Gebruik van APEX à la Forms

Authenticatie, sessiebeheer, auditing en meer

Hoewel APEX al weer een decennium oud is, wordt het pas de laatste jaren meer en meer gebruikt. Bij mijn klant, een Nederlandse grootbank, wordt APEX sinds kort gebruikt voor Oracle-applicaties met een grafische user interface (GUI) nadat eerdere pogingen met Java en .Net niet succesvol zijn gebleken. Voorheen werden de applicaties met Oracle Forms gebouwd.

De applicatie is een workflowapplicatie waarbij handelaren wereldwijd autorisatie van diverse personen moeten krijgen om te kunnen handelen in bepaalde producten. Verder moeten als laatste stappen ook de back office en front office hiervoor gereed worden gemaakt. Voorheen gebeurde dit met een Excel-sheet dat via de email werd verstuurd.
Binnen de afdeling worden Oracle-applicaties standaard met bewezen technologie gebouwd zoals Oracle Designer, Headstart en CDM Rulerframe.
Verder worden de volgende goede gebruiken voor GUI-ontwikkeling in ere gehouden:

- geen logica in de GUI;
- de GUI mag alleen views en een aantal packages gebruiken als interface met de data.

Ik behandel de volgende verschillen tussen APEX en Oracle Forms: authenticatie, sessiebeheer, auditing, autorisatie en foutmeldingen.

Authenticatie

Oracle Forms gebruikt de volgende authenticatiemethode: je logt aan op de database met een account en wachtwoord. Deze connectie wordt vervolgens exclusief gebruikt tijdens de hele (gebruikers)sessie.
APEX doet dat anders: eerst wordt gecontroleerd of de gebruiker wel bevoegd is. Daarna verlopen alle databaseinteracties via databaseconnecties onder een generiek APEX-account. Dit is analoog aan J2EE-omgevingen. Er kunnen dus twee opeenvolgende database-interacties worden uitgevoerd door verschillende databaseconnecties.
Een gebruikerssessie heeft geen exclusieve databaseconnectie meer.
APEX biedt diverse mogelijkheden zoals authenticatie via LDAP, APEX accounts, via databaseaccounts of een eigen manier. Aangezien de organisatie geheel is ingericht op databaseaccounts, hebben we in deze applicatie voor databaseaccounts gekozen maar dan wel op een eigen manier.
Een van de problemen die naar voren kwam is dat APEX voor oudere Oracle-versies de volgende procedure uitvoert:

- haal het oude (versleutelde) wachtwoord van het account op uit DBA_USERS.
- verander het wachtwoord van het account via: ALTER USER <account> IDENTIFIED BY <wachtwoord>
- vergelijk het nieuwe (versleutelde) wachtwoord in DBA_USERS met het oude (versleutelde) wachtwoord. Komt dat overeen, dan was het wachtwoord blijkbaar goed.
- zet het oude wachtwoord terug.

Indien een gebruiker een wachtwoordprofiel heeft waarbij het wachtwoord niet mag worden hergebruikt, ontstaan er problemen, omdat het ALTER USER commando het wachtwoord hergebruikt. Daar komt nog bij dat de DBA niet graag heeft dat het softwareaccount van APEX het ALTER USER privilege heeft. In nieuwere versies van Oracle geldt dit probleem niet meer, doordat een interne Oracle-procedure wordt gevolgd waar geen ALTER USER aan te pas komt.
Om nu onafhankelijk te zijn van de Oracle-versie is een Java-programma gemaakt dat een JDBC connectie maakt met het account en wachtwoord en daarna de connectie weer afsluit. Als dat lukt, dan is het wachtwoord blijkbaar goed. Als het wachtwoord van het account is verlopen of dreigt te verlopen dan wordt er een Oracle-fout (ORA-28001) respectievelijk waarschuwing (ORA-28002) gegeven. Met de JDBC fat driver kan ook het wachtwoord worden gewijzigd, zodat de applicatiegebruikers in het aanlogscherm de mogelijkheid hebben om hun wachtwoord te wijzigen. Het gebruik van de fat driver sluit wel gebruik van Java in de database uit. De fat driver wordt daar niet ondersteund. Het Java programma kan dan

via een Remote Procedure Call (zie EPC in Referenties) worden aangeroepen op de server.

Sessiebeheer

Het gebruik van sessies in APEX
Het feit dat APEX databaseconnecties niet exclusief gebruikt voor gebruikerssessies heeft implicaties voor applicatiecaching: wees voorzichtig met package state. Als de gebruiker database-interactie initieert door bijvoorbeeld een record op te slaan, wordt een HTTP POST actie uitgevoerd. Intern schoont APEX dan de package state op. Na een HTTP POST actie volgt een HTTP GET actie met mogelijk een andere databaseconnectie. Dit is afhankelijk van het type branch in APEX: 'Redirect to Page or URL' is er zo een. Dus alleen tijdens de HTTP POST actie is package state betrouwbaar.

CDM Ruleframe
Als CDM Ruleframe een schending van een business rule of database constraint ontdekt, dan wordt detailinformatie over de fout in een package variabele – een PL/SQL tabel – opgeslagen (bijvoorbeeld 'leeftijd moet minimaal 18 zijn') en wordt een ORA-20998/ORA-20999 foutmelding gegenereerd. Maar als je in APEX de desbetreffende ORA-melding krijgt, dan kun je de detailinformatie niet (gegarandeerd) meer ophalen, omdat dat met een nieuwe database-interactie moet en dus (mogelijk) een andere connectie. Met Oracle Forms kan het wel, omdat je daar gegarandeerd dezelfde databaseconnectie hebt. Je weet in APEX dus standaard wel dat het fout gaat, maar niet waarom. Overigens is mijns inziens de fouthandeling van CDM ruleframe altijd al ondermaats geweest. Want ook als je in SQL*Plus een schending van een business rule krijgt, dan krijg je alleen die vermaledijde ORA-20998. Later in dit artikel beschrijf ik een oplossing voor dit probleem.

Een oplossing voor caching
Maar wat moet je doen als je wel caching wilt gebruiken over database-acties heen? Gebruik geen package state, maar gewoon een echte tabel met als sleutel het ID van de APEX-sessie. Met een tabel met als kolom een object type (te vergelijken met een Java-klasse) kunnen we alle mogelijke vormen van informatie opslaan door gebruik te maken van overerving. Bij de start van de database-actie haal je het object op. Als je de databaseactie afsluit moet het worden teruggeschreven naar de tabel (alleen indien het is gewijzigd).
De EPC applicatiecomponen bevat deze voorziening. Om performanceredenen bevat het root object een dirty flag dat door de applicatie gezet moet worden indien er iets wijzigt in het object. Dan kan de update achterwege blijven indien niets is veranderd.

```
create or replace type std_object as object (
/*
  -- The dirty flag is used to speed up the performance when
  -- std_object_mgr.get_std_object()/std_object_mgr.set_std_object() are
 used.
  --
  -- These functions use an internal (PL/SQL package) or external
(database table) cache
  -- to get or set an object.
  --
  -- There are two situations when std_object_mgr.get_std_object() is
called in a
  -- constructor of a type depending on std_object:
  -- 1) std_object_mgr.get_std_object() raises no_data_found (no object
found).
  -- Now the application should set the dirty flag for a new object to
1
  -- because a new object has to be written back to the cache in
  -- std_object_mgr.set_std_object().
  -- 2) std_object_mgr.get_std_object() succeeds.
  -- The dirty flag is set to null automatically by std_object_mgr.get_
std_object().
  -- Now the application should set it immediately to 0. In the rest of
the application
  -- the dirty flag should be set it to 1 if one of the members of the
object changes.
  -- Now std_object_mgr.set_std_object() will not write the object back
to the cache.
  */
  dirty integer

, not instantiable
  member function name(self in std_object)
  return varchar2

, final
  member procedure store(self in std_object)

, final
  member procedure remove(self in std_object)

) not instantiable not final
```

Hier is een constructor van een type dat illustreert hoe de dirty flag gebruikt dient te worden:

```
constructor function dbug_plsdbug_obj_t
return self as result
is
  l_object_name constant std_objects.object_name%type := 'DBUG_PLSDBUG';
  l_std_object std_object;
begin
  begin
    std_object_mgr.get_std_object(l_object_name, l_std_object);
    self := treat(l_std_object as dbug_plsdbug_obj_t);
    self.dirty := 0;
  exception
    when no_data_found
    then
      self.dirty := 1;
      self.ctx := null;
  end;

  -- essential
  return;
end;
```

Zoals het commentaar in de code laat zien, kun je zowel een PL/SQL tabel als een database tabel gebruiken. Het package std_object_mgr bevat een procedure set_group_name die ervoor zorgt dat objecten in de database tabel std_objects worden opgeslagen:

```
create table std_objects (
  group_name varchar2(100)
, object_name varchar2(100)
, created_by varchar2(30)
, creation_date date
, last_updated_by varchar2(30)
, last_update_date date
, obj std_object
, constraint std_objects_pk primary key (group_name, object_name)
, constraint std_objects_chk1 check (created_by is not null)
, constraint std_objects_chk2 check (creation_date is not null)
, constraint std_objects_chk3 check (last_updated_by is not null)
, constraint std_objects_chk4 check (last_update_date is not null)
)
cache
```

Om het geheel nu voor APEX te gebruiken, moeten we het volgende regelen:

- roep voor elke databaseinteractie de 'Virtual Private Database PL/SQL hook' aan
- roep in deze PL/SQL procedure eerst std_object_mgr.set_group_name(v('APP_SESSION')) aan

De logging/debugging component DBUG maakt gebruik van deze oplossing om informatie op te slaan over databasesessies heen.

Auditing

Voor auditing maken Designer en Headstart gebruik van de Oracle pseudokolom USER (TAPI en journaling). Dat kan in APEX niet meer. Want USER is altijd 'APEX_PUBLIC_USER', het generieke APEX-account. De functieaanroep v('APP_USER') geeft de gebruikersnaam aan waarmee in APEX is aangelogd. Deze functie retourneert NULL wanneer niet via APEX is aangelogd. We achterhalen dus de gebruikersnaam in programmatuur via NVL(V('APP_USER'), USER). Beter nog is dit door een functieaanroep te vervangen zodat maar een stuk code vervangen hoeft te worden als de applicatie op een nieuwe GUI overgaat.

Let dus op dat USER wordt vervangen in alle applicatiecode.

Autorisatie

Oracle Forms applicaties maken gebruik van databaserollen: enerzijds om te controleren welke programma's een gebruiker mag starten (moduleautorisatie), anderzijds om te bepalen hoe welke database-objecten mogen worden gebruikt (objectautorisatie). Dat laatste wordt overigens niet door Oracle Forms gecontroleerd maar door de database zelf. Beide autorisaties zijn niet meer van belang in APEX, omdat er met een generiek account wordt gewerkt.
Maar je kunt moduleautorisatie wel simuleren als je, zoals bij deze applicatie, gebruikt maakt van databaseaccounts voor authenticatie. Sla in elk APEX scherm een lijst van databaserollen op (in een regio die nooit getoond wordt omwille van de veiligheid) waarmee dit scherm mag worden gestart en controleer met een functie bij de start van het scherm of de APEX gebruiker (eigenlijk de gekoppelde database gebruiker) een van deze rollen heeft:

```
function module_authorized
( /* roles from view SESSION_ROLES separated by a semi-colon */
  p_session_roles in varchar2
, /* allowed roles separated by a semi-colon */
  p_roles_allowed in varchar2
)
return integer
is
  l_role_allowed varchar2(4000);
  l_roles_allowed2 varchar2(4000) := p_roles_allowed;
  l_result integer := null; /* 1: OK; 0: not OK */

  /* Split a string in two parts: before and after a delimiter */
  procedure split
  ( p_string in varchar2
  , p_delim in varchar2
  , p_before out varchar2
  , p_after out varchar2
  )
  is
    l_pos pls_integer;
  begin
    l_pos := instr(p_string, p_delim);
    if l_pos > 0
    then
      p_before := substr(p_string, 1, l_pos-1);
      p_after := substr(p_string, l_pos+length(p_delim));
    else
      -- delimiter not found
      raise no_data_found;
    end if;
  end split;
begin
  loop
    begin
      split
      ( p_string => l_roles_allowed2
      , p_delim => ';'
      , p_before => l_role_allowed
      , p_after => l_roles_allowed2
      );
      if instr(';'||p_session_roles||';', ';'||l_role_allowed||';') > 0
      then
        l_result := 1;
        exit;
      end if;

    exception
      when no_data_found
      then
        /* the remainder has no semi-colon, hence is the last role
allowed */
        l_role_allowed := l_roles_allowed2;
        l_result :=
          case

when instr(';'||p_session_roles||';', ';'||l_role_allowed||';') > 0
            then 1
            else 0
          end;
        exit;
    end;
  end loop;

  return l_result;
end module_authorized;
```

Dus als in APEX

```
module_authorized(:P0_SESSION_ROLES, :P100_ROLES_ALLOWED)
```

de waarde 1 retourneert, dan mag de gebruiker het scherm 100 starten.
Ook objectautorisatie kun je simuleren en wel met behulp van 'row level security'. Als de gebruiker zich authenticeert (oftewel een databaseconnectie maakt) dan kunnen we de view SESSION_ROLES gebruiken om te kijken welke rollen een gebruiker standaard heeft. Als deze lijst wordt bewaard (bijvoorbeeld in APEX pagina 0 in een regio die niet wordt getoond) in een puntkomma gescheiden lijst, dan kunnen we de volgende functie gebruiken om te zien of een gebruiker een bepaald privilege heeft voor een object:

```
function object_authorized
( p_session_roles in varchar2
, p_owner in varchar2
, p_object_name in varchar2
, p_privilege in varchar2
)
return integer
is
  l_found integer;
begin
  select  count(*) as found
  into    l_found
  from    user_tab_privs
  where   owner = p_owner
  and     table_name = p_object_name
  and     privilege = p_privilege
  and     /* a privilege may be granted to the user or to one of its
roles */
           instr
           ( ';' || nvl(v('APP_USER'), USER) || ';' || p_session_roles
|| ';'
           , grantee
           ) > 0;

  return sign(l_found); /* 1 = OK, 0 = not OK */
end object_authorized;
```

Dus als in APEX

```
object_authorized(:P0_SESSION_ROLES, 'APP_OWNER', 'APP_SYSTEMS',
'INSERT')
```

de waarde 1 retourneert, dan mag de APEX gebruiker records toevoegen aan de tabel/view APP_OWNER.APP_SYSTEMS.

Foutmeldingen

Zoals al genoemd in de paragraaf over sessiebeheer kun je niet vertrouwen op package state. Een manier om dit probleem te ondervangen is om in APEX een eventuele foutmelding na een databaseaanroep in een PL/SQL block op te vangen:

```
begin
  <database aanroep>;
exception
  when others
  then
    if sqlcode = -20998
    then
      :P100_ERROR_MSG := cg$errors.get_errors;
    else
      :P100_ERROR_MSG := sqlerrm;
    end if;
end;
```

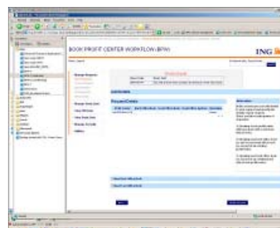
Deze aanpak wordt door APEX beschreven in de handleiding. Dat is behoorlijk omslachtig als je het mij vraagt.
Wat ik heb gedaan is radicaal anders. Ik gebruik geen extra PL/SQL code in APEX om fouten af te vangen, maar ik vang de foutmelding van CDM ruleframe in de database af en breid het foutbericht uit met de detailmeldingen. Via de procedure raise_application_error() wordt dan dezelfde foutcode gegenereerd, maar in plaats van de cryptische melding:

```
ORA-20998: Transaction failed.
```

krijg je nu:

```
ORA-20998: Transaction failed.
APP-00001: De leeftijd moet minimaal 18 zijn
```

Ook als je in SQL*Plus werkt is dit wel zo prettig. De oplossing is om op enkele plaatsen in de Headstart software de code aan te passen. Een script om code in de database aan te passen kun je vinden in Transferware. Verder wordt in APEX met behulp van Javascript en cookies de foutmelding (SQLERRM) opgevangen en netjes getoond aan de gebruiker:



Foutmelding vanuit CDM ruleframe.

Conclusies

Mits je je applicatie goed ontwerpt, hoeft ontwikkeling van een applicatie met een APEX GUI nauwelijks af te wijken van een applicatie met een Oracle Forms GUI voor wat betreft de databaseprogrammatuur. De vertrouwde omgeving van Designer, Headstart en CDM Ruleframe hoeft dus geen beletsel te zijn om te migreren, integendeel. Bij de organisatie waar ik werkzaam ben, zijn zowel ontwikkelaars als gebruikers uiterst tevreden met het gebruik van APEX à la Forms. De ontwikkelaars zijn blij met de productiviteit en stabiliteit van APEX en de gebruikersinterface voldoet aan de standaarden van vandaag. Ook qua performance is er niets te klagen.

Referenties

- *EPC, http://www.sourceforge.net/projects/transferware, module epc*
- *DBUG, http://www.sourceforge.net/projects/transferware, modules dbug en plsdbug*
- *Transferware, http://www.sourceforge.net/projects/transferware, module tw*

Gert-Jan Paulissen is Oracle-consultant bij Transfer Solutions.