Installatie van Oracle applicatiesoftware

Methode voor correcte procedure

U herkent vast wel de problemen die je helaas te vaak aantreft bij de bouw en installatie van applicatiesoftware bestaande uit Oracle PL/SQL, Oracle Forms en Oracle Reports. In dit artikel poogt de auteur een antwoord te bieden op de situatie die hij meerdere malen heeft aangetroffen op projecten: een grote hoeveelheid scripts, Oracle Forms, Oracle Reports en al dan niet attached libraries zonder perfecte installatiehandleiding. Hij beschrijft een generieke methode die kan worden gebruikt onafhankelijk van de versie van Oracle software.

Er zijn talloze voorbeelden van veel voorkomende problemen. Deze problemen geven aan dat het Software Configuratie Management niet de gewenste kwaliteit heeft:

- Er is geen (goede) installatiehandleiding.
- Het maken van de installatiehandleiding kost veel tijd.
- De installatieprocedure is grotendeels handmatig en daardoor foutgevoelig.
- De PL/SQL scripts worden in een verkeerde volgorde uitgevoerd waardoor een refererende sleutel ontbreekt.
- Grants of synoniemen ontbreken bij een installatie voor meerdere schema's.
- Er zijn verkeerde versies van packages geïnstalleerd doordat upgrade scripts eerder zijn gedraaid dan de scripts waarin de oudere versies stonden gedefinieerd.
- Er ontbreken reference forms of attached libraries tijdens de bouw van Oracle Forms modules.
- Plaatjes ontbreken in Oracle Reports modules.

De MAKE utility is een Unix utility om programma's te bouwen uit broncode-bestanden geschreven in C

Software Configuratie Management

Software Configuratie Management betreft het organiseren en bewaken van wijzigingen in ontwikkelde componenten van een softwaresysteem. Door effectief met SCM om te gaan is het mogelijk om op ieder gewenst moment een consistente en uniek identificeerbare versie van het systeem te achterhalen.

Software Configuratie Management kent de volgende drie aspecten:

- *Versiebeheer* betreft het beheren en onderhouden van meerdere versies van een bestand in een archief. Het archief bevat elke wijziging die het bestand heeft ondergaan.

- *Configuratie* en *Release Management* betreft het samenstellen, uit de individuele archieven van de bestanden, van een consistente, uniek identificeerbare, versie van het systeem als geheel.

- *Change Management* betreft het bewaken en beheren van wijzigingen die in de software worden geïntroduceerd.

In figuur 1 wordt, door het repeterende karakter, het belang van een perfecte installatie geïllustreerd.

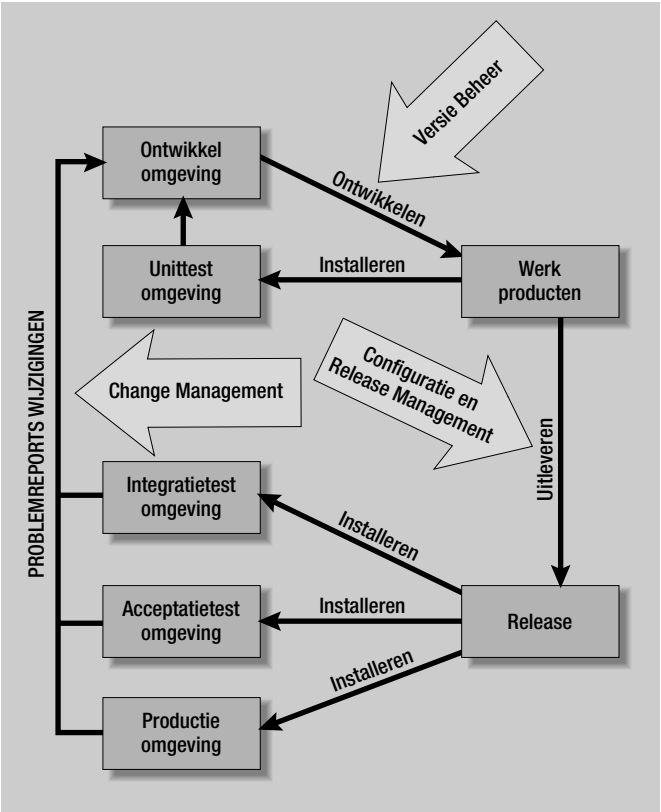
De Performance Monitor applicatie

Om de generieke installatiemethode te illustreren neem ik als voorbeeld de Performance Monitor applicatie. Transfer Solutions stelt de applicatie gratis beschikbaar (zie Referenties). De Performance Monitor verzamelt periodiek informatie over het functioneren van één of meer Oracle-databases. De monitor beantwoordt de volgende vragen, welke SQL-opdrachten worden uitgevoerd en hoeveel resources worden gebruikt? Met deze informatie bouwt de Performance Monitor een historie op die gebruikt wordt om trends te analyseren. Met deze trends kunnen we de oorzaken van problemen in de database(s) aanwijzen.

Een onderdeel van de Performance Monitor is de Total Performance Index. De applicatie bestaat uit:

- 14 tabellen
- 1 view

- 2 sequences
- 5 packages
- 1 menu
- 2 forms
- 2 libraries
- 1 reference form
- 12 rapporten



Figuur 1. Elementen van Configuratie Management

Database objecten

Een installatie van applicatiesoftware op de database bestaat uit het aanmaken van objecten, rollen, synoniemen en het uitdelen van grants (DDL statements). Voor de rest van dit artikel schaar ik rollen, synoniemen en grants onder de database objecten. Ik ga ervan uit dat tablespaces en gebruikers zijn aangemaakt met voldoende privileges om aan te loggen en objecten aan te maken. De volgende objecten zijn voor de installatie van belang: sequences, tabellen, constraints, indexen, functies, packages met hun bodies, procedures, views, triggers, rollen, synoniemen en grants.

Eisen voor een database installatie

Wat zijn nu de eisen voor een correcte database installatie?

1. Alle objecten moeten aangemaakt worden.
2. Als een object door meerdere scripts kan worden aangemaakt, dan moet het nieuwste script het laatst worden uitgevoerd.
3. Er dienen zo min mogelijk Oracle foutmeldingen op te treden, oftewel de afhankelijkheden tussen objecten dicteren de volgorde van uitvoeren van scripts.
4. De installatie moet na een onverwachte foutmelding kunnen stoppen om later vanaf dat punt te kunnen herstarten opdat zo min mogelijk tijd verloren gaat.
5. De installatie moet ongedaan kunnen worden gemaakt, waarbij alleen de objecten die zijn aangemaakt, worden verwijderd.
6. Eventuele hulpmiddelen moeten vrij beschikbaar zijn in verband met de acceptatie van de methode (managers zijn gevoelig voor dit argument).

Afhankelijkheden

Alles in ogenschouw nemend zijn er twee soorten afhankelijkheden:

1. Tussen objecten en scripts.
2. Tussen objecten onderling.

Afhankelijkheden tussen database objecten en scripts

Elk database object wordt gemaakt door een script. Wat we nodig hebben is een geautomatiseerde manier om te achterhalen welke objecten door welk script gemaakt worden. Met de script taal Perl (http://www.perl.org) kunnen we SQL*Plus scripts scannen op DDL statements.

Afhankelijkheden tussen database objecten

De Oracle dictionary biedt ons deze informatie via de ALL_OBJECTS, ALL_SYNONYMS, ALL_DEPENDENCIES, ALL_CONSTRAINTS en ALL_TAB_PRIVS views. Wel is er sprake van een kip en ei probleem: om de afhankelijkheden te bepalen, moet de installatie zijn uitgevoerd en om de installatie uit te voeren is informatie rond de afhankelijkheden nodig om de volgorde correct te bepalen. Een oplossing is om bij de allereerste installatie de scripts meerdere malen uit voeren in volgorde van creatiedatum (oudste eerst).

Implementatie

De implementatie is als volgt: er is een doel, namelijk maak alle objecten, en dat doel wordt bereikt door de SQL*Plus scripts in de juiste volgorde uit te voeren. Deze volgorde is handmatig te bepalen, maar foutgevoelig en niet uitdagend. Daarom is er een geautomatiseerde manier nodig om de scripts in de juiste volgorde uit te voeren.

MAKE

Voor de oudgedienden zal de MAKE utility bekend in de oren klinken. De MAKE utility is van oorsprong een Unix utility om programma's te bouwen uit broncodebestanden geschreven in de taal C. De stappen zijn: compileer de C code tot objectbestanden en link deze samen tot een programma. Dit dient zo efficiënt mogelijk te gebeuren. Dus indien slechts één bronco-

debestand wijzigt, dan moet alleen dat bestand gecompileerd worden om vervolgens de objectbestanden te linken tot een programma. MAKE regelt dit aan de hand van een definitiebestand (makefile) en de timestamps van bestanden. Als bestand f1 afhangt van f2 en f2 is nieuwer dan f1, dan voert MAKE het commando uit om f1 (opnieuw) te bouwen. Een voorbeeld van een MAKE definitiebestand (een makefile) dat gebruikt kan worden om een programma myprog.exe aan te maken aan de hand van het codebestand myprog.c:

```
OBJS = myprog.obj

myprog.exe: $(OBJS)
     $(LINK) $(LDFLAGS) $(OBJS)

myprog.obj: myprog.c
     $(CC) $(CFLAGS) -c myprog.c
```

De makefile bestaat uit definities van MAKE variabelen (OBJS) en een aantal regels om een target (bestand) te bouwen met een commando aan de hand van zijn afhankelijkheden. We zien dus dat myprog.exe afhangt van myprog.obj, die op zijn beurt afhangt van myprog.c. De MAKE variabelen LINK, LDFLAGS, CC, CFLAGS zijn standaard gedefinieerd en verwijzen naar de naam van de link utility, link opties, naam van de C compiler en compiler opties.

De volgende aanroep zorgt ervoor dat myprog.exe wordt gebouwd aan de hand van de regels in de makefile:

```
make -f makefile myprog.exe
```

Voor onze methode maken we gebruik van de GNU MAKE utility (http://www.gnu.org/software/make/make.html). Deze is uitgebreider dan de originele MAKE utility en is te gebruiken op Windows en Unix platforms.

In de Unix wereld is het gebruikelijk om voor het maken van een programma de build dependencies te bepalen

Uitvoeren van SQL*Plus scripts

De uitvoering van SQL*Plus scripts is via een MAKE regel te definiëren. Om niet voor elk SQL*Plus script een aparte regel op te nemen, maken we gebruik van een regel waarbij alleen de extensies van bestandsnamen aangeven hoe een doel gemaakt wordt. Het voorbeeld met myprog.exe bevat een regel met de volledige bestandsnaam. Deze heeft voorrang boven regels op basis van extensies. Een makefile voorbeeld om SQL*Plus scripts eindigend op extensie .sql uit te voeren:

```
SQLPLUS = sqlplus

# Silent start-up
SQLPLUSFLAGS = -s

SQLPLUSCMD := echo exit SQL.SQLCODE | $(SQLPLUS) $(SQLPLUSFLAGS) $(USERID)

.SUFFIXES: .sql-run .sql

.sql.sql-run:
       $(SQLPLUSCMD) @ $< $(SQLPLUSPARAMS)
```

De regel .SUFFIXES geeft aan dat bestanden met als naam <stam>.sql-run gemaakt wordt indien een bestand met naam <stam>.sql bestaat.

De regel .sql.sql-run: is een regel gebaseerd op de extensie van bestandsnamen, waarbij $< het afhankelijke bestand is (in dit geval het SQL*Plus script).

Het MAKE commando:

```
make -f makefile pm.sql-run USERID=pm_owner/pm_owner
```

zal dan het SQL*Plus opstarten voor pm_owner met als script pm.sql:

```
echo exit SQL.SQLCODE | sqlplus -s pm_owner/pm_owner @ pm.sql
```

De constructie echo exit SQL.SQLCODE | sqlplus zorgt ervoor dat SQL*Plus na het uitvoeren van het script de volgende input krijgt: exit SQL.SQLCODE, oftewel SQL*Plus stopt en geeft de foutcode terug. Zonder echo exit SQL.SQLCODE | moet de gebruiker zelf exit invoeren of exit opnemen aan het einde van het script. De foutcode SQL.SQLCODE is belangrijk om MAKE te kunnen laten stoppen indien het laatste SQL statement een foutcode genereert. Om nu te stoppen met uitvoering van een SQL*Plus script zodra er een SQL fout optreedt, dan moet de SQL*Plus opdracht WHENEVER SQLERROR EXIT FAILURE gegeven worden. Als deze opdracht wordt opgenomen in een script login.sql in de directory waar men MAKE opstart, dan zal automatisch gestopt worden bij een SQL fout, omdat SQL*Plus bij opstarten eerst het script login.sql uitvoert, indien dit gevonden kan worden.

Afhankelijkheden tussen database objecten en scripts

De tabel pm_sqlarea van de Performance Monitor applicatie wordt aangemaakt door het script pm_sqlarea.tab.
Dit script bevat de volgende regels:

```
CREATE TABLE pm_sqlarea(
db                              VARCHAR2(20)       NOT NULL,
 first_load_time                DATE               NOT NULL,
```

Dus tabel pm_sqlarea voor schema pm_owner wordt aangemaakt door uitvoering van pm_sqlarea.tab-run.

In MAKE taal wordt dat:

```
install.table.pm_owner.pm_sqlarea: pm_sqlarea.tab-run
```

Afhankelijkheden tussen database objecten

De query

```
select name, type, referenced_name, referenced_type
from all_dependencies
where owner = 'PM_OWNER'
and referenced_owner not in ( 'SYS', 'SYSTEM', 'PUBLIC' )
order by name, type, referenced_name
```

laat onder andere het volgende zien:

NAME	PM
TYPE	PACKAGE
REFERENCED_NAME	PM_RUN
REFERENCED_TYPE	TABLE
NAME	PM
TYPE	PACKAGE BODY
REFERENCED_NAME	PM
REFERENCED_TYPE	PACKAGE
NAME	PM
TYPE	PACKAGE BODY
REFERENCED_NAME	PM_RUN_SEQ
REFERENCED_TYPE	SEQUENCE
NAME	PM
TYPE	PACKAGE BODY
REFERENCED_NAME	TRC
REFERENCED_TYPE	PACKAGE

Tabel 2. ALL_DEPENDENCIES

We zien dat de package specificatie PM afhangt van de tabel PM_RUN. De package body PM hangt af van de package specificatie zelf, van sequence PM_RUN_SEQ en van package specificatie TRC.

In MAKE taal wordt dat:

```
install.package.pm_owner.pm: install.table.pm_owner.pm_run

install.package_body.pm_owner.pm: install.package.pm_owner.pm

install.package_body.pm_owner.pm: install.sequence.pm_owner.pm_run_seq

install.package_body.pm_owner.pm: install.package.pm_owner.trc
```

Merk op dat het schema wordt opgenomen om installaties over meerdere schema's uit te kunnen voeren.
Voor de de-installatie is de volgorde omgekeerd.

Een voorbeeld:

```
uninstall.table.pm_owner.pm_run: uninstall.package.pm_owner.pm
```

De query:

```
select constraint_type, table_name, constraint_name, r_constraint_name
from all_constraints
where owner = 'PM_OWNER'
and constraint_name not like 'SYS_%'
order by constraint_type, table_name
```

laat onder andere de volgende resultaten zien:

CONSTRAINT_TYPE	C
TABLE_NAME	PM_SQLAREA
CONSTRAINT_NAME	PM_ARE_CK1
CONSTRAINT_TYPE	P
TABLE_NAME	PM_RUN
CONSTRAINT_NAME	PM_RUN_PK
CONSTRAINT_TYPE	P
TABLE_NAME	PM_SQLAREA
CONSTRAINT_NAME	PM_ARE_PK
CONSTRAINT_TYPE	R
TABLE_NAME	PM_SQLAREA
CONSTRAINT_NAME	PM_ARE_RUN_FK1
R_CONSTRAINT_NAME	PM_RUN_PK

Tabel 3. ALL_CONSTRAINTS

We kunnen hieruit de volgende regels destilleren:

```
install.constraint.pm_owner.pm_are_ck1: install.table.pm_owner.pm_sqlarea

install.constraint.pm_owner.pm_run_pk: install.table.pm_owner.pm_run

install.constraint.pm_owner.pm_are_pk: install.table.pm_owner.pm_sqlarea

install.constraint.pm_owner.pm_are_run_fk1:install.table.pm_owner.pm_sqlarea
```

en de foreign key afhankelijkheid:

```
install.constraint.pm_owner.pm_are_run_fk1:
install.constraint.pm_owner.pm_run_pk
```

Aanmaken van dependencies

In de Unix wereld is het gebruikelijk om voor het maken van een programma de build dependencies te bepalen. Dit zorgt ervoor dat een build alleen het hoogst noodzakelijke doet. De conventie is dat afhankelijkheden worden berekend via:

```
make depend
```

Voor een Oracle installatie voert 'make depend' de volgende taken uit:

- Uitzoeken welke SQL*Plus scripts welke DDL statements bevatten door middel van uitvoering van het Perl script makedepend_oracle.pl;
- Bepalen van de afhankelijkheden van objecten in een Oracle schema door middel van uitvoering van het SQL*Plus script makedepend.sql.

De resultaten worden naar MAKE include bestanden geschreven die in de overkoepelende makefile worden gebruikt.

De Performance Monitor verzamelt periodiek informatie over het functioneren van één of meer Oracle-databases

Oracle Forms en Reports modules

Om de afhankelijkheden voor Oracle Forms en Reports modules te bepalen, wordt ook het Perl script makedepend_oracle.pl gebruikt. Alleen tekstversies van forms (.fmt), menu's (.mmt), libraries (.pld) en reports (.rex) kunnen doorzocht worden op afhankelijkheden. Enige voorbeelden van afhankelijkheden:

a. De volgende regel uit een library tekstbestand mylib.pll geeft aan dat de library de attached library ofg4hpl.pll nodig heeft:

```
.ATTACH LIBRARY ofg4hpl.pll
```

De MAKE regel wordt:

```
mylib.pll: ofg4hpl.pll
```

b. Een attached library in een bestand myform.fmt wordt gekenmerkt door de volgende regels:

```
TN = 156     TV = <<"OFG4HPL">>
```

De MAKE regel wordt:

```
myform.fmb: ofg4hpl.pll
```

c. De volgende regel toont dat een bestand is gekoppeld aan een report myreport.rex:

```
LINKEDFILE = <<"pm_logo.bmp">>
```

De MAKE regel wordt:

```
myreport.rdf: pm_logo.bmp
```

Installatie van de Performance Monitor

De volgende MAKE bestanden worden gebruikt bij de installatie:

Bestand	Omschrijving
Makefile.mak	Overkoepelende makefile
ora_defs.mki	Oracle definities
os_defs.mki	Operating System definities
site_defs.mki	Site specifieke definities
prj_defs.mki	Project definities
vc_defs.mki	Versie Controle definities
Objects.mki	Database object definities. Bevat een lijst van SQL*Plus scripts.
Depend.objects.file.mki	MAKE include bestand. Resultaat van makedepend_oracle.pl

Depend.objects.db.mki	MAKE include bestand. Resultaat van makedepend.sql
Forms.mki	Forms definities. Bevat een lijst van Forms modules.
Depend.forms.mki	MAKE include bestand. Resultaat van makedepend_oracle.pl
Reports.mki	Reports definities. Bevat een lijst van Reports modules.
Depend.reports.mki	MAKE include bestand. Resultaat van makedepend_oracle.pl

Tabel 4. MAKE bestanden

Bij een installatie kunnen we de volgende stappen onderscheiden:
- Aanmaken van het schema
- Bouwen van Developer executables (.fmx, .mmx en .rep bestanden).
- Installatie van Developer executables.

Aanmaken van het schema

Het commando om de database installatie uit te voeren is:

```
make -f makefile.mak USERID=<connect string> install.objects
```

Het doel install.objects is onderverdeeld in de volgende doelen:
- install.sequences, install.tables, install.packages,
- install.package_bodies, install.functions, install.procedures,
- install.views, install.synonyms, install.roles, install.grants,
- install.constraints, install.indexes, install.triggers.

Bouwen van Developer executables

Het commando om Developer executables te bouwen is:

```
make -f makefile.mak USERID=<connect string> build.developer
```

Installatie van Developer executables

Het commando om Developer executables te installeren in de daartoe bestemde installatiedirectory is:

```
make -f makefile.mak USERID=<connect string> install.developer
```

Conclusie

Met behulp van deze methode kan snel een correcte installatieprocedure worden gerealiseerd. Als de installatie van de Performance Monitor applicatie als voorbeeld wordt gebruikt voor de installatie van een andere applicatie, dan is alleen een aanpassing aan een drietal MAKE bestanden nodig. De methode voldoet verder aan de eisen voor een correcte database installatie zoals reeds eerder beschreven:

- Alle objecten worden aangemaakt indien de broncodebestanden worden opgesomd in objects.mki, forms.mki en reports.mki en de dependencies 'up to date' zijn.
- Indien een object door meerdere scripts aangemaakt kan worden, dan wordt het nieuwste script het laatst uitgevoerd doordat makedepend_oracle.pl dit controleert.
- Er zullen zo min mogelijk Oracle foutmeldingen optreden, doordat de afhankelijkheden tussen objecten onderling van belang zijn voor de volgorde van uitvoeren van scripts.
- De installatie wordt na een Oracle fout gestopt en kan daarna weer verder vanaf dat punt aangezien er wordt bijgehouden, welke scripts foutloos uitgevoerd zijn.
- De installatie kan ongedaan gemaakt worden via 'make uninstall.objects'.
- De hulpmiddelen MAKE en Perl zijn gratis via het Web beschikbaar.

Referenties

MAKE
http://www.gnu.org/software/make/make.html

Perl
http://www.perl.org

"De veranderende rol van informatiesystemen",
Optimize, juni 2000,
Marc Berkenbosch en Gert-Jan Paulissen

ir. Gert-Jan Paulissen
is werkzaam bij Transfer Solutions
e-mail: gpaulissen@transfer-solutions.com