Bases de Données

C6-UML et SQL 3

Lina Soualmia

Université de Rouen
LITIS - Équipe TIBS-CISMeF
lina.soualmia@chu-rouen.fr

7 octobre 2015

Plan

Rappels C5

- Traduction des associations binaires
- Traduction des associations binaires récursives
- Traduction des associations n-aires (n $>$ 2)
- Traduction des associations d'héritage
- Traduction des contraintes d'héritage
- Traduction des associations d'agrégation
- Traduction des contraintes d'intégrité fonctionnelles (contraintes : Partition, Exclusion, Totalité, Simultanéité, Inclusion)

Associations d'héritage dans UML (1)

- Recensement des différents cas d'héritage en fonction des instances
- Modélisation des différents héritages, dans le formalisme UML, à l'aide de contraintes

Associations d'héritage dans UML (1)

- Expression des cas d'héritage à l'aide de
  - couverture
  - disjonction
    d'instances dans une population donnée
- Modélisation des différents héritages, dans le formalisme UML, à l'aide de contraintes
  - partition
  - totalité
  - exclusion
  - absence de contrainte

Contraintes d'héritage

- Disjonction & Couverture $\longrightarrow$ Partition
- Couverture & Non-Disjonction $\longrightarrow$Totalité
- Disjonction & Non-Couverture $\longrightarrow$ Exclusion
- Non-Couverture & Non-Disjonction $\longrightarrow$ Absence de contraintes

Transformation des associations d'héritage

La traduction d'une association d'héritage se fait en fonction des contraintes de l'association d'héritage
Il y a 3 familles de décomposition :

- Décomposition par *distinction*
- Décomposition *descendante* (push-down)
- Décomposition *ascendante* (push-up)

Décomposition par distinction - Contrainte de partition

- déclencheurs : sur chaque sous-classe
- pour la sur-classe :
  - procédures cataloguées pour l'Insertion et la Suppression (par sous-classe)
  - déclencheurs pour la Modification de la sur-classe vers les sous-classes

Décomposition par distinction - Contrainte de totalité

- pour la sur-classe (seulement) :
  - procédures cataloguées pour l'Insertion et la Suppression
  - déclencheurs pour la Modification de la sur-classe vers les sous-classes

Décomposition par distinction - Contrainte d'exclusion

- déclencheurs : sur chaque sous-classe

Décomposition ascendante - Contrainte de partition

- Implémentation des contraintes de type `check` au niveau de la sur-classe
- vérifier que toutes les colonnes de la sur-classe ne soient pas toutes à `NULL`, ni toutes initialisées

Transformation de l'héritage en SQL3

Héritage de types

- Existe depuis la version 9.1 d'Oracle
- Uniquement héritage de types
- Pas d'héritage multiple

Type Sur-Type

- Un type peut hériter d'un seul autre type (**sur type**)
- Un sur-type peut permettre de définir plusieurs sous-types
- Chaque sous-type est spécialisé par rapport au sur-type qui est dit plus général

Principe

Le mécanisme d'héritage est automatiquement répercuté au niveau des tables objet à partir du moment où les types définissant les tables sont issus eux-mêmes d'une hiérarchie d'héritage

Héritage de types

- Définition d'un personnel de l'université :
- Création du type de la sur-classe

```
create type PERSONNEL_TYPE as object
(Numero NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE,
Genre CHAR(1))
NOT FINAL;/*peut inclure des sous-classes*/
```

Héritage de types

- Définition d'un enseignant
- Création du type de la sous-classe

```
create type ENSEIGNANT_TYPE under PERSONNEL_TYPE
(Echelon NUMBER(2),
Indice NUMBER(5),
Specialite VARCHAR(20))
FINAL;/*pas de sous-classes*/
```


Héritage de types

- Définition d'un Biatos
- Création du type de la sous-classe

```
create type BIATOS_TYPE under PERSONNEL_TYPE
(DateEmbauche DATE,
Service VARCHAR(20),
FINAL; /*pas de sous-classes*/
```


Création des tables objet et contraintes(1)

- Création des tables en fonction des types précédemment définis
- Aucune directive ne précise l'héritage : il est induit par la hiérarchie de type existante

Création de la table personnel de l'université : les contraintes ne sont définies que dans la table PERSONNEL

```
create table PERSONNEL of PERSONNEL_TYPE
(constraint CKGenrePersonnel check (Genre in ('M','F')),
constraint PKPersonnel primary key (Numero));
```


Création de la table Enseignant

```
create table ENSEIGNANT of ENSEIGNANT_TYPE;
```

Création de la table Biatos

```
create table BIATOS of BIATOS_TYPE;
```


Les contraintes ne sont définies que dans la table PERSONNEL on hérite d'un type

Illustration : insertion des données dans la table PERSONNEL

```
insert into PERSONNEL values (1,'B','F','17-09-2001','M');
insert into PERSONNEL values (1,'B','F','17-09-2001','M');
*
ERREUR à la ligne 1:
ORA-00001: violation de contrainte unique (FB.PkPersonnel)
```

```
select * from PERSONNEL;
```

Numero	Nom	Prenom	DateNaiss	Genre
1	B	F	17-09-2001	M

Les contraintes ne sont définies que dans la table PERSONNEL on hérite d'un type

Illustration : insertion des données dans la table ENSEIGNANT

```
insert into ENSEIGNANT values (7,'B','F','17-09-1970','M',2,780,'BaDo');
1 ligne créée
insert into ENSEIGNANT values (7,'B','F','17-09-1970','M',2,780,'BaDo');
1 ligne créée
insert into ENSEIGNANT values (8,'T','F','04-05-1975','F',2,930,'BaDo');
1 ligne créée
```

```
select * from ENSEIGNANT;
```

Numero	Nom	Prenom	DateNaiss	Genre	Echelon	Indice	Specialite
7	B	F	17-09-1970	M	2	780	BaDo
7	B	F	17-09-1970	M	2	780	BaDo
8	T	F	04-05-1975	F	2	930	BaDo

Création des tables objet et contraintes(2)

- Création des tables en fonction des types précédemment définis
- Définition des contraintes au niveau des tables

Création de la table personnel de l'université

```
create table PERSONNEL of PERSONNEL_TYPE
(constraint CKGenrePersonnel check (Genre in ('M','F')),
constraint PKPersonnel primary key (Numero));
```


Création de la table Enseignant avec définition des contraintes également dans la table Enseignant et héritage d'un type

```
create table ENSEIGNANT of ENSEIGNANT_TYPE
(constraint CKGenreEnseignant check (Genre in
('M','F')),
constraint PKEnseignant primary key (Numero));
```


Création de la table Biatos avec définition des contraintes également dans la table Biatos et héritage d'un type

```
create table BIATOS of BIATOS_TYPE
(constraint CKGenreBiatos check (Genre in ('M','F')),
constraint PKbiatos primary key (Numero));
```



```
insert into ENSEIGNANT values (7,'B','F','17-09-1970','M',2,780,'BaDo');
1 ligne créée
insert into ENSEIGNANT values (7,'B','F','17-09-1970','M',2,780,'BaDo');

 * ERREUR à la ligne 1:
ORA-00001: violation de contrainte unique (FB.PkEnseignant)
```



```
insert into ENSEIGNANT values (8,'B','D','05-04-1975','M',2,780,'BaDo');
1 ligne créée
insert into ENSEIGNANT values (9,'B','D','14-03-1976','K',2,780,'BaDo');

 * ERREUR à la ligne 1:
ORA-02290: violation de contraintes unique (FB.CkGenreEnseignant) de
vérification

select * from ENSEIGNANT;
```

Numero	Nom	Prenom	DateNaiss	Genre	Echelon	Indice	Specialite
7	B	F	17-09-1970	M	2	780	BaDo
8	B	D	05-04-1975	M	2	780	BaDo

Associations d'héritage en SQL3

- Héritage de tables par la commande **under**

```
create table PERSONNEL
(Numero NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE,
Genre CHAR(1),
constraint CKGenrePersonnel check (Genre in
('M','F')),
constraint PKPersonnel primary key (Numero));
```

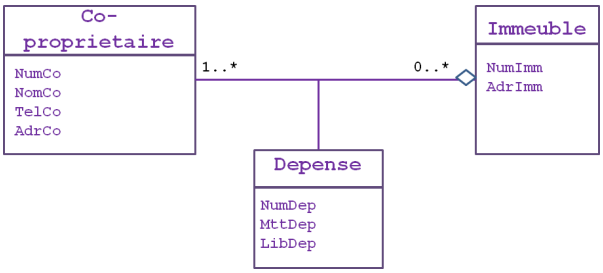

Création de la table Enseignant qui hérite de la table Personnel

```
create table ENSEIGNANT under PERSONNEL
(Echelon NUMBER(2),
Indice NUMBER(5),
Specialite VARCHAR(20));
```

Création de la table Biatos qui hérite de la table Personnel

```
create table BIATOS under PERSONNEL
(DateEmbauche DATE,
Service VARCHAR(20));
```


Traduction des associations d'agrégation



Traduction des associations d'agrégation

- Un co-propriétaire peut posséder plusieurs immeubles
- Un immeuble doit être possédé par un ou plusieurs co-propriétaires

```
create table COPROPRIETAIRE
(NumCo NUMBER(7),
NomCo VARCHAR(10),
TelCo VARCHAR(15),
AdrCo VARCHAR(50),
constraint  PKCoproprietaire  primary key (NumCo));
```



```
create table IMMEUBLE
(NumImm NUMBER(7),
AdrImm VARCHAR(10),
constraint  PKImmeuble  primary key (NumImm));
```



```
create table DEPENSE
(NumCo NUMBER(7),
NumImm NUMBER(7),
DateDep DATE,
MttDep NUMBER(10,2),
LibDep VARCHAR(50),
constraint PKDepense  primary key (NumCo,NumImm),
constraint FKDepenseNumCoCoproprietaire foreign key
(NumCo) references COPROPRIETAIRE(NumCo) on delete
cascade,
constraint FKDepenseNumImmImmeuble foreign key
(NumImm) references IMMEUBLE(NumImm) on delete
cascade);
```

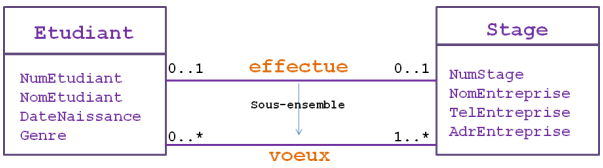

`ON DELETE CASCADE`

Indique qu'en cas de tentative de suppression d'une ligne possédant une clé référencée par des clés étrangères dans des lignes d'autres tables, toutes les lignes contenant ces clés étrangères sont également supprimées.

Traduction des contraintes d'intégrité fonctionnelles

- Contraintes : Partition, Exclusion, Totalité ... Inclusion
- Peuvent être définies ou programmées via :
  - la déclaration de contraintes (`constraint`)
  - la programmation de fonctions (`functions`)
  - la programmation de procédures (`procedures`)
  - la programmation de paquetages (`packages`)
  - la programmation de déclencheurs (`triggers`)

Contrainte d'inclusion :



Une table par classe est créée

```
create table STAGE
(NumStage NUMBER(2),
NomEntreprise VARCHAR(40),
TelEntreprise VARCHAR(15),
AdrEntreprise VARCHAR(50),
constraint PKStage  primary key (NumStage));
```

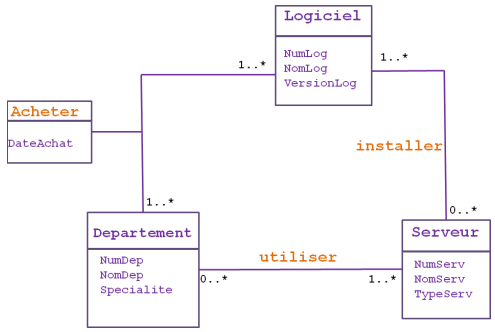


```
create table ETUDIANT
(NumEtudiant NUMBER(7),
NomEtudiant VARCHAR(10),
DateNaissance DATE,
Genre CHAR (1),
NumStage NUMBER(7),
constraint PKEtudiant primary key (NumEtudiant),
constraint FKEtudiantNumStageStage foreign key
(NumStage)references STAGE(NumStage),
constraint CKEtudiantGenre check (Genre in('M','F'));
```

```
create table VOEUX
(NumEtudiant NUMBER(7),
NumStage NUMBER(7),
constraint PKVoeux primary key (NumEtudiant,NumStage),
constraint FKVoeuxNumEtudiantEtudiant foreign key
(NumEtudiant)references ETUDIANT(NumEtudiant),
constraint FKVoeuxNumStageStage foreign key (NumStage)
references STAGE(NumStage));
```



```
alter table ETUDIANT add constraint
FKEffectuerInclusionVoeuxforeign key
(NumEtudiant,NumStage)
referencesVOEUX(NumEtudiant,NumStage);
```

Le logiciel doit être installé sur un serveur du département qui a acheté le programme

Un logiciel L acheté par le département D est installé sur un serveur S destiné entre autres à ce département

Une table par classe est créée

```
create table DEPARTEMENT
(NumDep NUMBER(7),
NomDep VARCHAR(10),
Specialite VARCHAR(20),
constraint PKDepartement primary key (NumDep));
```



```
create table LOGICIEL
(NumLog NUMBER(7),
NomLog VARCHAR(10),
VersionLog VARCHAR(10),
constraint PKLogiciel primary key (NumLog));
```



```
create table SERVEUR
(NumSer NUMBER(7),
NomServ VARCHAR(10),
TypeServ VARCHAR(10),
constraint PKServeur primary key (NumServ));
```


Une table par association ou par classe-association

```
create table ACHETER
(NumDep NUMBER(7),
NumLog NUMBER(7),
DateAchat DATE
constraint PKAcheter primary key (NumDep,NumLog),
constraint FKAcheterNumDepDepartement foreign key
(NumDep)references DEPARTEMENT(NumDep),
constraint FKAcheterNumLogLogiciel foreign key
(NumLog)references LOGICIEL(NumLog));
```



```
create table UTILISER
(NumDep NUMBER(7),
NumServ NUMBER(7),
constraint PKUtiliser primary key (NumDep,NumLog),
constraint FKUtiliserNumDepDepartement foreign key
(NumDep)references DEPARTEMENT(NumDep),
constraint FKUtiliserNumServServeur foreign key
(NumServ)references SERVEUR(NumServ));
```

```
create table INSTALLER
(NumLog NUMBER(7),
NumServ NUMBER(7),
constraint PKInstaller primary key (NumLog,NumServ),
constraint FKInstallerNumLogLogiciel foreign key
(NumLog)references LOGICIEL(NumLog),
constraint FKInstallerNumServServeur foreign key
(NumServ)references SERVEUR(NumServ));
```


Contrainte d'inclusion :

- programmation d'un déclencheur :
- ex. : Un logiciel L acheté par le département D est installé sur un serveur S, destiné entre autres à ce département


```
create or replace trigger TrigContrainteInclusion
before insert on INSTALLER
for each row
declare
    LOGIC NUMBER(7);
    SERV NUMBER(7);
begin
    select Acheter.NimLog, Utiliser.NumServ into LOGIC, SERV
    from ACHETER, UTILISER where
    Acheter.NumDep=Utiliser.NumDep and
    Acheter.NumLog=:new.NumLog and
    Utiliser.NumServ=:new.NumServ;
exception
    when no_data_found then
    raise_application_error(-20100,'Le logiciel doit être installé sur
un serveur du département acheteur');
```


Programmation Objet - SQL3

Programmation Objet SQL3

- Objet Relationnel - Objet
- Passage UML → Objet relationnel

Le modèle objet

Le modèle objet ne gère pas :

- requêtes ad hoc
- les vues
- les contraintes d'intégrité déclaratives
- les clés étrangères

La conception de BDO est très dépendante de l'application

Le modèle relationnel-objet

- Le modèle relationnel-objet :
  - c'est une extension du modèle relationnel avec des notions qui comblent les plus grosses lacunes du modèle relationnel
- La compatibilité est ascendante :
  - ce qui fonctionne pour le modèle retationnel fonctionne dans le modèle objet-relationnel

Pourquoi étendre le modèle relationnel ?

- la reconstruction d'objets complexes éclatés en tables relationnelles est très coûteuse : nombreuses jointures
- pour échapper aux jointures, le modèle objet-relationnel permet :
  - les références : implantation de structures complexes
  - attributs multivalués (listes, ensembles, tableaux)
- l'utilisation de références facilite l'utilisation de données très volumineuses en permettant leur partage à moindre coût (sans jointure)

Comparaison :

- modèle relationnel : impossibilité de définir de nouveaux types
- modèle objet-relationnel : possibilité de définir de nouveaux types :
  - simples
  - structurés
  - fonctions et procédures associées
  - modèle objet-relationnel : supporte l'héritage de type

Avantages du modèle relationnel :

- facilité et efficacité des requêtes complexes dans les grandes bases de données
- spécification des contraintes d'intégrité sans programmation
- théorie sous-jacente solide et normes reconnues

Autres causes de non utilisation des SGBDOO

- inertie de l'existant : très nombreuses bases de données relationnelles utilisées
- pas de normalisation des SGBDO
- moins de souplesse pour s'adapter à plusieurs applications
- peu d'informaticiens formés aux SGBDO

Nouvelles possibilités avec le relationnel-objet :

- définition de nouveaux types complexes avec fonctions de manipulation
- une colonne peut contenir une collection : liste, ensemble, ...
- une ligne est considérée comme un objet avec un identificateur OID
- utilisation de références aux objets extension de SQL : SQL3 (ou SQL99) pour la recherche et la manipulation des données

Inconvénients du modèle objet-relationnel :

- ne s'appuie pas sur une théorie solide comme le modèle relationnel
- pas de standardisation : implantations différentes, partielles dans les SGBD

Schéma relationnel SQL2

PROFESSEUR(NumProf,NomProf,Specialite,DateEntree,DerPromo,SalBase,SalActuel)

COURS(NumCours,NomCours,NbHeures,Annee)

CHARGE(NumProf*, NumCours*)


```
create table  COURS
(NumCours NUMBER(2) NOT NULL,
NomCours VACHAR(20) NOT NULL,
NbHeures NUMBER(2),
Annee NUMBER(1),
constraint PKCours primary key (NumCours));
```



```
create table PROFESSEURS
(NumProf NUMBER(4) NOT NULL,
NomProf VARCHAR(25) NOT NULL,
Specialite VARCHAR2(20),
DateEntree DATE,
DerPromo DATE,
SalaireBase NUMBER
SalaireActuel NUMBER
constraint  PKProfesseurs  primary key (NumProf);
```

```
create table  CHARGE
(NumProf NUMBER(4) NOT NULL,
NumCours NUMBER(4) NOT NULL,
constraintPkCharge primary key
(NumCours,NumProf));
```



```
alter table  CHARGE
  add constraint FKChargeCours foreign key (NumCours)
  references COURS(NumCours);

alter table  CHARGE
  add constraint FKChargeProfesseur foreign key  (NumProf)
  references PROFESSEURS(NumProf);
```


Schéma relationnel objet :

COURS(NumCours,NomCours,NbHeures,Annee)

PROFESSEURS(NumProf,NomProf,Specialite,DateEntree,
DerPromo,SalBase,SalActuel,**Ensemble(COURS)**)

en SQL3

```
create type  CoursType as object
(NumCours NUMBER(2),NomCours VARCHAR2(20),NbHeures
NUMBER(2),Annee NUMBER(1));
```

Création du type table

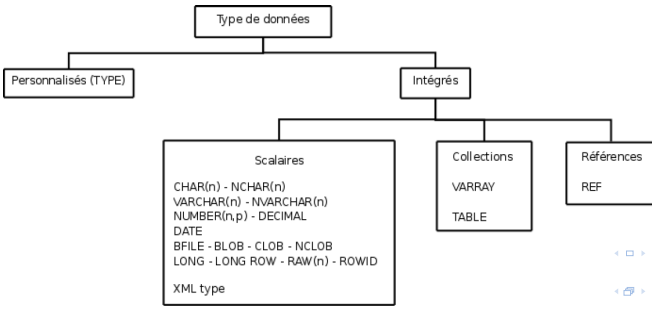
```
create type  LesCoursType as table of CoursType;
```

```
create type  ProfesseurType as object
(NumProf NUMBER(4),NomProf VARCHAR2(25),Specialite
VARCHAR2(20),..., Cours LesCoursType);
```

Création de la table maître

```
create table  Professeur of ProfesseurType
(primary key(NumProf)),
nested table Cours store as TableCoursP;
```

Cours est une colonne de type table qui fait le lien entre la table maître et la table imbriquée ; TableCoursP est le nom de la table qui contient les lignes de la table imbriquée



Types de données Oracle

Sous Oracle, 3 catégories d'objets :

- Objets colonne (`column objects`) :
  - stockés en tant que colonne structurée dans une table relationnelle
- Objets ligne (`row objects`) :
  - stockés en tant que ligne d'une table objet
  - possèdent un identificateur unique appelé OID (Object Identifier)
  - peuvent être indexés et partitionnés
- Objets non persistants :
  - non stockés ni dans une colonne d'une table relationnelle
  - ni dans une ligne d'une table objet
  - Ces objets n'existent que durant l'exécution d'un programme PL/SQL

- Définition de chaque objet à partir d'un type décrivant
  - une structure de données se positionnant dans une hiérarchie d'héritage
  - des méthodes
- Utilisation d'un type :
  - Construire d'autres types
  - Définir une ou plusieurs tables objet
  - Définir une colonne d'une table relationnelle
  - Construire des vues objet

- Création d'un type :

```
CREATE [OR REPLACE] TYPE Schéma.NomType
[AS OBJECT | UNDER] Schéma.NomSurType
```

définition de la structure
`(Colonne1 type1, Colonne2 type2 ....`

définition du comportement
`méthode1(paramètres1),méthode2(paramètres2)...)`
`[[NOT]INSTANTIABLE]`

positionnement dans le graphe d'héritage
`[[NOT]FINAL]`

- Directives FINAL et NOT FINAL : positionnement d'un type dans le graphe d'héritage
- Directive NOT FINAL : à appliquer aux types génériques
- Par défaut, tout type est FINAL
- Un type FINAL ne peut servir à définir des sous-types


```
create type  adresseT as object
(NRue NUMBER(3),Rue VARCHAR(40),Ville VARCHAR(30);

create type personnelT as object
(Nom VARCHAR(10),Prenom VARCHAR(10),Adresse adresseT)
NOT FINAL;

create type  enseignantT as object under personnelT
(Echelon NUMBER,Indice NUMBER)
FINAL;
```


- Directives INSTANTIABLE et NOT INSTANTIABLE
  - capacité d'instancitation d'un type
  - tous les types crées sont par défaut INSTANTIABLE
- NOT INSTANTIABLE : similaire à la notion de classe abstraite
- Chaque type possède
  - un constructeur permettant de créer des objets (persistants ou non) à l'aide de la commande NEW ou au sein d'une commande INSERT
  - un constructeur (par défaut) et plusieurs dans le cas de surcharge
- Un type NOT INSTANTIABLE ne peut pas être FINAL
- Un sous-type NOT INSTANTIABLE peut hériter d'un type INSTANTIABLE


```
create type personnelT as object
(Nom VARCHAR(10),Prenom VARCHAR(10),Adresse adresseT)
NOT INSTANTIABLE NOT FINAL;

create type  enseignantT as object under personnelT
(Echelon NUMBER,Indice NUMBER)
INSTANTIABLE FINAL;
```


Suppression d'un type

```
DROP TYPE NomType [FORCE|VALIDATE];
```

Directives :

- FORCE : suppression du type même s'il y a des objets de ce type dans une base
  Oracle marque les colonnes dépendant de ce type par UNUSED et elles deviennent inaccessibles (non recommandé)
- VALIDATE : Vérification si les instances du type à supprimer peuvent être substitués par un sur-type

```
DROP TYPE PersonnelT FORCE;
```


Spécification de l'objet :

```
CREATE TYPE Bank_Account AS OBJECT (
    acct_number INTEGER(5),
    balance            REAL,
    status             VARCHAR2(10),

    MEMBER PROCEDURE open
          (amount IN REAL),

    MEMBER PROCEDURE verify_acct
          (num IN INTEGER),

    MEMBER PROCEDURE close
          (num IN INTEGER, amount OUT REAL)
);
/
```

Définition des méthodes associées à l'objet :

```
CREATE TYPE BODY Bank_Account AS

   MEMBER PROCEDURE open (amount IN REAL) IS
   BEGIN -- open account with initial deposit
      IF NOT amount > 0 THEN
         RAISE_APPLICATION_ERROR(-20104, 'bad amount');
      END IF;
      -- SELECT acct_sequence.NEXTVAL INTO acct_number FROM dual;
      status := 'open';
      balance := amount;
   END open;
```

Description des méthodes associées à l'objet :

```
   MEMBER PROCEDURE verify_acct (num IN INTEGER) IS
   BEGIN -- check for wrong account number or closed account
      IF (num <> acct_number) THEN
         RAISE_APPLICATION_ERROR(-20105, 'wrong number');
      ELSIF (status = 'closed') THEN
         RAISE_APPLICATION_ERROR(-20106, 'account closed');
      END IF;
   END verify_acct;

   MEMBER PROCEDURE close (num IN INTEGER, amount OUT REAL) IS
   BEGIN -- close account and return balance
      verify_acct(num);
      status := 'closed';
      amount := balance;
   END close;
END;
```

Extraction de la description d'un type

Définition de nouvelles vues du Dictionnaire des Données pour prendre en compte les types

```
create type  emp_type as object
(ninsee VARCHAR2(13),age NUMBER,nom VARCHAR2(30))
```

Description de la structure au premier niveau d'un type :

```
SQL> DESC emp_type
```


Extraction de la description d'un type

Exemples de vues : (USER_..., DBA_..., ALL_...)

Description :

- des collections : USER_COLL_TYPES
- des index sur les types : USER_INDEX_TYPES
- des types d'une manière générale : USER_TYPES
- des attributs des types : USER_TYPE_ATTRS
- des méthodes des types : USER_TYPE_METHODS
- des versions des types : USER_TYPE_VERSIONS

Tables relationnelles

```
--   Table : MAGASINS2 SQL2
create table MAGASINS2
(   NUMMAG          INTEGER      ,
    NOMMAG          CHAR(30)     ,
    TELMAG          CHAR(15)     ,
    ADRNUMMAG       VARCHAR2(10) ,
    ADRRUEMAG       VARCHAR2(50) ,
    ADRCPMAG        VARCHAR2(10) ,
    ADRVILLEMAG     VARCHAR2(50) ,
    ADRPAYSMAG      VARCHAR2(50) ,
     constraint PK_MAGASINS2
        primary key (NUMMAG) );
```

```
--   Table : CLIENTS2 SQL2
create table CLIENTS2
(   NUMCLI          INTEGER      ,
    NOMCLI          CHAR(20)     ,
    TELCLI          CHAR(15)     ,
    ADRNUMCLI       VARCHAR2(10) ,
    ADRRUECLI       VARCHAR2(50) ,
    ADRCPCLI        VARCHAR2(10) ,
    ADRVILLECLI     VARCHAR2(50) ,
    ADRPAYSCLI      VARCHAR2(50) ,
     constraint PK_CLIENTS2
        primary key (NUMCLI));
```

```
insert into MAGASINS2 values (1, 'FB', '0145454545', '13', 'Avenue de la paix',
                              '75015', 'Paris', 'France');
```

NUMMAG	NOMMAG	TELMAG	ADRNU	ADRRUEMAG	ADRCP	ADRVILLEMA	ADRPAYSMAG
1	FB	0145454545	13	Avenue de la paix	75015	Paris	France
2	FB	0155555555	20	Avenue de la liberté	06100	Nice	France
3	FB	0155555555	10	Avenue des Amis	6050	Bruxelles	Belgique
4	FB	71226002	10	Avenue du soleil	1001	Tunis	Tunisie

NUMCLI	NOMCLI	TELCLI	ADRNU	ADRRUECLI	ADRCP	ADRVILLECL	ADRPAYSCL
1	TRAIFOR	0645454545	13	Avenue de la paix	75015	Paris	France
2	CLEMENT	0607080910	17	Avenue de la paix	75015	Paris	France
3	SOUCY	98980307	77	Route de la corniche	4001	Sousse	Tunisie

Création d'un type :

Première extension du modèle relationnel : Types Abstraits de Données (TAD)

- TAD (contexte BD) :
  - Nouveau type d'attribut défini par l'utilisateur
  - Enrichissement de la collection existante de types disponibles par défaut (number,date,char,varchar...
  - Structure de données partagée
    - Utilisation du type dans une ou plusieurs tables
    - Participation à la composition d'un ou plusieurs autres types
- Remarques :
  - Un TAD inclut des méthodes qui sont des procédures ou des fonctions
  - Elles permettent de manipuler les objets du type abstrait


```
create  type ADRESSE_TYPE as object
  ( ADRNUM      VARCHAR2(10),
    ADRRUE      VARCHAR2(50),
    ADRCP       VARCHAR2(10),
    ADRVILLE    VARCHAR2(50),
    ADRPAYS     VARCHAR2(50) )
/
```

```
 create type MAG_TYPE as object
 (   NUMMAG    INTEGER  ,
     NOMMAG    CHAR(30),
     TELMAG    CHAR(15),
     ADRMAG    ADRESSE_TYPE )
 /
```

```
create type CLI_TYPE as object
(   NUMCLI    INTEGER  ,
    NOMCLI    CHAR(30),
    TELCLI    CHAR(15),
    ADRCLI    ADRESSE_TYPE )
/
```

```
create table MAGASINS3 OF MAGTYPE
(constraint  PKMagasins3  primary key (NUMMAG));

create table CLIENTS3 OF CLITYPE
(constraint  PKClients3  primary key (NUMCLI));
```


Remarques :

- Un type ne peut pas contenir de contraintes (NOT NULL,CHECK,UNIQUE,DEFAULT,PRIMARY KEY,FOREIGN..
- Les contraintes doivent être déclarées au niveau de la table objet
- Accès à la description des types à partir du Dictionnaire de Données :

```
SQL > select table_name, object_id_type, table_type_own
          table_type from user_object_tables;
```


Création/description d'une table Exemples

```
SQL> desc clients2
```

Nom	NULL ?	Type
NUMCLI	NOT NULL	NUMBER(38)
NOMCLI		CHAR(20)
TELCLI		CHAR(15)
ADRNUMCLI		VARCHAR2(10)
ADRRUECLI		VARCHAR2(50)
ADRCPCLI		VARCHAR2(10)
ADRVILLECLI		VARCHAR2(50)
ADRPAYSCLI		VARCHAR2(50)

```
SQL> desc clients3
```

Nom	NULL ?	Type
NUMCLI	NOT NULL	NUMBER(38)
NOMCLI		CHAR(30)
TELCLI		CHAR(15)
ADRCLI		ADRESSE_TYPE

- Si les Object identier (OID) sont basés sur la clé primaire : utilisation de l'option `primary key`

```
create table CLIENTS3 OF CLITYPE
(constraint  PKClients3  primary key (NUMCLI))
object identifier is primary key;
```

- Index sur OID :

```
create table CLIENTS3 OF CLITYPE
(constraint  PKClients3  primary key (NUMCLI));
object identifier is system generated OIDINDEX
ndxclients3;
```


Instanciation : insertion

```
insert into MAGASINS3 values (MAG_TYPE(1, 'FB', '0145454545',
        ADRESSE_TYPE('13', 'Avenue de la paix', '75015', 'Paris', 'France'))
insert into MAGASINS3 values (MAG_TYPE(2, 'FB', '0155555555',
        ADRESSE_TYPE('20', 'Avenue de la liberté', '06100', 'Nice', 'France'
insert into MAGASINS3 values (MAG_TYPE(3, 'FB', '0155555555',
        ADRESSE_TYPE('10', 'Avenue des Amis', '6050', 'Bruxelles', 'Belgique
insert into MAGASINS3 values (MAG_TYPE(4, 'FB', '71226002',
        ADRESSE_TYPE('10', 'Avenue du soleil', '1001', 'Tunis', 'Tunisie')))
```

```
SQL> select * from magasins3 ;
NUMMAG  NOMMAG  TELMAG      ADRMAG(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
  1       FB    0145454545    ADRESSE_TYPE('13', 'Avenue de la paix',
                                           '75015', 'Paris', 'France')
  2       FB    0155555555    ADRESSE_TYPE('20', 'Avenue de la liberté',
                                           '06100', 'Nice', 'France')
  3       FB    0155555555    ADRESSE_TYPE('10', 'Avenue des Amis',
                                           '6050', 'Bruxelles', 'Belgique')
  4       FB    71226002      ADRESSE_TYPE('10', 'Avenue du soleil',
                                           '1001', 'Tunis', 'Tunisie')
```



```
insert into CLIENTS3 values (CLI_TYPE(1, 'TRAIFOR', '0645454545',
        ADRESSE_TYPE('13', 'Avenue de la paix', '75015', 'Paris', 'France')
insert into CLIENTS3 values (CLI_TYPE(2, 'CLEMENT', '0607080910',
        ADRESSE_TYPE('17', 'Avenue de la paix', '75015', 'Paris', 'France')
insert into CLIENTS3 values (CLI_TYPE(3, 'SOUCY', '98980307',
        ADRESSE_TYPE('77', 'Route de la corniche', '4001', 'Sousse', 'Tunis
```

```
SQL> Select * from clients3 ;
NUMCLI  NOMCLI     TELCLI     ADRCLI(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
  1     TRAIFOR  0645454545   ADRESSE_TYPE('13', 'Avenue de la paix', '75015',
                                          'Paris', 'France')
  2     CLEMENT  0607080910   ADRESSE_TYPE('17', 'Avenue de la paix', '75015',
                                          'Paris', 'France')
  3     SOUCY     98980307    ADRESSE_TYPE('77', 'Route de la corniche', '4001',
                                          'Sousse', 'Tunisie')
```


Instanciation Table objet-relationnelle :

- Table dépendante d'un type
- Enregistrements (lignes) dans cette table considérés comme des objets car ils possèdent tous un OID unique

```
SQL> SELECT * FROM clients3 ;
NUMCLI  NOMCLI     TELCLI     ADRCLI(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
  1     TRAIFOR  0645454545   ADRESSE_TYPE('13', 'Avenue de la paix', '75015',
                                          'Paris', 'France')
  2     CLEMENT  0607080910   ADRESSE_TYPE('17', 'Avenue de la paix', '75015',
                                          'Paris', 'France')
  3     SOUCY     98980307    ADRESSE_TYPE('77', 'Route de la corniche', '4001',
                                          'Sousse', 'Tunisie')
```


Renvoi des OID des objets de la table :

```
SQL> SELECT REF(c) FROM clients3 c ;

REF(C)
--------------------------------------------------------------------------------
0000280209E9E229206EDF47DF9996946C4BBD571C4EB9AF259F2F42BC813
0000280209550141E8898C4859AF0F3D48FA3041944EB9AF259F2F42BC813
0000280209C2C96804847047F6856499690AAC9E254EB9AF259F2F42BC813
```


Mises à jour Modifications/Suppressions de lignes ou d'objets

- Mise à jour d'une colonne standard

```
update CLIENTS3
set NOMCLI='CBON' where NUMCLI=2;
```

- Modification d'une colonne appartenant à un type imbriqué

```
update CLIENTS3 c
set c.ADRCLI.ADR.VILLE='MAVILLE' where
c.NUMCLI=2;
```

- Suppression d'objet

```
delete from CLIENTS3
where NUMCLI=3;
delete from CLIENTS3 c
where upper(c.ADRCLI.ADRPAYS)='FRANCE';
```


Utilisation de colonnes standards

```
select numcli, nomcli from clients3;
```

NUMCLI	NOMCLI
1	TRAIFOR
2	CLEMENT
3	SOUCY

Utilisation d'une colonne appartenant à un type imbriqué

```
select numcli, nomcli,c.ADRCLI.ADRPAYS
from clients3 c;
```

NUMCLI	NOMCLI	ADRCLI.ADRPAYS
1	TRAIFOR	France
2	CLEMENT	France
3	SOUCY	Tunisie

Interrogations avec formatage

```
col nom format A10
col loc format A15
select numcli as cli, nomcli as nom,
  c.ADRCLI.ADRVILLE || ' ' || c.ADRCLI.ADRPAYS as loc
from clients3 c;
```

CLI	NOM	LOC
1	TRAIFOR	Paris France
2	CLEMENT	Paris France
3	SOUCY	Sousse Tunisie

Interrogations avec contraintes

```
SQL> col c.ADRCLI.ADRPAYS format A10
SQL> col c.ADRCLI.ADRVILLE format A10
SQL> select numcli, nomcli,c.ADRCLI.ADRPAYS,
  2  c.ADRCLI.ADRVILLE from clients3 c
  3  WHERE upper(c.ADRCLI.ADRVILLE) like 'P%';
```

NUMCLI	NOMCLI	ADRCLI.ADRPAYS	ADRCLI.ADRVILLE
1	TRAIFOR	France	Paris
2	CLEMENT	France	Paris

Tables imbriquées

Table imbriquée (NESTED TABLE : collection non ordonnée et non limitée d'éléments de même type)

Exemple : table Département qui contient plusieurs employés

1 table contenant une colonne (table) : Association du type 1-N

NumDep	Budget	Employés		
		NInsee	Nom	Age

Création

```
create type EmpType as object
(ninsee VARCHAR2(13),age NUMBER ,nom VARCHAR2(30));
```

Création du type table

```
create type EmpsType as table of EmpType;
```

```
create type DepartementType as object
(NumDep VARCHAR(11),Budget NUMBER employes EmpsType;
```

Création de la table maître

```
create table Departement of DepartementType
(primary key(NumDep))
nested table employes store as tabemp;
```

- clause NESTED TABLE : définition d'une table imbriquée
- clause STORE AS : nommage de la structure interne qui stocke les enregistrements de cette table imbriquée


```
SQL> desc departement
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------------------
 NUMDEP                                    NOT NULL VARCHAR2(11)
 BUDGET                                             NUMBER
 EMPLOYES                                           EMPS_TYPE

SQL> desc emps_type
 emps_type TABLE OF EMP_TYPE
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------------------
 NINSEE                                             VARCHAR2(13)
 AGE                                                NUMBER
 NOM                                                VARCHAR2(30)
```


Insertion des données dans une table imbriquée

```
insert into departement values ('D1', 100000, emps_type());
insert into departement values ('D2', 200000, emps_type());
```

```
SQL> select * from departement ;
NUMDEP          BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ------------------------------
D1              100000 EMPS_TYPE()
D2              200000 EMPS_TYPE()
```


Attention : dans l'exemple suivant, la table vide est non initialisée

```
insert into departement (numdep, budget)
            values ('D3', 300000);
```

```
SQL> select * from departement ;
NUMDEP          BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ------------------------------
D1              100000 EMPS_TYPE()
D2              200000 EMPS_TYPE()
D3              300000
```


Insertion des données dans une table imbriquée

```
insert into departement values ('D4', 400000,
            emps_type(emp_type('N5', 25, 'Bibi'),
                      emp_type('N6', 26, 'Cici'),
                      emp_type('N7', 27, 'Didi'),
                      emp_type('N8', 28, 'Fifi')));
```

```
SQL> select * from departement ;
NUMDEP     BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ ---------- ------------------------------------------------
D1         100000 EMPS_TYPE()
D2         200000 EMPS_TYPE()
D3         300000
D4         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N6', 26, 'Cici'),
                            EMP_TYPE('N7', 27, 'Didi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
```

Remarque : La commande `INSERT` avec les constructeurs des types de la `NESTED TABLE` :

- stocke un objet dans la table
- initialise la table imbriquée associée avec des enregistrements

Insertion des données dans une table imbriquée

```
insert into departement values ('D5', 400000,
            emps_type(emp_type('N5', 25, 'Bibi'),
                      emp_type('N8', 28, 'Fifi')));

SQL> select * from departement ;
NUMDEP     BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ ---------- ------------------------------------------------
D1         100000 EMPS_TYPE()
D2         200000 EMPS_TYPE()
D3         300000
D4         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N6', 26, 'Cici'),
                            EMP_TYPE('N7', 27, 'Didi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
D5         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
```


Insertion avec `THE` dans une table imbriquée (D1 et D2 étaient initialisés à vide)

```
insert into THE (select d.employes from departement d
                 where d.numdep = 'D1')
          values ('N1', 21, 'CLEMENT');
insert into THE (select d.employes from departement d
                 where d.numdep = 'D2')
          values ('N2', 22, 'CLEMENTINE');
```



```
SQL> select * from departement ;
NUMDEP     BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ ---------- ------------------------------------------------
D1         100000 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2         200000 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'
D3         300000
D4         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N6', 26, 'Cici'),
                            EMP_TYPE('N7', 27, 'Didi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
```

Remarques :

- Commande `INSERT INTO THE (SELECT ...)` : stockage d'un enregistrement dans la table imbriquée désignée par `THE`
- `SELECT` après le `THE` : Retourne un seul objet, ce qui permet de sélectionner la table imbriquée associée

Insertion avec `THE` dans une table imbriquée (D3 n'était pas initialisé à vide)

Insertion d'un employé dans le département D3 alors que celui-ci n'a pas été initialisé


```
SQL> insert into the (select d.employes from departement d
  2  where d.numdep = 'D3') values ('N3', 23, 'NEMARCHEPAS')
insert into the (select d.employes from departement d
                 where d.numdep = 'D3')
*
ERREUR à la ligne 1 :
ORA-22908: référence à une valeur de table NULL
```

Explications :

1. Le département D3 est bien un objet de la table Departement
2. mais il ne possède pas de table imbriquée
3. car celle-ci n'a pas été créée lors de l'insertion. $\Longrightarrow$ Il faut détruire l'objet D3 puis le recréer !

Mise à jour de la table principale

```
update departement d
       set d.budget = d.budget * 1.5
           where d.budget <= 200000 ;

SQL> select * from departement ;
NUMDEP     BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ ---------- ------------------------------------------------
D1         150000 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2         300000 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'
D3         300000
D4         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N6', 26, 'Cici'),
                            EMP_TYPE('N7', 27, 'Didi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
D5         400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                            EMP_TYPE('N8', 28, 'Fifi'))
```


- Mise à jour de la table principale selon un prédicat dans la table imbriquée

```
update departement d set d.budget = d.budget + 777
       where  exists (select * from
       the ( select dt.employes from departement dt
             where dt.numdep =
             d.numdep ) nt
     where nt.age < 25 ) ;
```

- Description : Requête qui retourne les employés de chaque département
  `select dt.employes from departement dt where dt.numdep=d.numdep;`
- Condition sur un attribut de la table imbriquée : `where nt.age < 25`
- Alias de la table imbriquée :`nt`

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ -------- ----------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'
D3       300000
D4       400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```


Mise à jour de la table principale selon un prédicat dans la table imbriquée

```
update departement d set d.budget = d.budget + 999
       where  exists
              (select * from the
              (select dt.employes from departement dt
                            where dt.numdep = d.numdep ) nt
              where nt.age > 25 ) ;
```



```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ -------- ----------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'
D3       300000
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Remarque : les mêmes employés sont dans deux départements

Mise à jour dans la table imbriquée

```
update
    the (select d.employes from departement d
                           where d.numdep = 'D2' ) nt
    set nt.age = 44
        where nt.ninsee = 'N2' ;
```



```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ -------- ----------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'
D3       300000
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Il est impossible de modifier plusieurs enregistrements de différentes tables imbriquées avec une seule commande UPDATE