Oracle et Objet-relationnel

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2010-2011

Oracle et la norme SQL 3

La norme n'est pas respectée (problème d'antériorité) mais

- possibilité de définir des ADT (types Objets)
- collections pour typer les colonnes, en particulier tables imbriquées (cf cours sur Bulk SQL)
- types record = row type (cf cours de licence sur PL/SQL)
- Langage de requête adapté à la navigation entre objets
- héritage simple sur les ADT (pas sur les tables)

Création d'un schéma

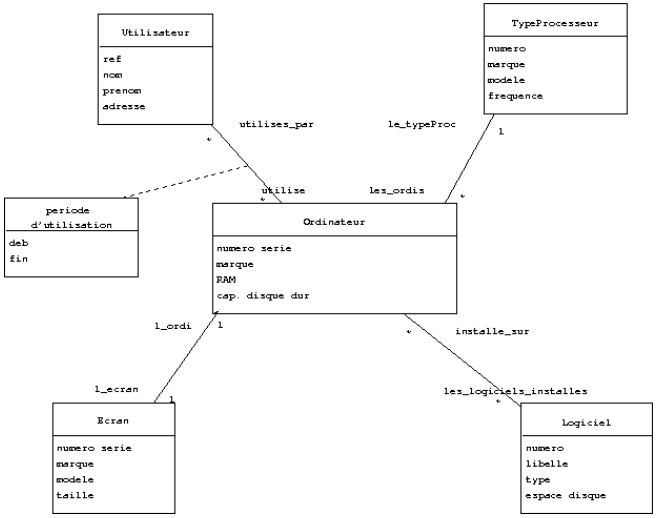
A partir d'un diagramme de classe UML,

- On définit des ADT (types objets), qui représentent les entités de notre diagramme
- Eventuellement, on a besoin d'autres types pour définir des records, des collections, ...
- Les objets correspondant aux entités sont stockées dans des tables (1 ligne = 1 objet).
- Ce qui était en relationnel traduit par des clés primaires / clés étrangères, va être maintenant traduit par des pointeurs sur des objets.
- Il est possible de définir des contraintes d'intégrité, à la manière du modèle relationnel

Types objets

- Un type objet contient une spécification et un corps
- La spécification contient les attributs de l'objet, et les spécifications des méthodes
- Les attributs sont
  - d'un type SQL,
  - d'un type composé (record, collection),
  - d'un type objet
  - d'un type pointeur vers un objet.
- Le corps du type contient le corps des méthodes.

Exemple



Association 1-1

```
create type ordi_type ;

create type ecran_type as object(
  numserie NUMBER(2),
  marque VARCHAR2(20),
  modele VARCHAR2(20),
  taille NUMBER(2),
  l_ordi REF ordi_type
)
```

```
create type ordi_type as object(
  numserie NUMBER(2),
  marque VARCHAR2(20),
  RAM NUMBER(4),
  capDisqueDur NUMBER(3),
  l_ecran REF ecran_type,
  ...)
```

Un objet `ecran_type` a un pointeur sur un objet `ordi_type` et réciproquement (on n'est pas obligé de traduire les deux sens de l'association !).

Association 1-N

```
create type refOrdi_type as OBJECT (refOrdi REF ordi_type)
create type ensOrdi_type as TABLE of refOrdi_type
```

```
create type typeProc_type
as object(
  marque VARCHAR2(20),
  modele VARCHAR2(20),
  frequence NUMBER(5),
  les_ordis ensOrdi_type
)
```

```
create type ordi_type as object(
  ...
  le_typeProc REF typeProc_type,
  ...)
```

Le type `ensOrdi_type` est une *nested table* qui permettra de stocker une collection de pointeurs sur les objets `ordi_type`

Association N-N

```
create type logiciel_type
create type reflogi_type as OBJECT (refLogi REF logiciel_type)
create type enslogi_type as TABLE of reflogi_type
```

```
create type logiciel_type
as OBJECT(
  numero NUMBER(2),
  libelle VARCHAR2(50),
  espaceDisque NUMBER,
  installeSur ensOrdi_type
)
```

```
create type ordi_type as object(
  ...
  les_logiciels enslogi_type,
  ...)
```

Classe-association

Création d'une classe pour l'association "utilise".

```
create type utilisateur_type ;

create type periodeUtil_type as OBJECT (
  debut date,
  fin date,
  l_utilisateur REF utilisateur_type,
  l_ordi REF ordi_type
)

create type refperiodeUtil_type
as OBJECT (periode REF periodeUtil_type)

create type ensPeriodeUtils_type as TABLE of refperiodeUtil_type
```



```
create type adresse_type
as object(
batiment VARCHAR2(20),
bureau VARCHAR2(10)
)

create type utilisateur_type
as OBJECT (
  numero NUMBER(2),
  nom VARCHAR2(20),
  prenom VARCHAR2(20),
  adresse adresse_type,
  utilise ensPeriodeUtils_type
)
```

```
create type ordi_type as object(
  ...
  utilisePar ensPeriodeUtils_type)
```

L'adresse pouvait aussi être d'un type `RECORD`.

Création des tables et des contraintes

- On va créer une table pour chaque entité de notre diagramme

  ```
  create table Entite of Entite_type ; -- au minimum !
  ```

- On peut ajouter des contraintes :
  - contraintes de domaine (en particulier, `NOT NULL`)
  - valeurs par défaut, bien utiles pour les collections.
  - contraintes de clé primaire (un identifiant est toujours utile pour les requêtes, mais moins indispensable que pour le modèle relationnel à cause des identifiants d'objet)
  - contraintes de clés étrangères (sur les types `REF`)
- On doit préciser comment stocker les tables imbriquées (`nested table`)

Exemple (suite)

```
create table ecran of ecran_type(
  constraint ecran_pkey primary key(numserie),
  constraint marque_ecran_non_null marque not null,
  constraint modele_ecran_non_null modele not null
);

create table typeProc of typeProc_type (
  constraint marque_proc_non_null marque not null,
  constraint modele_proc_non_null modele not null,
  constraint les_ordis_defaut les_ordis default ensOrdi_type()
) NESTED TABLE les_ordis STORE AS tab_lesOrdi ;

create table logiciel of logiciel_type (
  constraint libelle_log_non_null libelle not null
) NESTED TABLE installeSur STORE AS tab_installeSur ;
```

```
create table utilise of periodeUtil_type ;

create table utilisateur of utilisateur_type(
  constraint utilisateur_pkey primary key(numero)
) NESTED TABLE utilise STORE AS tab_utilise ;

create table ordi of ordi_type(
  constraint ordi_pkey primary key(numserie),
  constraint marque_ordi_non_null marque not null
) NESTED TABLE les_logiciels STORE AS tab_logiciels,
  NESTED TABLE utilisePar STORE AS tab_utilisePar ;
```


A propos des contraintes

- On a intérêt à mettre une valeur par défaut aux tables imbriquées
  par exemple, dans la table typeProc :

  `constraint les_ordis_defaut les_ordis default ensOrdi_type()`

- *On peut mettre des contraintes sur les attributs de type REF*
  - *clef étrangère*
    Par exemple dans la table ordi :

    `constraint ordi_proc_fkey le_typeProc references typeProc`
  - `SCOPE` *: Cette clause permet de préciser que les pointeurs d'objets ne pointent que sur une table donnée. Ca permet de stocker des pointeurs plus petits (relatifs dans la table) mais pas de vérifier des clés étrangères.* Par exemple dans la table ordi :

    `l_ecran SCOPE IS ecran`

Requêtes de mise à jour

Nécessité de faire appel aux constructeurs de types.

```
insert into utilisateur(numero,nom,prenom,adresse)
values (1,'Caron','Anne-Cécile',adresse_type('M3-ext','219'));

update typeproc set les_ordis = ensordi_type()
where modele = 'AMD Athlon II' and frequence = 28000 ;
```


Gestion des pointeurs

Création d'un écran et rattachement à un ordinateur :

```
insert into ecran values
  (1,'dell','E1910',17,(select ref(o) from ordi o where numserie=1));

update ordi o
set l_ecran=(select ref(e)
             from ecran e where e.l_ordi = ref(o))
where o.numserie=1 ;
```

Gestion des pointeurs et tables imbriquées

- La commande `insert into the (select ...)` stocke une ligne dans la table imbriquée désignée par `the(...)`.
- Le `select` après le `the` ne doit retourner qu'une seule table.

```
-- le jdk 6.0 (numero=1) est installe sur la machine 1
insert into the(select o.les_logiciels from ordi o where numserie = 1)
select ref(l) from logiciel l
where numero = 1 ;
-- on met a jour la table logiciel
insert into the (select l.installesur from logiciel l
                 where numero = 1)
select ref(o) from ordi o
where o.numserie = 1 ;
```


Navigation – Notation pointée

Elle permet d'exprimer un chemin dans le graphe des données.

```
select e.numserie,e.marque,e.modele,e.taille,e.l_ordi.numserie,e.l_ordi.marque
from ecran e;
NUMSERIE  MARQUE       MODELE  TAILLE  L_ORDI.NUMSERIE  L_ORDI.MARQUE
1         dell         E1910       17       3             dell
2         dell         ST2410      24       2             compaq
3         Samsung      SyncMaster  19       1             dell


select t.les_ordis from typeproc t;
LES_ORDIS(REFORDI)
-------------------------------------------
ENSORDI_TYPE(REFORDI_TYPE(...adresse...))
ENSORDI_TYPE(REFORDI_TYPE(...adresse...), REFORDI_TYPE(...adresse...))
```


Navigation – Tables imbriquées

On peut accéder aux colonnes des lignes de la table imbriquée en la nommant par un alias

```
-- logiciels installes sur la machine 1
select nlogi.reflogi.libelle
from the(select o.les_logiciels
         from ordi o
         where o.numserie = 1) nlogi ;

REFLOGI.LIBELLE
---------------------------
postgresql 8.3.5
jdk 6
windows vista
```


Les expressions TABLE

- Une expression `TABLE(...)` peut remplacer la requête `THE(...)`

```
select nlogi.reflogi.libelle
from table (select o.les_logiciels from ordi o
            where o.numserie = 3) nlogi ;
```

- Une expression `TABLE(...)` peut être utilisée pour parcourir une table imbriquée.

```
select o.numserie, l.reflogi.libelle
from ordi o,
     table(o.les_logiciels) l

 NUMSERIE  REFLOGI.LIBELLE
2  windows vista
3  windows vista
1  postgresql 8.3.5
1  jdk 6
2  jdk 6
1  windows vista
```

Définition de méthodes

- La définition d'un ADT inclut des méthodes de manipulation des objets de ce type (fonctions ou procédures).
- Surcharge autorisée.
- La notion de méthode publique ou privée n'existe pas.
- Il est possible d'autoriser ou d'interdire l'utilisation d'un ADT par un autre utilisateur (`grant execute ... to ...`) mais pas l'accès à certaines méthodes (tout ou rien).
- Il existe des méthodes particulières de comparaison d'objets.

Corps de la méthode

- On distingue la spécification du corps de la méthode.
- Pour appeler une méthode, on utilise la notation pointée.

```
alter type ordi_type replace as object(
  ...les attributs ...,
  member procedure set_ecran(num_ecran in number),
  ...autres méthodes...
)

create or replace type body ordi_type as
  member procedure set_ecran(num_ecran in number) is
  ...
  end ;
  ... autres méthodes ...
end ;
```



```
member procedure set_ecran(num_ecran in number) is
  deja_affecte number;
begin
  if self.l_ecran is not null then erreur(...) ; end if ;
  -- verifier que l''ecran n''est pas deja affecte
  select count(*) into deja_affecte from ordi o
  where o.l_ecran.numserie = num_ecran ;
  if (deja_affecte = 0) then
    -- on met a jour la table ordi
    update ordi o set o.l_ecran = (select ref(e) from ecran e
                                   where num_ecran = e.numserie)
    where o.numserie = self.numserie ;
    -- on met a jour la table ecran
    update ecran e
    set e.l_ordi = (select ref(o) from ordi o
                    where o.numserie = self.numserie)
    where e.numserie = num_ecran ;
  else erreur(...) ;
  end if ;
end set_ecran;
```


Fonctions de comparaison.

Pour comparer des objets : il faut définir une fonction `MAP` ou `ORDER`.

```
alter type adresse_type replace as object(
... attributs ...,
ORDER MEMBER FUNCTION
  compareAvecAdresse(adr in adresse_type) return integer
)
/
```

Cette méthode compare `self` et `adr` selon l'ordre lexicographique sur (batiment,bureau).

```
select u.nom,u.prenom,u.adresse.batiment,u.adresse.bureau
from utilisateur u
order by adresse;
```

Requêtes et appels de méthode

```
alter type logiciel_type replace as OBJECT(
  numero NUMBER(2),
  libelle VARCHAR2(50),
  espaceDisque NUMBER,
  installeSur ensOrdi_type,
  member function nbr_machines return number
)


create or replace type body logiciel_type as
  member function nbr_machines return number is
  begin
    if self.installeSur is null then return 0 ;
    else return self.installeSur.count ;
    end if ;
  end nbr_machines ;
end ;
```



```
select l.numero, l.libelle, l.nbr_machines()
from logiciel l ;
```

On peut aussi écrire :

```
select l.numero, l.libelle,
       (select count(*) from table(l.installeSur))
       as nbr_machines
from logiciel l ;
```