Oracle Warehouse Builder

Scripten of Gui?

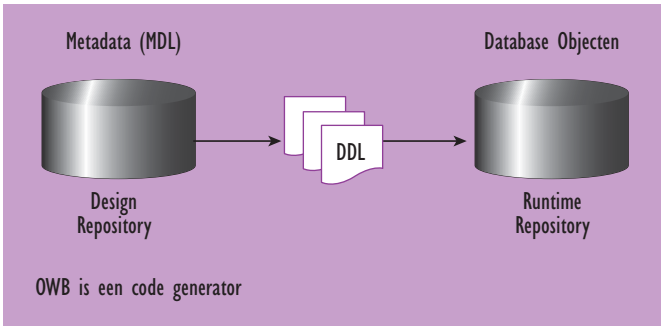
In een tijd waarin programmeren steeds gemakkelijker wordt gemaakt, door de komst van ontwikkeltools met grafische gebruikersinterface, kan Oracle niet achterblijven. Oracle heeft voor de ontwikkeling en het beheren van datawarehouses en de bijbehorende ETL-processen (extractie-, transformatie-, en laden) Oracle Warehouse Builder (OWB) op de markt gebracht.

In OWB is het bouwen van database packages sterk vereenvoudigd tot het selecteren en slepen van objecten op je ontwerpvlak en door middel van het trekken van lijnen tussen de objecten komen extractie en laadprocessen tot stand.
Natuurlijk blijft er voor ingewikkelde transformaties nog wel behoefte aan het typen van eigen code, wat in OWB ook mogelijk is, maar voor zover dat kan levert Oracle zoveel mogelijk standaardfuncties en transformaties mee als klikbare objecten.
Hoewel OWB versie 10.2 reeds op de markt is verschenen wordt in dit artikel OWB versie 10.1 tegen het licht gehouden. Dit vanwege het simpele feit dat veel projecten nog niet zijn begonnen met de laatste OWB-versie. Voor de projecten die al wel zijn gestart met OWB-versie 10.2 is de inhoud van dit artikel waardevol als het woord 'runtime repository' wordt vervangen door 'Control Center'.

Architectuur OWB

OWB maakt gebruik van een design repository voor de opslag van metadata over tabellen, sequences, dimensies, feitentabellen, transformaties (mappings), procedures en functies. De objecten zelf bestaan echter alleen in de runtime repository. OWB-code genereert namelijk scripts voor het creëren van database-objecten, bijvoorbeeld tabellen, procedures, packages of sequences. Pas wanneer deze scripts worden uitgevoerd via de runtime repository ontstaan de objecten in de database.

De design repository en de runtime repository kunnen verschillende database-instanties zijn of verschillende schema's binnen één database-instantie. De keuze voor de opzet voor de architectuur is hoofdzakelijk afhankelijk van de manier waarop een organisatie haar OTAP (ontwikkel-, test-, acceptatie- en productie-)-omgeving heeft ingericht. Bij een infrastructuur waarbij de verschillende databases voor ontwikkel, test, acceptatie en productie strikt gescheiden zijn, door bijvoorbeeld een firewall, ontkomt men er niet aan om per omgeving (O, T, A of P) één database-instantie te maken met daarin één schema voor de design repository en één of meerdere schema's voor de runtime repository's (ofwel targetschema's).

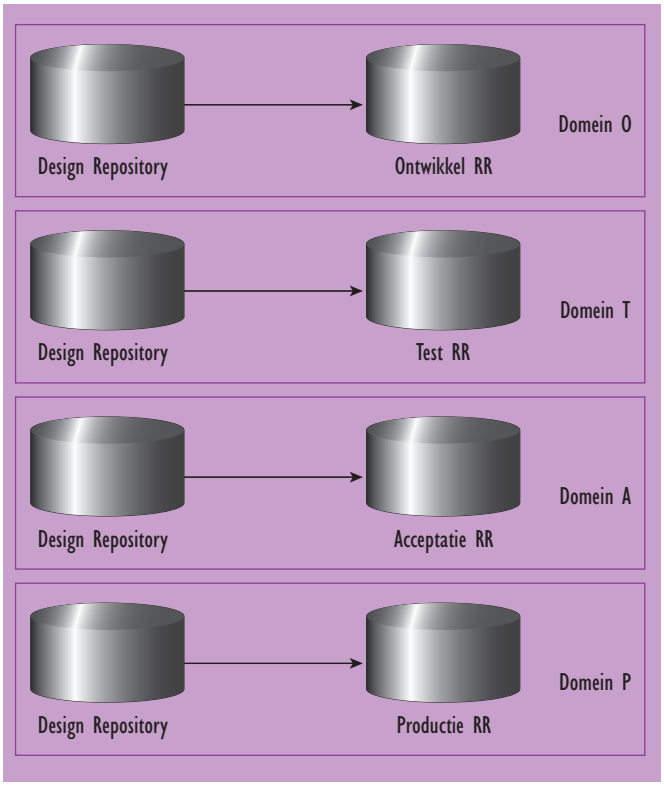


Afbeelding 1. Globale architectuur van Warehouse Builder.

Als database-omgevingen niet strikt gescheiden zijn, is het ook mogelijk om slechts één design repository in een aparte database-instantie in te richten en vervolgens elke runtime repository ieder in een eigen database-instantie in te richten. Vanuit één design repository is het ook mogelijk om op verschillende runtime-repository's de scripts van de metadata uit te voeren.

OMBPlus en OMB scripting

Vanaf OWB versie 9i wordt er bij OWB ook een op TCL (spreek uit als tickel) gebaseerde scripting taal meegeleverd onder de naam OMB Scripting, wat staat voor Oracle Metabase language. OMBPlus is de interface waarin de OMB-scripts uitgevoerd dienen te worden. In principe kan er via de scripts hetzelfde bewerkstelligd worden wat met de GUI-variant van OWB kan. Metadata in de design-repository kunnen worden



Afbeelding 2. Architectuur waarbij de domeinen van OTAP gescheiden zijn.

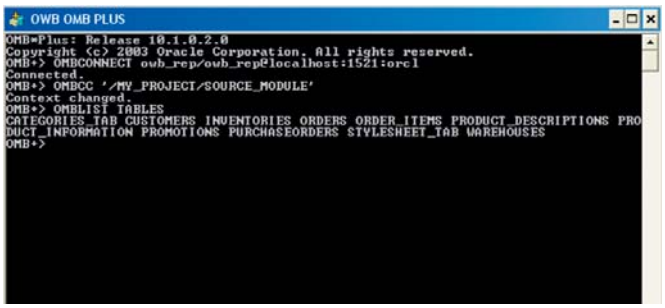
aangepast en ook het deployen van de metadata op de runtime repository is mogelijk.

Scripten of de GUI?

Als de grafische interface van OWB ontwikkelaars het programmeren zoveel makkelijker maakt, waarom zouden we dan nog gaan scripten? Voor de overzichtelijkheid van datastromen en voor niet repeterende taken is de grafische interface duidelijk het ideale hulpmiddel, maar in sommige situaties verdient scripting toch de voorkeur.

Efficiëntie

Door repeterende handelingen in een script op te nemen hoeven deze slechts eenmaal gedefinieerd te worden om vervolgens deze handelingen meerdere malen over een reeks van



Afbeelding 3. De OMB Plus interface om OMB script in uit te voeren.

objecten uit te voeren. Een voorbeeld: in de staging-module is een reeks tabellen gedefinieerd en op een bepaald moment dienen deze tabellen voorzien te worden van twee kolommen voor de verwerkingsdatum en de verwerkingscode. In plaats van alle tabellen te openen en te bewerken via de grafische gebruikersinterface kan dit heel eenvoudig bewerkstelligd worden via een script. Via een loop-constructie worden de extra kolommen aan alle tabellen toegevoegd.

```
Foreach tabelnaam [OMBLIST TABLES] {
  OMBALTER TABLE '$tabelnaam' \
  ADD COLUMN 'VERWERK_DATUM' SET PROPERTIES (DATATYPE) VALUES ('DATE')
\
  ADD COLUMN 'VERWERK_CODE' SET PROPERTIES (DATATYPE, LENGTH) \
    VALUES ('VARCHAR2', 30)

  puts "Tabel $tabelnaam is uitgebreid";
}
```

Versiebeheer

Het uitrollen van nieuwe releases van het datawarehouse op de diverse omgevingen is een proces wat zich elke keer weer herhaalt en zich dus goed leent voor automatisering door middel van OMB-scripting. Niet alleen het installatieproces wordt hierdoor geautomatiseerd, maar ook het proces voor versiebeheer wordt eenduidiger. Te allen tijde is te controleren welke versie op welke wijze geïnstalleerd is.

Voordelen

- Installatie is eenduidig en op elke omgeving exact hetzelfde.
- Een installatie wordt uitgevoerd middels een script, hierdoor is het niet noodzakelijk dat installateurs kennis hebben van specifieke ontwikkeltools, namelijk OWB.
- De installatie is reproduceerbaar.

Het distribueren van het metadatamodel gebeurt via een exportfunctie in OWB waarmee een MDL (metadata language) bestand wordt gegenereerd. Dit MDL-bestand kan vervolgens in een andere design repository geïmporteerd en uitgevoerd worden. Het deployen van de objecten op de runtime-omgeving is een activiteit die in een specifieke volgorde dient uitgevoerd te worden. Een mapping wordt namelijk invalid als de bron- en doeltabellen, die binnen de mapping gebruikt worden, nog niet bestaan in de runtime-repository. Tabellen dienen dus eerst gecreëerd te worden in de runtime repository, evenals gebruikte sequences, functions en procedure. Pas daarna kan de mapping met succes gecompileerd en gecreëerd worden. Dit hele proces kan in OMB gescript worden, zodat objecten in de goede volgorde gecreëerd worden.

Voorbereiding export metadata

Als eerste wordt de design repository op de ontwikkelomge-

ving bevroren en wordt de projectomschrijving aangepast met het nieuwe versienummer en datum van de versie. Vervolgens wordt de export van de te distribueren repository gemaakt in een MDL-bestand.

Stap 1- Verbinding maken met de Repository

Met het `OMBCONNECT` commando wordt een verbinding gemaakt met de repository. Variabelen die gezet zijn, kunnen aangeroepen worden door een $ voor de variabele naam te zetten. Het is tevens handig om de logging aan te zetten en dit weg te schrijven naar een logbestand.

```
##  Connect naar de OWB repository
  OMBCONNECT $rpusr/$rppwd@$uxhost:1521:$rpdbname USE REPOSITORY
$reposnm ;

##  Zet de omgevingsvariabelen.
  set OMBLOG $logbestand;                # -- Zet logging aan.
  set OMBPROMPT 1;                       # -- Zet de prompt aan.
Wijzigt per context.
```

Stap 2- Instellen van een nieuw versienummer

Met het commando `puts` en `gets` kan respectievelijk tekst naar het scherm (= stdout) worden geschreven c.q. ingetypte karakters (= stdin) in een variabele worden opgeslagen. Vervolgens kan met het commando `OMBALTER PROJECT` de eigenschappen van het project gewijzigd worden. Met het `OMBCOMMIT` commando worden wijzigingen in de repository opgeslagen.

```
##  Vraag om de versienaam van het project.
  puts -nonewline stdout "Geef versienaam Project (versie_1.0_ddmmjj):
"; gets stdin vnaam;

##  Geef het project een nieuwe versie in het description veld. Zelfde
als het bestandsnaam.
  OMBALTER PROJECT 'STUURHUT1_PROJECT' SET PROPERTIES (DESCRIPTION)
VALUES ('$vnaam') ;

  OMBCOMMIT;
```

Stap 3- Exporteren van de design respository naar een MDL bestand

```
##  Exporteer de OWB Meta data file (MDL).
  OMBEXPORT TO MDL_FILE 'I:\\$vnaam.mdl' PROJECT 'STUURHUT1_PROJECT'
COMPONENTS (PROJECT 'STUURHUT1_PROJECT') OUTPUT LOG TO 'I:\\$vnaam.
log';

## Disconnect Repository
  OMBDISCONNECT;
```

Script voor distribueren van metadata-model

Stap 1 - Variabelen instellen

Aan het begin van het script worden de variabelen ingesteld. Deze variabelen hebben betrekking op de locatie van het log-bestand, de naam van de host, gebruikersnamen en wachtwoorden van de verschillende schema's in de databases en de namen van de gebruikte Oracle sid's en ook de bestandsnaam van het metadata-bestand. Deze zouden natuurlijk ook hard gecodeerd kunnen worden in het script, maar om veiligheidsredenen gebeurt dat in het hier getoonde script hieronder niet. In plaats daarvan is gekozen om aan het begin van de installatie de gebruiker om deze gegevens te vragen.

```
  set logbestand //install//Installatie.log; # -- Logbestand van dit
proces.

## -- SET Hostname Unix machine
  puts " ***** UNIX gegevens: *****";
  puts -nonewline stdout "Geef unix hostnaam :<ENTER> =>  "; gets stdin
uxhost;

## -- SET Unix Username en Password
  set uxusr userunix;                                          ##
Username op unix
  puts -nonewline stdout "Geef password van unix gebruiker 'userunix'
:<ENTER> =>  "; gets stdin uxpwd;

## -- SET Repository gegevens, database name, username, password
puts -nonewline stdout "Geef repository database name (SID) :<ENTER> =>
"; gets stdin rpdb;
puts -nonewline stdout "Geef repository name (orcl) :<ENTER> =>  ";
gets stdin reposnm;
puts -nonewline stdout "Geef repository gebruikersnaam :<ENTER> =>  ";
gets stdin rpusr;
puts -nonewline stdout "Geef repository gebruikers password :<ENTER> =>
"; gets stdin rppwd; set reposnm 'reposnm';

## -- SET Runtime repository gegevens, database name, username,
password
puts -nonewline stdout "Geef Runtime database name (SID) :<ENTER> =>
"; gets stdin rtsid;
puts -nonewline stdout "Geef Runtime gebruikersnaam (OWB_
Runtime):<ENTER> =>  "; gets stdin rtusrn;
puts -nonewline stdout "Geef Runtime gebruikers password :<ENTER> =>
"; gets stdin rtpwd;
puts -nonewline stdout "Geef Runtime Owner naam :<ENTER> =>  "; gets
stdin rtownr;

## -- SET Staging database gegevens, database name, username, password.
Doe hetzelfde voor elke
## -- target locatie zoals DHW en Datamart.
set dsaschema STAGE;                              # -- Schema (user)
STAGE.
puts -nonewline stdout "Geef (STAGE schema) password van 'STAGE'
:<ENTER> =>  "; gets stdin dsapwd;

## -- SET Runtime Workflow, database name, username, password
puts -nonewline stdout "Geef Username van Workflow manager (owf_mgr)
:<ENTER> =>  "; gets stdin wfusrn;
```

```
puts -nonewline stdout "Geef Password van Workflow manager (owf_pwd)
:<ENTER> =>  "; gets stdin wfpwd;

## -- SET Metadata bestandsnaam.
puts -nonewline stdout "Geef naam van mdl bestand: <ENTER> =>  "; gets
stdin bnaam;
```

Stap 2- Installeren en bijwerken repository

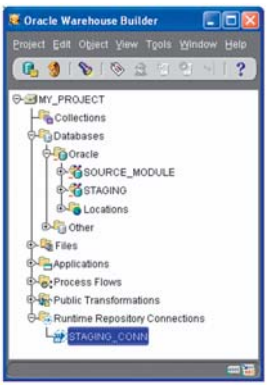
Met OMBCONNECT commando wordt een verbinding gemaakt met de design repository. In de aanroep van de verbinding, worden variabelen uit stap 1 gebruikt om de verbinding te kunnen maken.
OMBIMPORT wordt gebruikt om vanuit het MDL bestand een design repository op te bouwen gevolgd door OMBCOMMIT om de wijzigingen te bevestigen.
Met behulp van USE REPLACE_MODE kan aangegeven worden of de repository aangevuld of overschreven moet worden of dat alleen nieuwe dingen toegevoegd mogen worden. Als geen importmethode is opgegeven, betekent dit een nieuw project maken (CREATE_MODE). Zie voor meer mogelijke importmethodes de *scripting reference*.

```
OMBCONNECT $rpusr/$rppwd@$uxhost:1521:$rpdb USE REPOSITORY $reposnm ;

OMBIMPORT FROM MDL_FILE '//install//$bnaam' USE REPLACE_MODE OUTPUT LOG
TO '//install//log//$bnaam.log';
OMBCOMMIT;
```

Stap 3- Instellen Runtime Repository Connection

Om te kunnen deployen moet er vanuit de design repository een verbinding zijn naar de runtime omgeving. Deze verbinding heet de runtime repository connection (zie afbeelding 4).



Afbeelding 4. De runtime connection naar staging.

In de geïmporteerde design-repository bevinden zich nog de verwijzingen (verbindingen) naar de runtime repository van de ontwikkelmachine en moet aangepast worden voor de nieuwe runtime repository. Het OMBALTER RUNTIME_REPOSITORY_CONNECTION commando bewerkstelligt dit. Elke handeling die moet worden uitgevoerd in het script moet in de juiste context plaatsvinden. Een context is te vergelijken met een directory in een directory-tree. Met OMBCC (OMB change context) wordt de context en dus van locatie in de tree gewijzigd. Met OMBDCC (OMB Display Context) kan de huidige locatie in de tree opgevraagd worden.

```
OMBCC '/MY_PROJECT';

##  Wijzig Runtime connection naar de juiste omgeving.
OMBALTER RUNTIME_REPOSITORY_CONNECTION 'STAGE_CONN' \
  SET PROPERTIES (DESCRIPTION, BUSINESS_NAME, HOST, PORT, \
  SERVICE_NAME, CONNECT_AS_USER, RUNTIME_REPOSITORY_OWNER) \
  VALUES ('Runtime connectie', 'STAGE_CONN', '$uxhost', 1521, '$rtsid',
'$rtusrn', '$rtownr');
puts "Runtime Connectie is aangepast aan huidige omgeving";

OMBCOMMIT;

## Connecteer met runtime omgeving.
OMBCONNECT RUNTIME 'STAGE_CONN' USE PASSWORD '$rtpwd';

##  Registreer locaties.
puts "Registreer Staging locatie - STAGING_LOC";
catch {
          OMBREGISTER LOCATION 'STAGING_LOC' \
            SET PROPERTIES (HOST,PORT, SERVICE, SCHEMA, PASSWORD) \
            VALUES ('$uxhost', 1521,'$rtsid','$dsaschema', '$dsapwd');
} result; puts "STAGING locatie registratie compleet"; puts "$result";
OMBCOMMIT;
```

Stap 4- Deployment Action Plan

Het Deployment Action Plan geeft aan welke objecten op welke manier gedeployed moeten worden. Het is te vergelijken met wat er in de OWB Deployment Manager Tool gebeurt. Per object wordt aangegeven of het verwijderd (DROP) moet worden of juist gecreëerd (CREATE) moet worden. Het toevoegen van objecten aan het Deployment Action Plan kan alleen vanuit de context waarin het script zich op dat moment bevindt. Voorwaarde voor het maken van een Deployment Action Plan is dat verbinding is gemaakt met de Runtime Repository Connection.

OMBCREATE TRANSIENT DEPLOYMENT_ACTION_PLAN 'naam_plan' hiermee wordt een nieuw action plan gecreëerd. Hierin worden vervolgens de acties opgeslagen.
OMBALTER DEPLOYMENT_ACTION_PLAN 'naam plan' hiermee worden objecten die gedeployed moeten worden toegevoegd aan het plan.
Met ADD ACTION '$mappingname.DROP' geef je de actie een naam. Met SET PROPERTIES (OPERATION) VALUES ('DROP') wordt aangegeven dat het hier een verwijder instructie betreft.
SET REFERENCE MAPPING '$mappingname'; geeft aan welk object in dit geval verwijderd moet worden.

Het uitvoeren van een actionplan gaat via het commando: `OMBDEPLOY DEPLOYMENT_ACTION_PLAN` naam_plan';

Met `OMBDROP DEPLOYMENT_ACTION_PLAN` 'naam_plan' wordt het action plan weer verwijderd en dus leeggemaakt.

Het spreekt voor zich dat na elke wijziging een Commit-statement moet worden gegeven.

```
OMBCC '/MY_PROJECT/STAGING';   ## -- Juiste context zetten !!

## --  DROP MAPPINGS STAGING AREA ----------------
OMBCREATE TRANSIENT DEPLOYMENT_ACTION_PLAN 'STAGING_DROP';

foreach mappingname [OMBLIST MAPPINGS] {
    catch {
    OMBALTER DEPLOYMENT_ACTION_PLAN 'STAGING_DROP' ADD ACTION '$map-
pingname.DROP' \
    SET PROPERTIES (OPERATION) VALUES ('DROP') SET REFERENCE MAPPING
'$mappingname';
    } result; puts "$mappingname toegevoegd aan Deployment action plan
$result";
}
OMBCOMMIT;

## --  DEPLOY DROP MAPPINGS STAGING AREA ---------
catch {
OMBDEPLOY DEPLOYMENT_ACTION_PLAN 'STAGING_DROP';
} result; puts "Drop MAPPINGS in STAGING: $result";
OMBCOMMIT;

OMBDROP DEPLOYMENT_ACTION_PLAN 'STAGING_DROP';
OMBCOMMIT;
## --  END DROP MAPPINGS DWH PACKAGE ------------
```

OMB Script: primitief, maar effectief

Foutafhandeling en reproduceerbaar maken van het script

TCL is een primitieve programmeertaal. Als er een fout optreedt tijdens het uitvoeren van het script zal het script er in zijn geheel mee stoppen. Om te voorkomen dat het script bij een fout stopt, moeten fouten afgevangen worden. Voor het afvangen van fouten wordt het `catch { <statement A> } result;` gebruikt, waarbij `<statement A>` staat voor de uit te voeren code die in de fout zou kunnen gaan. Als een fout optreedt wordt deze opgevangen in de variabele result.

Lijsten opvragen en loop constructie

De script-taal OMBPlus voorziet ook in het opvragen van lijsten van objecten vanuit een bepaalde context. Bij het opvragen van een lijst objecten van een bepaald type moet het type altijd in het meervoud staan. Dus… mapping wordt mappings. Met het commando `OMBLIST <MEERVOUD OBJECTNAAM>` kun je een lijst van objecten opvragen die in die context aanwezig is. (zie ook de Scripting Reference)

```
OMBLIST MAPPINGS
```

TCL maakt het mogelijk om een lijst te doorlopen en telkens dezelfde acties uit te voeren op steeds een volgend element uit de lijst. Hiervoor wordt het commando foreach variabelennaam lijstnaam { statements } gebruikt. Een combinatie van het opvragen van een lijst en de loop instructie levert dynamische code op. De code zegt dan voer voor elk element in de lijst uit, de statements 1 tot n.

```
foreach mappingname [OMBLIST MAPPINGS] {
    < STATEMENTS 1 tot n > ;
}
```

Conclusie

Is de grafische gebruikersinterface van OWB efficiënter en effectiever in gebruik dan scripting tijdens de bouwfase van een datawarehouse? Die vraag kunnen we als volgt beantwoorden: OMB-scripting is zeer effectief voor het uitvoeren van repeterende taken en kan productiviteit van de bouwfase van een datawarehouse aanzienlijk verhogen. Daar tegenover staat dat datastromen veel overzichtelijker in beeld worden gebracht in de grafische gebruikersinterface, dan wat met de OMBPlus-interface mogelijk is. Zeer complexe transformaties zijn eenvoudiger te implementeren via de grafische gebruikersinterface. Om de kwaliteit van de verschillende versies van het metadatamodel uit de design repository goed te kunnen borgen en herhaalbaarheid en reproduceerbaarheid van distributies te garanderen, is OMB scripting onontbeerlijk.

Ilona Tielke is Senior BI Consultant bij IT-eye.