TD3 et TD4 – SQL 3

Exercice 1 : Application CAO

L'objectif est d'étudier et de modéliser une application en vue d'en établir une base de données objet-relationnelle. On utilisera le système objet-relationnel Oracle8i comme support d'implantation.

Dans une entreprise, on souhaite développer une **application CAO** pour connaître l'ensemble des éléments entrant dans la composition d'une pièce. On considère deux types de pièces :

- les pièces **de base**,
- les pièces **composites** obtenues par assemblage d'autres pièces (composites ou non).

Voir l'exemple de données (table, billard) ci-dessous.

Les données

On veut constituer une base de données comportant tous ces types de pièces :

- pour les pièces de base, on stockera son nom et sa matière. On suppose en effet qu'il existe différents types de pièces de base (cubique, sphérique, cylindrique, parallélépipède...) réalisées dans différentes matières (bois, acier, plastique...) et dont on connaît les dimensions. Pour chaque matière on connaît son nom, son prix au kilo et sa masse volumique.
- pour les pièces composites, on stockera son nom, le coût de son assemblage, l'ensemble des différentes pièces entrant dans sa fabrication en précisant la quantité et l'ensemble des pièces dans la fabrication desquelles elles entrent en précisant la quantité.

Exemple de données. La base contient les objets suivants

- 3 matières décrites par leurs nom, prix_kilo et masse_volumique :

  ('bois', 10, 2), ('fer', 5, 3), ('ferrite', 6, 10).

- 6 pièces de base décrites par leur forme : nom, dimension et matière :

  Cylindre : 'canne', (2, 30), bois,
  Parall : 'plateau', (1, 100, 80), bois,
  Sphère : 'boule', 30, fer,
  Sphère : 'pied', (30), bois,
  Cylindre : 'clou', (1, 20), fer,
  Cylindre : 'aimant', (2,5), ferrite.

- 2 pièces composites :

  Une table composée d'un plateau, 4 pieds et 12 clous, le coût d'assemblage est de 100 EUR.

  Un billard composé d'une table ; 3 boules et 2 cannes, le coût d'assemblage est de 10 EUR.

Les traitements

On veut pouvoir calculer le prix de revient et les différentes caractéristiques physiques (masse, volume) d'une pièce dont on connaît le nom. On veut aussi pouvoir connaître le nombre exact de pièces de base entrant dans la fabrication d'une pièce composite. Enfin, pour une pièce donnée, on veut savoir dans la fabrication de quelle(s) pièce(s) elle entre.

Question 1 : Proposer un diagramme de cette application en utilisant le modèle objet.

Question 2 : Schéma SQL3

Définir les types nécessaires en respectant la syntaxe SQL3. Intégrer votre schéma (fichier schema.sql), dans le SGBD Oracle.

Question 3 : Stockage et instanciation

1. Quelles sont les relations (ou collections d'objets) nécessaires pour poser les requêtes R1à R7 ?
2. Définir les relations permettant de stocker toutes les pièces de l'application dans la base de données (fichier instance.sql).
3. Ecrire les ordres de manipulation de données pour créer 3 matières : bois, fer, ferrite (fichier instance.sql),
4. Créer la procédure insere_pieces_de_base pour instancier les pièces de base 3 pièces en bois, 2 en fer et 1 en ferrite.
5. Créer la procédure insere_pieces_composites pour instancier les pièces composites.

Question 4 : Requêtes

Écrire les requêtes suivantes en SQL3. Pour chaque requête, donner le **type du résultat**.

R1. Quel est le nom et le prix au kilo des matières ?

R2. Quel est le nom des matières dont le prix au kilo est inférieur à 5 euros ?

R3. Quelles sont les pièces de base en bois ?

R4. Quel est le nom des matières dont le libellé contient 'fer'. Utiliser la syntaxe : *like '%fer%'*

R5. Donner le nom des pièces formant la pièce nommée 'billard'. Le résultat est la liste {boule, canne, table}

R6. Donner le nom de chaque matière avec son nombre de pièces de bases.

R7. Quelles sont les matières pour lesquelles il existe au moins 3 pièces de base ?

Question 5 : Définition des méthodes

Compléter les types de données Piece_base et Piece_Composite avec des méthodes écrites en PL/SQL.

1. Écrire trois méthodes calculant respectivement le *volume*, la *masse* et le *prix* d'une pièce de base.
2. Écrire une méthode *nb_pieces_base* calculant le nombre de pièces de base entrant dans la fabrication d'une pièce composite.
3. Écrire une méthode *composee_de* donnant **le nom** des pièces de base contenues dans une pièce composite.
4. Écrire les méthodes calculant la *masse* et le *prix* d'une pièce composite.
5. (facultatif) Écrire une méthode *contenue_dans* donnant les pièces finales (*i.e.* les pièces principales n'entrant dans aucune pièce composite) contenant une pièce donnée.
6. (facultatif) Écrire les méthodes *init_pièce_base* et *init_pièce_composite* pour initialiser une pièce de base et une pièce composite.

Question 6 : Requêtes avec appel de méthode

Écrire les requêtes suivantes en SQL3. Pour chaque requête, donner le **type du résultat**.

RM 1. Quels sont le nom, le volume, la masse et le prix des pièces de base ?

RM 2. Quels sont le nom et la masse des pièces composites dont la masse est supérieure à 100 ?

RM 3. Quelles sont les pièces composites contenant plus de 10 pièces ?

RM 4. Quel est le nom des composants de chaque pièce composite ?

RM 5. Quel est l'ensemble des pièces répertoriées dans la base ?

RM 6. Quel est le nom des pièces composites qui ont une canne ?

RM 7. Quel est le nom des pièces composites qui n'ont aucun clou?

RM 8. (*) Quel est le nom des pièces finales (i.e. celle qui n'entrent dans la fabrication d'aucune pièce)?

RM 9. (*) Combien de pièces de base entrent dans la composition d'une pièce donnée ?

Note : les questions marquées avec une (*) sont facultatives.

Exercice 2. Bars à bières : Insertions

On considère le schéma SQL3 de l'exercice Bars à bières du TD précédent.

La base contient deux bars : Le 'Bar du coin', d'une capacité de 100, et le bar 'Le Select', d'une capacité de 80. Elle contient également deux bières dont les marques sont 'Chimay' et 'Stella'.

a) Ecrivez l'instruction SQL3 qui permet d'insérer le consommateur s'appelant 'Max', qui a l'identificateur 1, et qui a 22 ans. Il fréquente le bar 'Le Select' et consomme de la bière 'Stella'.

b) Max consomme aussi de la bière 'Chimay'. Ecrivez une instruction SQL3 permettant d'insérer cette information dans la base.

Exercice 3: Inscription pédagogique : Insertion, Requêtes, Méthodes

Suite de l'exercice du TD précédent

Question 1 : Instanciation

a) Insérer l'UE MLBDA code 4I801 pour 6 crédits. Les séances de cette UE sont dans l'ordre SQL, SQL3, XML.

b) Insérer l'UE PLDAC 4I101 pour 6 crédits et avec les séances Intro et Méthodologie.

c) Ajouter Alice 28 ans inscrite en Master1 avec un contrat ayant les UE 4I801 et 4I101.

d) Enregistrer la présence d'Alice à la 3ème séance de MLDBA.

Question 2 : Requêtes. Formuler les requêtes suivantes.

a) Quels sont les étudiants âgés de plus de 25 ans et inscrits à l'unité de code 4I801 ? Le résultat affiche des objets Etudiant.

b) Quel est le nom des étudiants présents à la séance 3 de l'unité nommée MLBDA ? Répondre en interrogeant la table LesU.

c) Pour chaque étudiant inscrit à l'unité de code 4I801, afficher son nom et le nombre de séances qu'il a suivies. Le résultat contient aussi le nom des étudiants inscrits à 4I801 mais qui n'ont suivi *aucune* séance.

d) Pour les étudiants présents à la 2ème séance de MLBDA, afficher leur nom et le nom des unités auxquelles ils sont inscrits. Chaque ligne du résultat est un couple nomE, nomU.

e) Pour chaque séance de chaque unité d'enseignement, quel est le nombre de **présents**? Afficher le code de l'unité, le numéro de la séance et le nombre de présents.

f) Pour chaque étudiant et chaque unité à laquelle il est inscrit, quel est le nombre de séances ?

g) Afficher le nom des étudiants présents à toutes les séances de toutes les unités de leur contrat ?

Question 3 : Méthodes

On considère la méthode créditTotal() du type Etudiant, retournant la somme des crédits des unités d'enseignement pour lesquelles l'étudiant a une note supérieure ou égale à 10 (en supposant qu'on connait les notes d'un étudiant).

a) Afficher le nom et le crédit total des étudiants qui ont un crédit total supérieur à 600.

b) On considère la méthode nbUCommunes qui renvoie le nombre d'unités d'enseignement communes entre 2 étudiants. En invoquant cette méthode, écrire la requête qui affiche les paires d'objets étudiants qui ont 10 unités en commun. La signature de nbUCommunes est .

Member function nbUCommune(Etudiant e) return number ;

c) On complète le type Etudiant avec la méthode **notes()** qui retourne l'ensemble des couples (codeU, note) de l'étudiant, et dont la signature est :

member function **notes** return EnsC1;

en supposant que les types EnsC1 et C1 ont préalablement été définis comme suit :

```
create type C1 as object (code varchar2(30), note Number);        -- un couple (code unité, note)
/
create type EnsC1 as table of C1; -- un ensemble de couples
/
```

L'étudiant 1234 est inscrit à 2 unités. Il a eu 18 en 4I801 et 16 en 4I200. Quel est le résultat de la requête : select e.notes() from LesE e where e.numéro='1234' ;

Répondre en précisant tous les constructeurs des objets (et des ensembles d'objets) affichés.

En invoquant la méthode notes(), écrire la requête : quel est le code des unités d'enseignement où l'étudiant numéro 3456 a une note inférieure à 10 ? Ne pas afficher les notes mais seulement le code des unités.

d) On connait les notes des étudiants inscrits à cette unité :

Ecrire le corps de la méthode notes (définie dans la question précédente) du type **Etudiant**.

Question 4 : Récursion

On complète le type Unité avec un attribut **prérequis** pour décrire les pré-requis d'une UE. Une UE peut avoir pour prérequis une ou plusieurs autres UE. On suppose que les pré-requis sont des UE du niveau précédent : une UE de M2 n'a que des pré-requis de M1, une UE de M1 n'a que des pré-requis de L3, ainsi de suite. Une unité de L1 n'a aucun pré-requis. On appelle **EnsU** le type du nouvel attribut prérequis.

a) Définir le type EnsU.

b) Pour le type Unité, définir la méthode récursive prérequisTousNiveaux() retourne l'ensemble des pré-requis directs et indirects d'une unité d'enseignement. Répondre un donnant la signature et le corps de la méthode prérequisTousNiveaux().

Exercice 4. Avis sur les restaurants : Instances, Requêtes , Méthodes

On considère le schéma SQL3 de l'exercice Avis sur les restaurants

Question 1 : Instances

a) Insérer le restaurant 3 étoiles Lutecia à Paris.

b) Insérer Alice qui n'a pas d'amis et qui a commenté le Lutecia en 2010 avec une note valant 5 et le texte 'délicieux'. Procéder en 2 instructions SQL3.

4) Alice retourne au Lutecia en 2013 et est déçue par la nouvelle carte. Ecrire une seule instruction SQL3 pour ajouter un nouveau commentaire en 2013 avec la note 2 et le texte 'sans saveur'.

Question 2 : Requêtes

On suppose qu'Alice a des amis. Alice veut choisir des restaurants d'après les notes attribuées par ses amis. Ecrire la requête qui affiche, pour chaque restau noté au moins une fois par un des amis d'Alice, la note moyenne que les amis d'Alice ont attribuée. Afficher le nom des restaurants avec leur note moyenne.

Question 3 : Méthodes

On complète le type Personne avec la méthode *entourage* dont la signature est :

```
member function entourage return Amis
```

Cette méthode retourne **les amis des amis** d'une personne *p*. Une personne n'est pas dans son propre entourage. Rmq : pour simplifier, l'entourage ignore les amis de *p* s'ils ne sont pas aussi des amis d'amis de *p*.

a) Ecrire la requête qui invoque cette méthode pour afficher le nom des personnes dans l'entourage d'Alice.

b) Compléter le corps de la méthode entourage. Ecrire seulement la requête qui détermine l'ensemble des personnes incluses dans l'entourage d'une personne. Pour simplifier, on suppose que l'instruction *into* permet d'affecter l'ensemble du résultat de la requête à la variable *resultat*.

Question 4 : Méthodes récursives

On complète le type Personne avec la méthode *entourageGeneral* dont la signature est :

```
member function entourageGeneral(dist number ) return Amis
```

Cette méthode retourne l'ensemble des amis directs et indirects (i.e., les amis d'amis et ainsi de suite) d'une personne *p*. Le paramètre *dist* limite l'entourage aux personnes proches de *p* telles qu'il faut parcourir au plus *dist* cercles pour les atteindre. Par exemple, si A a pour ami B qui a pour ami C qui a pour ami D. Alors D n'est pas dans l'entourageGeneral(2) de A, tandis que les autres le sont.

a) On veut connaitre les couples de personnes qui partagent une personne dans leur entourage. Ecrire la requête affichant les paires de nom de personnes qui ont au moins une personne en commun dans leur entourageGeneral(3).

b) Compléter le corps de la méthode entourageGeneral. On procède avec une requête d'union de deux sous-requêtes : la première sous requête pour obtenir les amis directs, la deuxième pour obtenir les autres personnes à inclure dans l'entourage.

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

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

<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 (utile pour définir un type mentionnant un autre type qui n'est pas encore défini).

<déclaration_type > = **create type** <type_objet> **;** --
/

<type> = <type_atomique> | <type_objet> | <type_ensembliste>

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

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 types 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

create or replace procedure <nom_procédure> **as**
<declaration_variables_locales>
begin
…
end;
@compile

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

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.