HAI901I : Paquetages, SQL dynamique et méta-schéma

I.Mougenot

UM

2023

Paquetage PL/SQL

Vision modulaire : ensemble de constantes, fonctions et procédures qui outillent le même domaine métier

1. faciliter la réutilisation, la maintenance et la testabilité
2. vision finaliste (en droite ligne du métier)
3. bonnes pratiques (organisation, documentation, contrôle)

Exemples de paquetages déjà construits

- DBMS_OUTPUT : stocke l'information dans une mémoire tampon (buffer) afin de la restituer ultérieurement
- DBMS_METADATA : retourne l'ensemble des informations (définitions des objets du schéma) provenant du dictionnaire de données
- UTL_FILE : fournit les éléments nécessaires à des opérations de lecture/écriture dans des fichiers textes externes à la BD
- DBMS_ALERT : notifier des évènements aux usagers de la BD
- DBMS_SQL : fonctionnalités SQL étendues
- DBMS_XMLGEN : proposer les tables au format XML
- DBMS_STATS : collecter des statistiques sur les schémas (très utile pour l'optimisation des accès)
- DBMS_UTILITY : fonctions utilitaires : temps, conversion, analyse en routine ...

Commande DESCRIBE (ou DESC) pour consulter la structure publique du paquetage

```
desc dbms_output

PROCEDURE PUT_LINE
Nom d'argument  Type            E/S par defaut ?
--------------  --------------  ---------------
A               VARCHAR2        IN
```

Mettre à jour les statistiques pour l'ensemble des objets d'un schéma

```
EXEC DBMS_UTILITY.ANALYZE_SCHEMA(user,'COMPUTE')
```

Paquetage

Un peu comme pour un module : spécification et implémentation

1. déclaration du paquetage : variables et les méthodes (fonctions comme procédures) publiées et rendues publiques
2. définition du corps du paquetage : code des variables privées, méthodes privées et méthodes publiques

Exemple de tête de paquetage (Finances)

```
create or replace package Finances
 as
vTx_EF constant number := 6.55957;
vTx_ED constant number := 1.3926;

 function conversionF_EF (euros in number) return number;
 procedure conversionP_EF (euros in number, francs out
     number);
 function conversionF_ED (euros in number) return number;
end Finances ;
 /
```

Listing 1: Déclaration paquetage

Corps de paquetage (Finances) - partie 1

```
create or replace package body Finances
as
function conversion (montant in number, taux in number)
return number
is
begin
return (round(montant * taux, 2));
Exception when OTHERS then return null;
end;
function conversionF_ED (euros in number)
return number is
begin
return conversion (euros, vTx_ED);
end;

-- a suivre
```

Listing 2: Corps du paquetage

Corps de paquetage (Finances) - partie 2

```
function conversionF_EF (euros in number)
return number is
begin
return conversion (euros, vTx_EF);
end;
procedure conversionP_EF (euros in number, francs out
    number)
is
begin
francs := (round(euros * vTx_EF, 2));
Exception when OTHERS then dbms_output.put_line('
    erreur argument ');
end;
end Finances;
/
```

Listing 3: Corps du paquetage

Utilisation de Finances

```
select salaire, Finances.conversionF_ED(salaire) as
    enDollars,
Finances.conversionF_EF(salaire) as enFrancs from emp;

select * from emp where
    Finances.conversionF_EF(salaire) > 10000 ;
```

Listing 4: Quelques usages possibles

Partager Finances avec d'autres

```
grant execute on Finances to public;

--- pour les autres usagers : prefixer par le schema de
    provenance
select * from emp where
    p00000009432.Finances.conversionF_EF(salaire) >
    10000 ;

-- a la place utiliser un synonyme
create public synonym calculetteFinanciere for finances;
grant execute on calculetteFinanciere to public;
select * from emp where
    calculetteFinanciere.conversionF_EF(salaire) > 10000
    ;
```

Listing 5: Donner des droits

Retour sur les exceptions

Les erreurs levées et traitées via le mécanisme d'exception possèdent un code erreur (SQLCODE) et un message d'erreur (SQLERRM) dédiés.

```
alter table emp modify salaire not null ;

begin
update emp set salaire = null where n_dept = 10 ;
dbms_output.put_line('exception ?');
exception
when others
then
dbms_output.put_line('code erreur '||SQLCODE);
dbms_output.put_line('message erreur '||SQLERRM);
end;
/
```

Listing 6: Illustration exceptions

Après exécution

```
code erreur -1407
message erreur ORA-01407: impossible de mettre a jour
    ("P00000009432"."EMP"."SALAIRE") avec NULL
```

Listing 7: Illustration exceptions

Test exception utilisateur

```
declare
monExc exception;
montant number(7,2);
begin
montant := &montant;
if montant < 800
then raise monExc ;
end if;
update emp set salaire = montant where n_dept = 10 ;
dbms_output.put_line('exception ?');
exception
when monExc then
dbms_output.put_line('code erreur '||SQLCODE||' et
    message erreur '||SQLERRM);
end;
/
```

Listing 8: Illustration exception utilisateur

Après exécution

```
code erreur 1
message erreur User-Defined Exception
```

Listing 9: Illustration exceptions

Usage de pragma (directive compilateur) et exception_init

```
declare
monExc exception;
PRAGMA EXCEPTION_INIT(monExc, 100 );
montant number(7,2);
begin
montant := &montant;
if montant < 800
then raise monExc ;
end if;
update emp set salaire = montant where n_dept = 10 ;
dbms_output.put_line('exception ?');
exception
when monExc then
dbms_output.put_line('code erreur '||SQLCODE||' et
    message erreur '||SQLERRM);
end;
```

Après exécution

```
-- code 100 = code de NO_DATA_FOUND
code erreur 100
message erreur ORA-01403: aucune donnee trouvee
```

Listing 11: Illustration exceptions

Usage de pragma et exception_init

```
declare
monExc exception;
PRAGMA EXCEPTION_INIT(monExc, -20100 );
montant number(7,2);
begin
montant := &montant;
if montant < 800
then raise_application_error(-20100,'trop faible') ;
end if;
update emp set salaire = montant where n_dept = 10 ;
dbms_output.put_line('exception ?');
exception
when monExc then
dbms_output.put_line('code erreur '||SQLCODE||' et
    message erreur '||SQLERRM);
end;
/
```

Après exécution

```
code erreur -20100
message erreur ORA-20100: trop faible
```

Listing 13: Illustration exceptions

SQL dit "dynamique"

Passage obligé en PL/SQL dans deux cas de figures précis :

1. répétitivité traitement et prise en charge de valeurs différentes à l'exécution
   - performance des accès (optimisation des requêtes)
   - sécurité des accès (éviter "SQL injection")
2. recours aux instructions DDL : `CREATE, DROP ou ALTER TABLE`
   - cohérence schéma et BD

Variable attachée ou liée : exemple

```
variable fonction varchar2(20) ;
execute :fonction := 'commercial'
select nom from emp where fonction = :fonction;
```

Listing 14: Attachement de variable

Mise à jour du schéma (recours à EXECUTE IMMEDIATE)

```
create table test (a varchar(5));

create or replace procedure supp (tble_name in varchar)
    is
 begin
 execute immediate 'drop table '||tble_name||' cascade
 constraints';
 dbms_output.put_line('table '||tble_name||' detruite ');
 exception
 when others then dbms_output.put_line('Pb suppression
     ');
 end;
 /
execute supp('test');
```

Listing 15: Suppression de table

Lutter contre l'injection SQL

```
CREATE OR REPLACE FUNCTION get_num_of_employees (p_job
    VARCHAR2)
RETURN NUMBER
IS
  v_query_str VARCHAR2(1000);
  v_num_of_employees NUMBER;
BEGIN
  v_query_str := 'SELECT COUNT(*) FROM emp'
              || ' WHERE fonction = :bind_job';
  EXECUTE IMMEDIATE v_query_str
    INTO v_num_of_employees
    USING p_job;
  RETURN v_num_of_employees;
END;
/
```

Exemple d'utilisation de la fonction

```
select get_num_of_employees('directeur') from dual;
```

Le niveau méta

Abstraire pour mieux organiser et contrôler

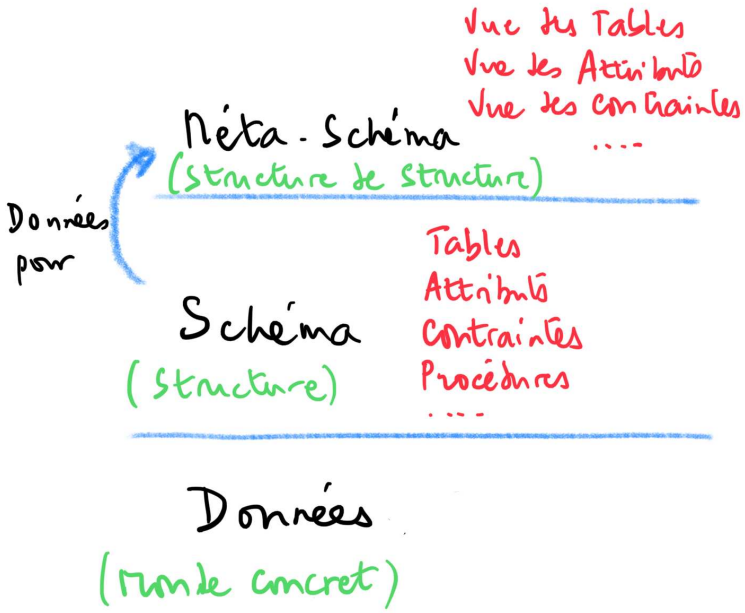

Figure: Trois niveaux

Dictionnaire de données ou méta-schéma

Ensemble de vues (avec pour propriétaire l'utilisateur SYS) permettent de disposer d'une vue d'ensemble sur :

1. les schémas utilisateurs (vues statiques). La méta-vue est DICTIONNARY ou DICT
2. les activités courantes appliquées à la BD (vues dynamiques). La méta-vue est V$FIXED_TABLE

Décrire les structures de la BD

Les éléments du schéma envisagés comme des métadonnées et consultables comme des données à part entière. Trois grands niveaux organisationnels :

1. USER_ : Informations sur tous les objets du schéma utilisateur
2. ALL_ : Informations sur tous les objets qui sont accessibles à l'utilisateur
3. DBA_ : Informations sur tous les objets de la base

Des exemples

```
select table_name from dict;

select table_name from user_tables;
select table_name from all_tables;
select table_name from dba_tables;

select constraint_name, table_name, column_name from
    user_cons_columns;
```

Listing 16: Consultation vues statiques

Des exemples

```
desc dba_tables
col table_name for a30
col owner for a20
select table_name, owner from dba_tables;
desc dba_objects
select table_name, t.owner, created from dba_tables t
    join dba_objects o
on table_name = object_name and o.owner = t.owner order
    by created;
```

Listing 17: Jointure vues statiques

Décrire les processus et évènements qui s'appliquent à la BD (surtout côte instance)

Ces aspects dynamiques ciblent surtout la performance, le "tuning" et la sécurité

1. V$ (ou bien V_$) : vues publiques
2. GV$: idem avec une colonne identifiant en plus (G pour Global)
3. X$: tables difficiles à décrypter qui servent à alimenter les vues V$

Des exemples

```
select username, terminal, program from v$process;
desc v$lock
select sid, username, logon_time from v$session where
    type='USER';
```

Listing 18: Exemples vues dynamiques

Exemples sur les privilèges système

Deux sortes de privilèges co-existent : sur les droits des utilisateurs, sur les objets des schémas. Nous nous intéressons ici seulement aux droits des usagers.

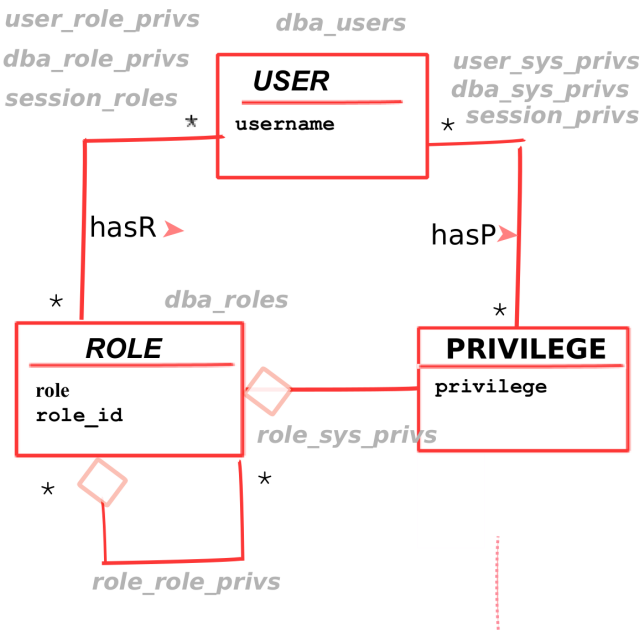

Figure: Diagramme de classes privilèges systèmes

Des exemples

```
select role, privilege from role_sys_privs where
    role='RESOURCE';
 select * from dba_role_privs where grantee ='DBA';
 select * from dba_role_privs where grantee like 'P000%';
```

Listing 19: Exemples consultation

Architecture

Partage de la même base de données Oracle

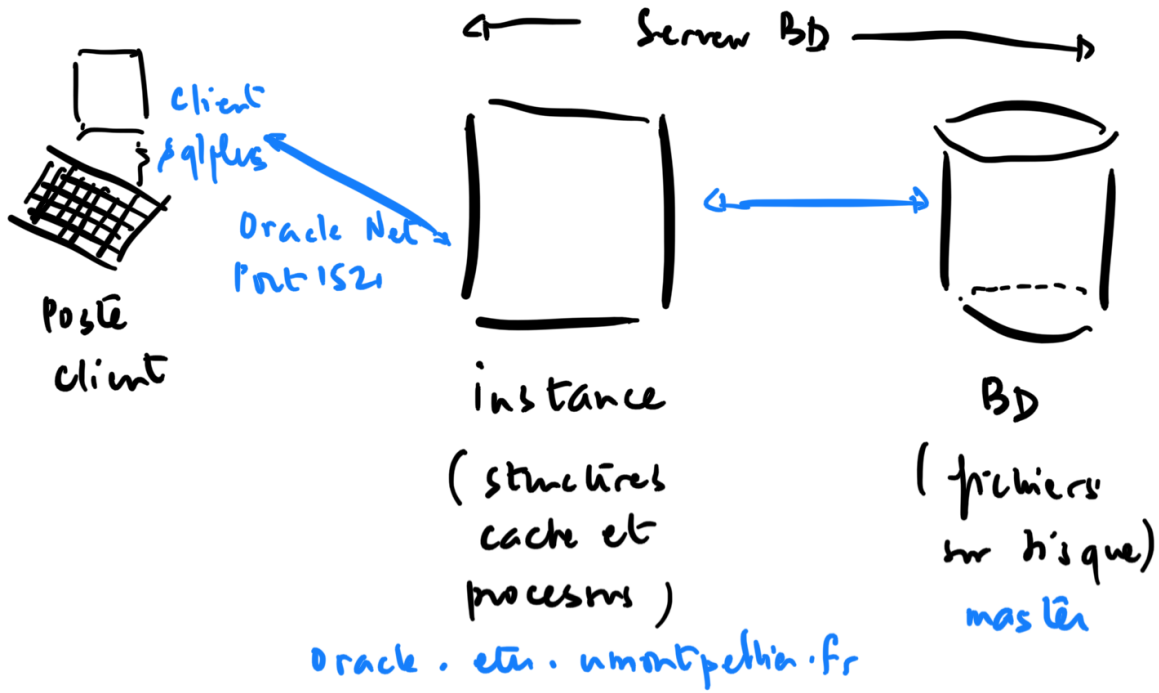

Figure: Client/Serveur Oracle