaan. In ons voorbeeld is dit geen optie, want dit zou betekenen dat wanneer de oude applicatieserver een nieuwe connectie in de pool maakt, deze met de nieuwe editie verbonden wordt en dus zou kunnen verwijzen naar een niet meer bestaande kolom.
- Een extra service configureren. Sinds 11gR2 kan per service een editie opgegeven worden. In beide applicatieservers hoeft nu enkel de connect string naar de juiste service aangepast te worden.

Mijn persoonlijke voorkeur is om een combinatie van de tweede en derde mogelijkheid te gebruiken. Ervan uitgaande dat het bedoeling is om eerst applicatieserver 2 te upgraden en vervolgens op redelijk korte termijn ook applicatieserver 1, zodat het aantal actieve versies weer teruggebracht wordt naar één. In dit geval zou ik twee services aanmaken. De eerste met geen waarde voor editie, wat betekent dat de standaard editie gebruikt wordt. In de tweede service wordt de nieuwe editie geconfigureerd. Op het moment dat applicatieserver versie één ook geüpgraded wordt naar de nieuwe versie hoeft enkel nog de default editie in de database gezet te worden. Vervolgens begint de hele cyclus weer opnieuw.

In principe heeft EBR geen impact op de performance. Wanneer er gebruik gemaakt wordt van crossedition triggers kan dit uiteraard wel gevolgen hebben voor de performance. Mijn advies is daarom om, nadat een versie niet meer nodig is, op zijn minst de crosseditions en uiteindelijk ook de editie zelf te verwijderen.

Voorbereidingen

Zoals eerder aangegeven is er een aantal taken die uitgevoerd moeten worden voordat we tijdens een installatie gebruik kunnen maken van EBR.

- De applicatie mag geen rechtstreeks gebruik meer maken van de tabellen, alle DML moet via de editioning view. Het handigste is dan ook om alle bestaande tabellen te hernoemen en een editioning view aan te maken met de zelfde naam als van de tabel.
- Alle triggers op de tabellen moeten verwijderd en opnieuw aangemaakt worden, het zelfde geldt voor alle grants.
- Packages, stored procedures en functies hoeven enkel opnieuw gecompileerd te worden.
- Als laatste moeten alle schema's met objecten die editioned dienen te worden 'enabled' worden voor editioning. Dit geldt ook voor de schema's die private synonyms op editioned objecten hebben.

Indien twee schema's aan elkaars objecten refereren, zal het enablen van editioning falen omdat er dan objecten gebaseerd worden op een non-editioned object. Dit kan opgelost worden door FORCE te gebruiken. Pas hier echter wel mee op, omdat er bijvoorbeeld wanneer er in een van de tabellen een user-defined datatype gebruikt wordt, de tabel zelf invalid wordt.

Wat kan er niet?

Tot nu toe hebben we het alleen gehad over de mogelijkheden van EBR, er is echter ook een aantal beperkingen. Zo kan een editie maar één 'child' editie hebben en is branching (vooralsnog) onmogelijk. Vooralsnog, omdat bij het aanmaken van een editie optioneel aangegeven kan worden wat de naam van de parent editie is. Met de restrictie van enkel één 'child' is dit uiteraard altijd de laatst gemaakte editie. Persoonlijk hoop ik ook dat dit zo blijft, omdat de onderhoudbaarheid van een applicatie met meerdere childs op zijn minst erg lastig, zo niet onmogelijk wordt.

Een non editioned-object mag niet refereren aan een editioned object. Zo kan een function based index niet op geëditioneerde functie gebaseerd zijn.

Naar mijn mening is het met EBR mogelijk om 80 tot 90 % van alle wijzigingen online te doen. Wel denk ik dat binnen ieder project moet worden gezocht moet worden naar de balans tussen eventuele extra complexiteit om

online deployment en fallback mogelijk te maken versus het additionele risico op fouten, wat extra complexiteit met zich mee brengt.

Conclusie

Ik kan me voorstellen dat in complexe omgevingen, waarin meerder databases en applicaties op allerlei mogelijk manieren met interfaces aan elkaar hangen, het implementeren van EBR een behoorlijke uitdaging zal zijn - mede door de additionele testcapaciteit die benodigd is. Er moet namelijk niet alleen getest worden of de nieuwe versie van de applicatie nog juist werkt, maar ook of na installatie van de nieuwe versie de oude nog correct werkt. Indien fallback een van de requirements is, zal ook data die reeds ingevoerd of aangepast is in de nieuwe versie, na een fallback nog steeds bruikbaar moeten zijn met de oude versie.

Ook zal goed nagedacht moeten worden over onderhoudbaarheid. Een functioneel beheerder moet bijvoorbeeld eenvoudig kunnen zien op welke versie een gebruiker met een probleem actief was.

Ondanks de hierboven genoemde uitdagingen, die vooral op het organisatorische vlak liggen, is EBR technisch relatief eenvoudig te implementeren. Dat maakt naar mijn mening EBR zeer interessant voor alle organisaties waar hoge beschikbaarheid en flexibiliteit van opleveringen van belang is, omdat een beschikbaarheid van bijna 100% zeker mogelijk is.

Referentie: Oracle EBR White Paper - http://www.oracle.com/technetwork/database/features/availability/edition-based-redefinition-1-133045.pdf

John van den Hurk is Consultant bij Qualogy, Oracle Certified Partner en gespecialiseerd in Oracle, Java en Agile en Web 2.0 (http://www.qualogy.com).