SGBD objet relationnel : Langage SQL3

Syntaxe SQL3

La syntaxe simplifiée du langage SQL3 est exprimée au moyen d'une grammaire BNF. La syntaxe de la grammaire est :

un caractère du langage est représenté en **gras**,

<mot> représente un élément non terminal

Les trois opérateurs d'occurrence sont :

(x)* représente 0 ou plusieurs fois l'élément x,

(x)+ représente 1 ou plusieurs fois l'élément x,

[x] :représente une occurrence optionnelle de x.

x | y représente l'élément x OU y.

1. Définition du schéma

1.1 Définition d'un type de données

<schema> =
(<définition_type_objet> | <définition_type_ensembliste>
| <déclaration_type>)*

<définition_type_objet> =
create type <type_objet> **as Object (**
(<nom_attr> [**ref**] <type>,)+
(<declaration_methodes>,)*
);
/

<définition_type_ensembliste> =
create type <type_ensembliste> **as** (**Table** | **Varray(**<longueur>**)**) **of** <type> ;
/

Déclaration d'un type incomplet qui sera complété ultérieurement.

<déclaration_type > =
create type <type_objet> **;** --Utile pour définir un type qui mentionne un autre type qui n'est pas encore défini.
/

<type> = <type_atomique> | <type_objet> | <type_ensembliste>
<type_atomique> = **Varchar2(**<longueur>**)** | **Number(**<longueur> [,<longueur>]**)** | **Date** | …

Compiler chaque définition de type **individuellement** avec la commande « / » ou @compile

1.2 Définition d'une association entre types

Une association est représentée en ajoutant des attributs aux types reliés par l'association. Lorsque l'arité (*i.e.* la cardinalité) de l'association est supérieure à 1, le type de l'attribut ajouté est ensembliste (table ou varray).
Le graphe des associations entre type doit être acyclique. Utiliser le type REF pour éviter les cycles.

a) Association 1-1 entre deux types X et Y :

Direction X → Y :

Si l'association est une **agrégation** :
create type **X** as object (a **Y**, …);
sinon,
create type **X** as object (a ref **Y**, …);
+ contrainte d'unicité pour indiquer qu'il n'existe pas 2 objets X qui font référence au même objet Y.

Direction Y → X
Solution symétrique.

b) Association 1-N entre deux types Y et X :

Si l'association est une **agrégation** :

```
create type Y as object (...);                                  -- définir le type Y
create type Ens_Y as varray(n) of Y;   -- définir le type "ensemble de Y" en utilisant varray ou table of …
create type X as object ( a Ens_Y, …);                          -- definir le type X
```

sinon

Direction X → Y (X est associé à plusieurs Y)

```
create type Y; /
create type Ens_Y as varray(n) of ref Y;                        -- ou table of …
create type X as object ( a Ens_Y, …);
```

Direction Y → X (Y est associé à un X)

```
create type Y as object ( b ref X, …);
```

c) Association N-M entre deux types X et Y :

Direction X → Y

```
create type Y;
create type Ens_Y as varray(n)  of ref Y;                       --
create type X as object ( a Ens_Y, …);
```

Direction Y → X

```
create type Ens_X as varray(n)  of ref X;                       --
create type Y as object ( b Ens_X, …);
```

1.3 Définition d'une méthode

<déclaration_méthode> = <déclaration_fonction> | <déclaration_procédure>

<déclaration_fonction> =
member function <nom_fonction> [**(**<nom_paramètre> in <type>, ...**)**]
return <type_du_resultat>

<déclaration_procédure> =
member procedure <nom_procedure> [(<nom_paramètre> in <type>, ...)]

Le corps des méthodes est défini ensuite au moyen de la commande :

```
create or replace type body <type_objet> as
<déclaration_methode> is
  <déclaration_variables_locales>
  begin
    corps de la méthode
  end <nom_methode>;
…
end;
```

1.4 Invocation d'une méthode

Un envoi de message ou un appel de méthode est symbolisé par un **point** :

`receveur.méthode(paramètres).` La méthode est définie dans la classe du receveur.

1.5 Exemples

Exemple de définition de type	
create type Adresse **as object (** num **Number(**3**),** rue **Varchar2(**40**),** ville **Varchar2(**30**),** code_postal **Number(**5**)** **);** @compile	**create type** Personne **as object (** nom **Varchar2(**30**),** prenom **Varchar2(**30**),** habite **Adresse,** date_naissance **Date,** **member function** age **return Number** **);** @compile

Exemple de corps de méthode

create type body Personne **as**
member function age **return Number is**
begin
return sysdate – date_naissance;
end age;
end;
@compile

2. Définition du stockage

Les données sont stockées dans des tables. Définir des relations (avec les éventuelles relations imbriquées).
<Stockage> =
(<définition_table>) *

<définition_table> =
create table <nom_table> **of** <nom_type>
(**nested table** <nom_attribut> **store as** <nom_table_imbriquée> **,**)* **;**

Inutile de préciser de *nested table* pour le type ensembliste *Varray*. En revanche, définir une table imbriquée pour chaque attribut ensembliste de type «*table of*». Toute instance possédant un identifiant d'objet doit être stockée dans une relation. Il ne peut y avoir de référence à une instance qui n'est pas stockée dans une relation. Ne pas compiler les instructions de création de table. (*i.e.*, pas de / après un *create table*).

3. Création des données

Utiliser directement des instructions SQL *insert* ou *update* pour instancier la base.
insert into <relation> **values(**<valeur>, …**)**;

<valeur> = <nombre> | **'**chaîne de caractère**'** | <objet> | variable

<objet> = <type>**(**<valeur>**,** …**)** -- le constructeur <type> est le nom du type de l'objet

Utiliser une procédure PL/SQL pour traiter l'insertion : définir une procédure d'insertion, puis l'exécuter
<définition_procédure> =
create procedure <nom_procédure> **as**
<declaration_variables_locales>
begin
…
end;

@compile

Compiler chaque définition de procédure individuellement avec la commande « / » ou @compile
<exécution_procédure> =
begin
<nom_procédure>
end;
/

4. Requêtes SQL3

Forme générale d'une requête

select [distinct] <projection>
from <données>
[**where** <conditions>]

La clause select exprime le résultat recherché (*i.e.*, la projection). La syntaxe simplifiée est :
<projection> = (<attribut_proj>,) +
<attribut_proj> = <chemin> | **value(**<chemin>**)** | **ref(**<chemin>**)** | **deref(**<chemin>**)**
<chemin> = <elt_chemin> | <elt_chemin>**.**<chemin>
<elt_chemin> = <variable> | <attribut> | <nom_fonction>

La clause from exprime l'ensemble des données accédées. La syntaxe simplifiée est :
<données> =
(<collections> <variable> **|** **Table**(<chemin>) <variable>)*

Requête avec group by

Comme en SQL on peut partitionner un ensemble suivant certains attributs de regroupement :

group by <attributs de regroupement>

Requête avec group by ... having

On peut également filtrer les groupes obtenus avec la clause **having.** La syntaxe est

group by <attributs de regroupement>
having <prédicat>

Un groupe est sélectionné ssi il satisfait le prédicat de la clause **having**. (Rmq : Ne pas confondre where avec having)

Les fonctions d'agrégat : count, sum; min, max, avg

Comme en SQL, on trouve différentes fonctions d'agrégat qui retournent une valeur numérique.

Les quantificateurs

Comme en SQL, on dispose des quantificateurs **exists** et **all**.

x.age > **all** 60

Cette expression rend vrai ssi le prédicat x.age>60 est vrai **pour tout x**.

exists (sous-requête)

Cette expression rend vrai ssi le résultat de la sous requête n'est pas vide.

Requêtes imbriquées

Comme en SQL, les clauses **select** , **from** et **where** peuvent contenir des sous requêtes imbriquées.