Corrigés détaillés des exercices

Diagramme des tables objet

La figure suivante vous aidera dans la compréhension des manipulations des objets de la base.

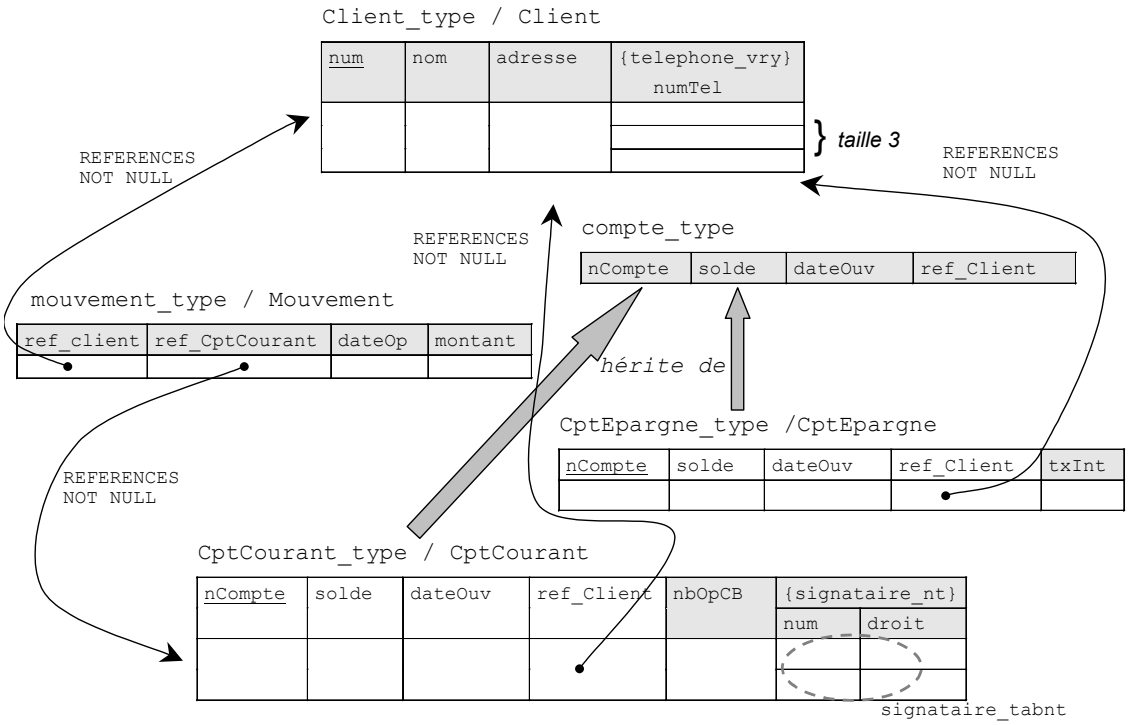

Figure S-1. Diagramme des tables objets

Types SQL3 Oracle

La création des types SQL3 Oracle correspondant au diagramme UML et aux indications fournies est le suivant. Les conventions d'écriture sont indiquées en gras pour chaque variable dès la première fois qu'elle apparaît.

La traduction de l'association *un-à-plusieurs* `Possede` est réalisée via le *varray* `telephone_vry` qui permet de stocker au plus trois numéros de téléphone pour un client donné.

```
--Création du type client
CREATE TYPE telephone_elt_vry_type AS OBJECT
 (numTel VARCHAR2(14))
/
CREATE TYPE telephone_vry_type AS VARRAY(3) OF telephone_elt_vry_type
/
CREATE TYPE client_type AS OBJECT
 (num NUMBER(5), nom VARCHAR2(30), adresse VARCHAR2(30),
  telephone_vry telephone_vry_type)
/
```

La traduction de l'association *un-à-plusieurs* `Proprietaire` est réalisée à l'aide de la référence `ref_Client` qui permet de relier un compte à son client propriétaire. Le type `compte_type` est déclaré `NOT FINAL` car deux sous-types vont être dérivés (les comptes épargnes et courants), et `NOT INSTANTIABLE` car on ne stocke pas dans la base de données des comptes n'étant ni courant ni épargne.

```
--Création du type compte
CREATE TYPE compte_type AS OBJECT
 (nCompte VARCHAR2(5), solde NUMBER(10,2), dateOuv DATE,
  ref_Client REF client_type)
 NOT FINAL NOT INSTANTIABLE
/
```

La traduction de l'association *plusieurs-à-plusieurs* `Signataire` est réalisée à l'aide de la *nested table* `signataire_nt` qui contiendra les numéro des clients signataires avec leurs droits sur un compte courant donné.

```
--Création du type compte courant
CREATE TYPE signataire_elt_nt_type AS OBJECT
 (num NUMBER(5), droit CHAR(1))
/
CREATE TYPE signataire_nt_type AS TABLE OF signataire_elt_nt_type
/
CREATE TYPE cptCourant_type UNDER compte_type
 (nbOpCB NUMBER(5), signataire_nt signataire_nt_type)
/
--Création du type compte épargne
CREATE TYPE cptEpargne_type UNDER compte_type
 (txInt NUMBER(2,1))
/
```

La traduction de l'association *n-aire* `Operations` est réalisée à l'aide de la table `Mouvement` contenant deux références (une vers le type d'un client, une autre vers le type d'un compte courant.

```
--Création du type mouvement
CREATE TYPE mouvement_type AS OBJECT
 (ref_Client REF client_type, ref_CptCourant REF cptCourant_type,
  dateOp DATE, montant NUMBER(8,2))
/
```

Tables SQL3 Oracle

Les contraintes relatives à l'intégrité référentielle sur les tables sont surlignées, celles relatives aux domaines de valeur des colonnes sont en gras.

```
--Table Client
CREATE TABLE Client OF client_type
  (CONSTRAINT pk_client PRIMARY KEY (num));

--Table compte courant
CREATE TABLE CptCourant OF cptCourant_type
  (CONSTRAINT pk_cptCourant          PRIMARY KEY (nCompte),
   CONSTRAINT nn_Courant_ref_Client CHECK (ref_Client IS NOT NULL),
   CONSTRAINT refer_Courant_Client  ref_Client REFERENCES Client)
   NESTED TABLE signataire_nt STORE AS signataire_tabnt;

ALTER TABLE signataire_tabnt
      ADD CONSTRAINT ck_droit CHECK (droit IN ('X','R','D'));

ALTER TABLE signataire_tabnt
      ADD CONSTRAINT nn_signataire_num CHECK (num IS NOT NULL);

ALTER TABLE signataire_tabnt
      ADD CONSTRAINT nn_signataire_droit CHECK (droit IS NOT NULL);

--Table compte épargne
CREATE TABLE CptEpargne OF cptEpargne_type
  (CONSTRAINT pk_cptEpargne PRIMARY KEY (nCompte),
   CONSTRAINT nn_Epargne_ref_Client CHECK (ref_Client IS NOT NULL),
   CONSTRAINT refer_Epargne_Client  ref_Client REFERENCES Client,
   CONSTRAINT ck_txInt              CHECK (txInt < 3.5));

--Table de liaison pour les opérations
CREATE TABLE Mouvement OF mouvement_type
  (CONSTRAINT refer_Mvt_Client      ref_Client REFERENCES Client,
   CONSTRAINT nn_Mvt_ref_Client     CHECK (ref_Client IS NOT NULL),
   CONSTRAINT refer_Mvt_CptCourant  ref_CptCourant REFERENCES CptCourant,
   CONSTRAINT nn_Mvt_ref_CptCourant CHECK (ref_CptCourant IS NOT NULL),
   CONSTRAINT df_dateOp     dateOp  DEFAULT (SYSDATE-2) );
```

La régénération du schéma devra inclure le script suivant avant de lancer toutes les créations.

```
DROP TABLE Mouvement;
DROP TABLE CptEpargne;
DROP TABLE CptCourant;
DROP TABLE Client;

DROP TYPE mouvement_type;
DROP TYPE cptEpargne_type;
DROP TYPE cptCourant_type;
DROP TYPE signataire_nt_type;
```

```
DROP TYPE signataire_elt_nt_type;
DROP TYPE compte_type;
DROP TYPE client_type;
DROP TYPE telephone_vry_type;
DROP TYPE telephone_elt_vry_type;
```

Insertion d'objets dans les tables

Insérons cinq clients en initialisant tous les éléments du *varray* `telephone_vry` sauf pour le premier client.

```
INSERT INTO Client VALUES
  (client_type(1, 'Albaric', 'Pont Vieux  - Vielle Toulouse',
            telephone_vry_type(telephone_elt_vry_type('05-61-75-68-39'),
                               telephone_elt_vry_type(NULL)) ));
INSERT INTO Client VALUES
  (client_type(2, 'Bidal', 'Port Royal - Paris',
            telephone_vry_type(telephone_elt_vry_type(NULL),
                               telephone_elt_vry_type(NULL),
                               telephone_elt_vry_type('06-76-85-14-89'))));
INSERT INTO Client VALUES
  (client_type(3, 'Miranda', 'Antipolis - Nice',
            telephone_vry_type(telephone_elt_vry_type(NULL),
                               telephone_elt_vry_type('04-35-60-77-89'),
                               telephone_elt_vry_type('06-81-94-44-31'))));
INSERT INTO Client VALUES
  (client_type(4, 'Payrissat', 'Salas - Ramonville St Agne',
            telephone_vry_type(telephone_elt_vry_type('05-61-75-98-44'),
                               telephone_elt_vry_type(NULL),
                               telephone_elt_vry_type('06-46-45-72-30'))));
INSERT INTO Client VALUES
  (client_type(5, 'Vielle', 'INRA - Auzeville Tolosane',
            telephone_vry_type(telephone_elt_vry_type('05-61-73-12-74'),
                               telephone_elt_vry_type('05-62-74-75-63'),
                               telephone_elt_vry_type('06-65-41-83-35'))));
```

Insérons sept comptes courants en initialisant à vide toutes les *nested tables* `signataire_nt`.

```
INSERT INTO CptCourant VALUES
      (cptCourant_type('CC1', 4030, '01-02-2001',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 1), 509,
       signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC2', 3000, '15-02-2002',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 1), 0,
       signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC3', 460, '13-05-2000',
```

```
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 678,
        signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC4', 730, '17-09-2002',
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 0,
        signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC5', 15, '10-12-1998',
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 1390,
        signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC6', 55, '16-01-1965',
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 2), 2400,
        signataire_nt_type())  );

INSERT INTO CptCourant VALUES
      (cptCourant_type('CC7', 6700, '04-03-1976',
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 3), 5600,
        signataire_nt_type())  );
```

Insérons des signataires pour les comptes courants CC2, CC3, CC6 et CC7 en ajoutant des éléments aux *nested tables* `signataire_nt`.

```
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC2')
      VALUES (signataire_elt_nt_type(2,'D'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC2')
      VALUES (signataire_elt_nt_type(2,'R'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC2')
      VALUES (signataire_elt_nt_type(3,'R'));

INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC3')
      VALUES (signataire_elt_nt_type(1,'D'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC3')
      VALUES (signataire_elt_nt_type(5,'D'));

INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC6')
      VALUES (signataire_elt_nt_type(3,'D'));

INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7')
      VALUES (signataire_elt_nt_type(2,'D'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7')
      VALUES (signataire_elt_nt_type(2,'R'));
INSERT INTO TABLE
```

```
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7')
      VALUES (signataire_elt_nt_type(2,'X'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7')
      VALUES (signataire_elt_nt_type(1,'D'));
INSERT INTO TABLE
      (SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7')
      VALUES (signataire_elt_nt_type(5,'D'));
```

Affectons chacun des six comptes épargnes CE1, CE2…, CE6 à un des trois clients suivants (numéros 2, 4 et 3).

```
INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE1', 600, '05-02-1965',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 2), 2.7 ));
INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE2', 4500, '04-12-1998',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 2), 2.9 ));

INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE3', 500, '05-03-2000',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 2.9 ));
INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE4', 500, '05-02-2001',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 2.4 ));
INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE5', 500, '13-05-1995',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 4), 3.4 ));

INSERT INTO CptEpargne VALUES
      (cptEpargne_type('CE6', 3300, '23-08-1997',
       (SELECT REF(cli) FROM Client cli WHERE cli.num = 3), 3.3 ));
```

Insérons des opérations pour chaque compte, opérations faites par le titulaire du compte ou par un signataire.

```
--Insertion de mouvements Cpt courant
--cpt Albaric pas de signataire sur CC1
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 1),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC1'),
      SYSDATE-7, 100)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 1),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC1'),
      SYSDATE-7, -65)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 1),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC1'),
      SYSDATE-5, 40)  );
```

```
--cpt Albaric CC2 signataire 2,D
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 2),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC1'),
      SYSDATE-5, -80)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 2),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC1'),
      SYSDATE-3, -50)  );

--cpt Bidal un signataire 3,D
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 2),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC6'),
       SYSDATE-7, 30)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 2),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC6'),
      SYSDATE-7, -15)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 3),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC6'),
      SYSDATE-5, -20)  );

--cpt Miranda 3 signataire 2-DRX, 1-D, 5-D
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 3),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC7'),
       SYSDATE-7, 300)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 3),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC7'),
      SYSDATE-6, -105)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 2),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC7'),
      SYSDATE-5, -20)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 1),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC7'),
      SYSDATE-4, -10)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 5),
```

```
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC7'),
      SYSDATE-5, -60)  );

--cpt Payrissat (4),  1 signataire sur CC3 1-D
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 4),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC4'),
       SYSDATE-2, 10)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 4),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC4'),
       SYSDATE-2, -70)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 4),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC5'),
       SYSDATE-5, 300)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 4),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC3'),
       SYSDATE-1, -50)  );
INSERT INTO mouvement VALUES
  (mouvement_type(
      (SELECT REF(cli) FROM Client cli     WHERE cli.num     = 1),
      (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC3'),
       SYSDATE-2, -70)  );
```

La régénération des données du schéma devra inclure le script suivant avant de lancer toutes les créations.

```
DELETE FROM Mouvement;
DELETE FROM CptEpargne;
DELETE FROM CptCourant;
DELETE FROM Client;
```

Manipulation de la base

a) Ajout du client `99`, `Paturel`, `Port-Royal – Paris`, téléphone domicile `04-47-56-98-16`, en initialisant le *varray* à un seul élément.

```
INSERT INTO Client VALUES
  (client_type(99, 'Paturel', 'Port Royal  - Paris',
            telephone_vry_type(telephone_elt_vry_type('04-47-56-98-16')) ));
```

b) Ajout du compte courant `CC99`, de solde 0 sans signataire et opération CB, à la date du jour (`SYSDATE`) associé au client .

```
INSERT INTO CptCourant VALUES
      (cptCourant_type('CC99', 0, SYSDATE,
```

```
        (SELECT REF(cli) FROM Client cli WHERE cli.num = 1), 0,
        signataire_nt_type())
      );
```

c) Associer le compte courant au client `99`.

```
UPDATE CptCourant cou
      SET cou.ref_Client = (SELECT REF(cli) FROM Client cli WHERE cli.num = 99)
      WHERE cou.nCompte  = 'CC99';
```

d) Insertion d'un mouvement de 50€ sur ce compte par ce client, en ne renseignant pas la date d'opération. Notons ici l'absence du constructeur car toutes les colonnes ne sont pas renseignées. Par défaut la date d'opération sera antérieure de deux jours à celle du jour.

```
INSERT INTO mouvement(ref_Client, ref_CptCourant, montant)
      VALUES
      ((SELECT REF(cli) FROM Client cli     WHERE cli.num     = 99),
       (SELECT REF(cou) FROM CptCourant cou WHERE cou.nCompte = 'CC99'), 50 );
```

Vérifions:

```
SQL> SELECT m.ref_Client.num, m.ref_Client.nom, m.dateOp, SYSDATE, m.montant
            FROM Mouvement m
            WHERE m.ref_CptCourant.nCompte = 'CC99';

REF_CLIENT.NUM REF_CLIENT.NOM                 DATEOP   SYSDATE     MONTANT
-------------- ------------------------------ -------- -------- ----------
            99 Paturel                        31/03/03 02/04/03         50
```

e) Procédure cataloguée `change_Portable(paramcli IN NUMBER, paramtel IN VARCHAR2)` qui affecte à un client donné (premier paramètre) un nouveau numéro de portable (second paramètre). Le *varray* est étendu ou pas.

```
CREATE OR REPLACE PROCEDURE change_Portable
            (paramcli IN NUMBER, paramtel IN VARCHAR2) IS
   tableau_tel  telephone_vry_type;
   indice NUMBER;
BEGIN
      SELECT telephone_vry  INTO tableau_tel
            FROM Client WHERE num = paramcli  FOR UPDATE;
      IF NOT tableau_tel.EXISTS(3) THEN
            indice := 1;
            WHILE (indice  <= tableau_tel.LIMIT) LOOP
            IF NOT (tableau_tel.EXISTS(indice)) THEN
                  tableau_tel.EXTEND;
                  tableau_tel(indice) := telephone_elt_vry_type(NULL);
            END IF;
            indice  := indice  + 1;
      END LOOP;
      END IF;
      tableau_tel(3).numTel := paramtel;
      UPDATE Client SET  telephone_vry = tableau_tel WHERE num = paramcli;
      EXCEPTION
       WHEN NO_DATA_FOUND THEN
```

```
            DBMS_OUTPUT.PUT_LINE('Client inexistant');
END;
/
```

Testons cette procédure en interrogeant avant et après exécution la table concernée.

```
SQL> SELECT * FROM Client WHERE num = 99;

       NUM NOM                            ADRESSE
---------- ------------------------------ ------------------------------
TELEPHONE_VRY(NUMTEL)
--------------------------------------------------------------------------------
        99 Paturel                        Port Royal  - Paris
TELEPHONE_VRY_TYPE(TELEPHONE_ELT_VRY_TYPE('04-47-56-98-16'))

SQL> EXECUTE change_Portable(99, '06-07-08-09-10');
Procédure PL/SQL terminée avec succès.

SQL> SELECT * FROM Client WHERE num = 99;

       NUM NOM                            ADRESSE
---------- ------------------------------ ------------------------------
TELEPHONE_VRY(NUMTEL)
--------------------------------------------------------------------------------
        99 Paturel                        Port Royal  - Paris
TELEPHONE_VRY_TYPE(TELEPHONE_ELT_VRY_TYPE('04-47-56-98-16'),
TELEPHONE_ELT_VRY_TYPE(NULL), TELEPHONE_ELT_VRY_TYPE('06-07-08-09-10'))
```

f) Transaction dans la procédure cataloguée `drop_Client(paramcli IN NUMBER)` qui détruit un client, ses comptes et les mouvements sur ces derniers.

```
CREATE OR REPLACE PROCEDURE drop_Client(paramcli IN NUMBER) IS
BEGIN
      DELETE FROM CptEpargne cep WHERE cep.ref_Client.num = paramcli;
      DELETE FROM Mouvement mou
            WHERE mou.ref_CptCourant.nCompte IN
            (SELECT nCompte FROM CptCourant cou
             WHERE cou.ref_Client.num = paramcli);
      DELETE FROM CptCourant cou WHERE cou.ref_Client.num = paramcli;
      DELETE FROM Client WHERE num = paramcli;
      COMMIT;
END;
/
```

Exécutons cette procédure pour le client `99`.

Avant exécution, listons le contenu des objets à détuire.

```
SQL> SELECT * FROM Client WHERE num = 99;

       NUM NOM                            ADRESSE
---------- ------------------------------ ------------------------------
TELEPHONE_VRY(NUMTEL)
--------------------------------------------------------------------------------
        99 Paturel                        Port Royal  - Paris
```

```
TELEPHONE_VRY_TYPE(TELEPHONE_ELT_VRY_TYPE('04-47-56-98-16'),
TELEPHONE_ELT_VRY_T
YPE(NULL), TELEPHONE_ELT_VRY_TYPE('06-07-08-09-10'))

SQL> SELECT nCompte FROM CptCourant cou WHERE cou.ref_Client.num = 99;

NCOMP
-----
CC99

SQL> SELECT nCompte FROM CptEpargne cep WHERE cep.ref_Client.num = 99;

aucune ligne sélectionnée

SQL> SELECT m.ref_Client.num, m.ref_Client.nom,
            m.ref_Cptcourant.nCompte "CPTE", m.dateOp, SYSDATE, m.montant
      FROM Mouvement m
      WHERE m.ref_CptCourant.nCompte IN
      (SELECT nCompte FROM CptCourant cou
       WHERE cou.ref_Client.num = 99);

REF_CLIENT.NUM REF_CLIENT.NOM                CPTE    DATEOP  SYSDATE   MONTANT
-------------- ----------------------------- ----- -------- -------- ---------
            99 Paturel                       CC99  31/03/03 02/04/03        50
```

Après exécution :

```
SQL> EXECUTE  drop_Client(99);
Procédure PL/SQL terminée avec succès.

SQL> SELECT * FROM Client WHERE num = 99;
aucune ligne sélectionnée

SQL> SELECT nCompte FROM CptCourant cou WHERE cou.ref_Client.num = 99;
aucune ligne sélectionnée

SQL> SELECT nCompte FROM CptEpargne cep WHERE cep.ref_Client.num = 99;
aucune ligne sélectionnée

SQL> SELECT m.ref_Client.num, m.ref_Client.nom,
            m.ref_Cptcourant.nCompte "CPTE", m.dateOp, SYSDATE, m.montant
        FROM Mouvement m
        WHERE m.ref_CptCourant.nCompte IN
                  (SELECT nCompte FROM CptCourant cou
                   WHERE cou.ref_Client.num = 99);
aucune ligne sélectionnée
```

Requêtes SELECT

Simples

1. Liste des clients sans leur numéro de téléphone.

```
SQL> SELECT c.num, c.nom, c.adresse FROM Client c;

       NUM NOM                            ADRESSE
---------- ------------------------------ ------------------------------
         1 Albaric                        Pont Vieux  - Vielle Toulouse
         2 Bidal                          Port Royal - Paris
         3 Miranda                        Antipolis - Nice
         4 Payrissat                      Salas - Ramonville St Agne
         5 Vielle                         INRA - Auzeville Tolosane
```

2. Liste des comptes courant (numéro, solde, date d'ouverture, nombre d'opération CB).

```
SQL> SELECT  nCompte, solde, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", nbOpCB
             FROM CptCourant;

NCOMP       SOLDE DATEOUV        NBOPCB
----- ---------- ---------- ----------
CC1          4030 01/02/2001        509
CC2          3000 15/02/2002          0
CC3           460 13/05/2000        678
CC4           730 17/09/2002          0
CC5            15 10/12/1998       1390
CC6            55 16/01/1965       2400
CC7          6700 04/03/1976       5600
```

3. Liste des comptes épargne (numéro, solde, date d'ouverture, taux d'intérêt).

```
SQL> SELECT  nCompte, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", solde, txInt
             FROM CptEpargne;

NCOMP DATEOUV          SOLDE      TXINT
----- ---------- ---------- ----------
CE1   05/02/1965        600        2,7
CE2   04/12/1998       4500        2,9
CE3   05/03/2000        500        2,9
CE4   05/02/2001        500        2,4
CE5   13/05/1995        500        3,4
```

Références

4. Liste des comptes courant du client de numéro 4 (jointure implicite dans le `WHERE`).

```
SQL> COLUMN nCompte FORMAT A30 HEADING 'Comptes courants du client 4'

SQL> SELECT  cou.nCompte, cou.solde, TO_CHAR(cou.dateOuv,'DD/MM/YYYY' "DATEOUV"
             FROM CptCourant cou
```

```
                WHERE cou.ref_Client.num = 4;

Comptes courants du client 4        SOLDE DATEOUV
------------------------------ ---------- ----------
CC3                                   460 13/05/2000
CC4                                   730 17/09/2002
CC5                                    15 10/12/1998
```

5. Même requête pour tous ses comptes (courant et épargne).

```
SQL> SELECT  cou.nCompte, cou.solde,TO_CHAR(cou.dateOuv,'DD/MM/YYYY') "DATEOUV"
             FROM CptCourant cou
             WHERE cou.ref_Client.num = 4
    UNION
    SELECT  cep.nCompte, cep.solde, TO_CHAR(cep.dateOuv,'DD/MM/YYYY')
            FROM CptEpargne cep
            WHERE cep.ref_Client.num = 4;

Comptes courants du client 4        SOLDE DATEOUV
------------------------------ ---------- ----------
CC3                                   460 13/05/2000
CC4                                   730 17/09/2002
CC5                                    15 10/12/1998
CE3                                   500 05/03/2000
CE4                                   500 05/02/2001
CE5                                   500 13/05/1995
```

6. Liste des clients et numéro des comptes courants de solde inférieur à 400€ donné (jointures implicites dans le `SELECT`).

```
SQL> COL  ref_Client.nom FORMAT A15 HEADING 'NOM'

SQL> SELECT cou.ref_Client.num, cou.ref_Client.nom, cou.ref_Client.adresse,
            cou.nCompte, cou.solde
            FROM CptCourant cou
            WHERE cou.solde < 400;

REF_CLIENT.NUM NOM             REF_CLIENT.ADRESSE              NCOMP      SOLDE
-------------- --------------- ------------------------------ ----- ----------
             4 Payrissat       Salas - Ramonville St Agne     CC5            15
             2 Bidal           Port Royal - Paris             CC6            55
```

7. Numéro, nom et adresse des clients titulaires d'au moins un compte épargne.

```
SQL> SELECT DISTINCT(cep.ref_Client.num), cep.ref_Client.nom,
                     cep.ref_Client.adresse
            FROM CptEpargne cep;

REF_CLIENT.NUM REF_CLIENT.NOM                REF_CLIENT.ADRESSE
-------------- ------------------------------ ------------------------------
             2 Bidal                          Port Royal - Paris
             3 Miranda                        Antipolis - Nice
             4 Payrissat                      Salas - Ramonville St Agne
```

8. Numéro, nom et adresse des clients titulaires d'un seul compte épargne.

```
SQL> SELECT cep.ref_Client.num, cep.ref_Client.nom, cep.ref_Client.adresse
           FROM CptEpargne cep
      GROUP BY (cep.ref_Client.num, cep.ref_Client.nom, cep.ref_Client.adresse)
      HAVING COUNT(*)=1;

REF_CLIENT.NUM REF_CLIENT.NOM                 REF_CLIENT.ADRESSE
-------------- ------------------------------ ------------------------------
             3 Miranda                        Antipolis - Nice
```

9. Pour chaque client (numéro, nom), afficher le nombre de compte épargne qu'il possède.

```
SQL> COLUMN  nbr HEADING 'Nombre de comptes épargne'
SQL> SELECT  cep.ref_Client.num, cep.ref_Client.nom, COUNT(*) nbr
             FROM  CptEpargne cep
             GROUP BY (cep.ref_Client.num, cep.ref_Client.nom);

REF_CLIENT.NUM REF_CLIENT.NOM                 Nombre de comptes épargne
-------------- ------------------------------ -------------------------
             2 Bidal                                                  2
             3 Miranda                                                1
             4 Payrissat                                              3
```

10. Même requête pour afficher aussi les clients n'ayant pas de compte épargne.

```
SQL> COLUMN  nbr HEADING 'Nombre de comptes épargne'
SQL> SELECT  cep.ref_Client.num, cep.ref_Client.nom, COUNT(*) nbr
             FROM  CptEpargne cep
             GROUP BY (cep.ref_Client.num, cep.ref_Client.nom)
     UNION
     SELECT  num, nom, 0
             FROM client
              WHERE num NOT IN
                  (SELECT DISTINCT cep.ref_Client.num FROM CptEpargne cep);

REF_CLIENT.NUM REF_CLIENT.NOM                 Nombre de comptes épargne
-------------- ------------------------------ -------------------------
             1 Albaric                                                0
             2 Bidal                                                  2
             3 Miranda                                                1
             4 Payrissat                                              3
             5 Vielle                                                 0
```

11. Numéro, nom et nombre de compte épargne du titulaire ayant le plus de comptes épargne.

```
SQL> SELECT cep.ref_Client.num, cep.ref_Client.nom, COUNT(*) "Nombre"
           FROM CptEpargne cep
           GROUP BY (cep.ref_Client.num, cep.ref_Client.nom)
           HAVING COUNT(*) =
            (SELECT MAX(COUNT(*)) FROM CptEpargne cep2
             GROUP BY (cep2.ref_Client.num, cep2.ref_Client.nom));

REF_CLIENT.NUM REF_CLIENT.NOM                     Nombre
-------------- ------------------------------ ----------
             4 Payrissat                               3
```

12. Liste des clients (numéro et nom) qui ont fait des opérations (date et montant) sur le compte courant de numéro CC7.

```
SQL> SELECT m.ref_Client.num, m.ref_Client.nom, m.dateOp, m.montant
            FROM Mouvement m
            WHERE m.ref_CptCourant.nCompte = 'CC7';

REF_CLIENT.NUM REF_CLIENT.NOM                 DATEOP      MONTANT
-------------- ------------------------------ -------- ----------
             3 Miranda                        24/03/03        300
             3 Miranda                        25/03/03       -105
             2 Bidal                          26/03/03        -20
             1 Albaric                        27/03/03        -10
             5 Vielle                         26/03/03        -60
```

13. Liste des opérations sur le compte courant CC7 (numéro de compte, date et montant) qui ont été faites par un client propriétaire du compte modifié (Jointures dans le `SELECT` et dans le `WHERE`).

```
SQL> SELECT m.ref_Client.num, m.ref_Client.nom, m.dateOp, m.montant
            FROM Mouvement m
            WHERE m.ref_CptCourant.nCompte        = 'CC7'
            AND   m.ref_CptCourant.ref_Client.num = m.ref_Client.num;

REF_CLIENT.NUM REF_CLIENT.NOM                 DATEOP      MONTANT
-------------- ------------------------------ -------- ----------
             3 Miranda                        24/03/03        300
             3 Miranda                        25/03/03       -105
```

14. Même requête pour les opérations qui ont été faites par les clients non propriétaire du compte modifié.

```
SQL> SELECT m.ref_Client.num, m.ref_Client.nom, m.dateOp, m.montant
            FROM Mouvement m
            WHERE m.ref_CptCourant.nCompte          = 'CC7'
            AND NOT(m.ref_CptCourant.ref_Client.num = m.ref_Client.num);

REF_CLIENT.NUM REF_CLIENT.NOM                 DATEOP      MONTANT
-------------- ------------------------------ -------- ----------
             2 Bidal                          26/03/03        -20
             1 Albaric                        27/03/03        -10
             5 Vielle                         26/03/03        -60
```

Collections

15. Nombre de téléphones du client 1 (première écriture de l'opérateur `TABLE`). Attention `COUNT(*)` compte deux numéros de téléphone (la taille du *varray* du client 1), même si seul un élément est non nul.

```
SQL> SELECT COUNT(numTel) "Nombre Téléphones, client 1"
            FROM TABLE(SELECT telephone_vry FROM Client WHERE num = 1);

Nombre Téléphones, client 1
---------------------------
```

```
                          1
```

```
SQL> SELECT COUNT(*) "Taille telephone_vry, client 1"
            FROM TABLE(SELECT telephone_vry FROM Client WHERE num = 1);

Taille telephone_vry, client 1
------------------------------
                             2
```

16. Même requête pour chaque client (deuxième écriture de l'opérateur `TABLE`).

```
SQL> SELECT  c.num, COUNT(avry.numTel) "Nombre Téléphones"
             FROM Client c, TABLE(c.telephone_vry) avry
             GROUP BY  c.num;

       NUM Nombre Téléphones
---------- -----------------
         1                 1
         2                 1
         3                 2
         4                 2
         5                 3
```

17. Nombre de signataire du compte courant CC7 (première écriture de l'opérateur `TABLE`).

```
SQL> SELECT COUNT(*) "Nombre de signataires CC7"
            FROM TABLE(SELECT signataire_nt
                       FROM CptCourant WHERE nCompte = 'CC7');

Nombre de signataires CC7
-------------------------
                        5
```

18. Même requête pour chaque compte courant (deuxième écriture de l'opérateur `TABLE`).

```
SQL> SELECT cou.nCompte, COUNT(DISTINCT(ant.num)) "Nombre de signataires"
            FROM CptCourant cou, TABLE(cou.signataire_nt) ant
            GROUP BY cou.nCompte;

NCOMP Nombre de signataires
----- ---------------------
CC2                       2
CC3                       2
CC6                       1
CC7                       3
```

19. Numéros de téléphone du client numéro 1 (première écriture de l'opérateur `TABLE`).

```
SQL> SELECT avry.numTel
            FROM TABLE(SELECT telephone_vry FROM Client WHERE num = 1) avry;

NUMTEL
--------------
05-61-75-68-39
```

20. Même requête pour chaque client (deuxième écriture de l'opérateur `TABLE`).

```
SQL> SELECT c.num, avry.numTel
            FROM Client c, TABLE(c.telephone_vry) avry
            ORDER BY 1,2;

       NUM NUMTEL
---------- --------------
         1 05-61-75-68-39
         1
         2 06-76-85-14-89
         2
         2
         3 04-35-60-77-89
         3 06-81-94-44-31
         3
         4 05-61-75-98-44
         4 06-46-45-72-30
         4
         5 05-61-73-12-74
         5 05-62-74-75-63
         5 06-65-41-83-35
```

21. Numéro et droit des signataires du compte courant CC7 (première écriture de l'opérateur TABLE).

```
SQL> COL droit FORMAT A5
SQL> SELECT ant.num "Signataires CC7", ant.droit
            FROM TABLE(SELECT signataire_nt
                       FROM CptCourant WHERE nCompte = 'CC7') ant;

Signataires CC7 DROIT
--------------- -----
              2 D
              2 R
              2 X
              1 D
              5 D
```

22. Même requête pour pour tous les comptes courant (deuxième écriture de l'opérateur TABLE).

```
SQL> SELECT cou.nCompte, ant.num "SIGNATAIRE", ant.droit
            FROM CptCourant cou, TABLE(cou.signataire_nt) ant
            ORDER BY 1,2;

NCOMP SIGNATAIRE DROIT
----- ---------- -----
CC2            2 D
CC2            2 R
CC2            3 R
CC3            1 D
CC3            5 D
CC6            3 D
CC7            1 D
CC7            2 D
CC7            2 R
```

```
CC7            2 X
CC7            5 D
```

23. Même requête avec un *nested cursor*.

```
SQL> SELECT cou.nCompte, CURSOR(SELECT ant.num "SIGNATAIRE", ant.droit
                                FROM TABLE(signataire_nt) ant )
            FROM CptCourant cou;

NCOMP CURSOR(SELECTANT.NUM
----- --------------------
CC1   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
aucune ligne sélectionnée

CC2   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
SIGNATAIRE DROIT
---------- -----
         2 D
         2 R
         3 R

CC3   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
SIGNATAIRE DROIT
---------- -----
         1 D
         5 D

CC4   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
aucune ligne sélectionnée

CC5   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
aucune ligne sélectionnée

CC6   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
SIGNATAIRE DROIT
---------- -----
         3 D

CC7   CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
SIGNATAIRE DROIT
---------- -----
         2 D
         2 R
         2 X
         1 D
         5 D
```

24. Numéro et adresse des signataires du compte courant CC7 (première écriture de l'opérateur `TABLE`). Nous faisons une jointure entre la *nested table* et la table `Client`.

```
SQL> SELECT ant.num "Signataires CC7", cli.adresse "ADRESSE", ant.droit
            FROM TABLE(SELECT signataire_nt
                       FROM CptCourant WHERE nCompte = 'CC7') ant, Client cli
            WHERE cli.num = ant.num;

Signataires CC7 ADRESSE                        DROIT
--------------- ------------------------------ -----
              2 Port Royal - Paris             D
              2 Port Royal - Paris             R
              2 Port Royal - Paris             X
              1 Pont Vieux  - Vielle Toulouse  D
              5 INRA - Auzeville Tolosane      D
```

Bloc PL/SQL

25. Numéro de téléphone du travail du client 3.

```
DECLARE
  nouv_tel telephone_vry_type;
BEGIN
  SELECT telephone_vry INTO nouv_tel FROM Client WHERE num = 3;
  IF (nouv_tel(2).numTel IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('Deuxième numéro non renseigné (travail)');
ELSE
    DBMS_OUTPUT.PUT_LINE('Le numéro du travail est du client 3 est : ' ||
                          nouv_tel(2).numTel);
  END IF;
END;
/
```

Résultat :

```
Le numéro du travail est du client 3 est : 04-35-60-77-89

Procédure PL/SQL terminée avec succès.
```

26. Bloc précédent modifé en utilisant un curseur pour afficher le numéro du téléphone du travail de tous les clients titulaire d'un compte courant.

```
DECLARE
  CURSOR CLientsCourant IS SELECT DISTINCT(cou.ref_Client.num) num,
      cou.ref_Client.nom nom
      FROM CptCourant cou ORDER BY 1;
  tab_Telephone telephone_vry_type;
BEGIN
  FOR UnClientCourant IN CLientsCourant LOOP
     DBMS_OUTPUT.PUT_LINE('Client : ' || TO_CHAR(UnClientCourant.num) || ' '
                   || UnClientCourant.nom);
    SELECT telephone_vry INTO tab_Telephone
      FROM Client cli WHERE cli.num = UnClientCourant.num;
    IF (tab_Telephone(2).numTel IS NULL) THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Deuxième numéro non renseigné (travail)');
      ELSE
        DBMS_OUTPUT.PUT_LINE('Le numero du travail est : ' ||
                             tab_Telephone(2).numTel);
      END IF;
    END LOOP;
END;
/
```

Résultat :

```
Client : 1 Albaric
Deuxième numéro non renseigné (travail)
Client : 2 Bidal
Deuxième numéro non renseigné (travail)
Client : 3 Miranda
Le numero du travail est : 04-35-60-77-89
Client : 4 Payrissat
Deuxième numéro non renseigné (travail)
Procédure PL/SQL terminée avec succès.
```

Méthodes

Fonctions

27. Méthode `nbCepargne` qui renvoie le nombre de compte épargne d'un client donné.

```
ALTER TYPE client_type
      ADD MEMBER FUNCTION nbCepargne RETURN NUMBER CASCADE;

CREATE OR REPLACE TYPE BODY client_type
  AS MEMBER FUNCTION nbCepargne RETURN NUMBER IS
  nb_compte NUMBER;
  BEGIN
    SELECT COUNT(*) INTO nb_compte  FROM CptCourant cep
      WHERE cep.ref_Client.num = SELF.num;
    RETURN nb_compte;
  END nbCepargne;
END;
/
```

Testons cette méthode dans une requête et dans un bloc PL/SQL pour le client numéro 4.

```
SQL> SELECT cli.num, cli.nbCepargne() FROM Client cli;

       NUM CLI.NBCEPARGNE()
---------- ----------------
         1                2
         2                1
         3                1
         4                3
```

```
         5               0

DECLARE
  un_Client client_type;
BEGIN
  SELECT VALUE(cli) INTO un_Client
      FROM Client cli WHERE cli.num = 4;
  DBMS_OUTPUT.PUT_LINE('Le client numéro 4 possède '|| un_Client .nbCepargne()
      || ' compte(s) épargne(s)');
END;
/
Le client numéro 4 possède 3 compte(s) épargne(s)
Procédure PL/SQL terminée avec succès.
```

28. Méthode `nbSignataire` qui renvoie le nombre de signataire d'un compte courant donné.

```
ALTER TYPE cptCourant_type
      ADD MEMBER FUNCTION nbSignataire RETURN NUMBER CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type
  AS MEMBER FUNCTION nbSignataire RETURN NUMBER Is
  nb_sign NUMBER;
  BEGIN
    SELECT COUNT(DISTINCT(ant.num)) INTO nb_sign
      FROM TABLE (SELECT cou.signataire_nt
            FROM cptCourant cou WHERE cou.nCompte = SELF.nCompte) ant;
    RETURN nb_sign;
  END nbSignataire;
END;
/
```

Testons cette méthode dans une requête et dans un bloc PL/SQL pour le compte CC7.

```
SQL> SELECT  cou.nCompte, cou.nbSignataire() FROM cptCourant cou;

Comptes courants du client 4   COU.NBSIGNATAIRE()
------------------------------ ------------------
CC1                                             0
CC2                                             2
CC3                                             2
CC4                                             0
CC5                                             0
CC6                                             1
CC7                                             3

DECLARE
  unCptCourant cptCourant_type;
  resultat NUMBER;
BEGIN
  SELECT VALUE(cou) INTO unCptCourant
      FROM cptCourant cou WHERE cou.nCompte = 'CC7';
  resultat := unCptCourant.nbSignataire();
  DBMS_OUTPUT.PUT_LINE('Le compte courant CC7 a '||
```

```
                        resultat || ' signataire(s)');
END;
/
Le compte courant CC7 a 3 signataire(s)
Procédure PL/SQL terminée avec succès.
```

29. Méthode `nbSignataire(droitparam IN CHAR)` qui ne tient compte que les signataires de droit passé en paramètre.

```
ALTER TYPE cptCourant_type
      ADD MEMBER FUNCTION nbSignataire(droitparam IN CHAR)
          RETURN NUMBER CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type AS
 MEMBER FUNCTION nbSignataire RETURN NUMBER Is
  ...vue au dessus
  END nbSignataire;
 MEMBER FUNCTION nbSignataire(droitparam IN CHAR) RETURN NUMBER Is
  nb_sign NUMBER;
  BEGIN
    SELECT COUNT(DISTINCT(ant.num)) INTO nb_sign
      FROM TABLE (SELECT cou.signataire_nt
            FROM cptCourant cou WHERE cou.nCompte = SELF.nCompte) ant
      WHERE ant.droit = droitparam;
    RETURN nb_sign;
  END nbSignataire;
END;
/
```

Testons cette méthode dans une requête pour chaque droit de chaque compte.

```
SQL> SELECT cou.nCompte, cou.nbSignataire('X'),
            cou.nbSignataire('R'), cou.nbSignataire('D')
            FROM cptCourant cou;

NCOMP COU.NBSIGNATAIRE('X') COU.NBSIGNATAIRE('R') COU.NBSIGNATAIRE('D')
----- -------------------- -------------------- --------------------
CC1                       0                    0                    0
CC2                       0                    2                    1
CC3                       0                    0                    2
CC4                       0                    0                    0
CC5                       0                    0                    0
CC6                       0                    0                    1
CC7                       1                    1                    3
```

30. Méthode `calculeInteret` qui renvoie la somme augmentée des intérêts d'un compte épargne donné.

```
ALTER TYPE cptEpargne_type
      ADD MEMBER FUNCTION calculeInterets RETURN NUMBER CASCADE;

CREATE OR REPLACE TYPE BODY cptEpargne_type AS
  MEMBER FUNCTION calculeInterets RETURN NUMBER IS
```

```
    SommeAvecInterets cptEpargne.solde%TYPE;
  BEGIN
    SELECT solde*(1+ txInt/100) INTO SommeAvecInterets
      FROM cptEpargne
      WHERE nCompte = SELF.nCompte;
    RETURN SommeAvecInterets;
  END calculeInterets;
END;
/
```

Testons cette méthode pour le compte épargne CE5. Avant d'exécuter cette méthode, consultons la table.

```
SQL> SELECT ncompte,solde,txInt FROM cptEpargne;

NCOMP      SOLDE      TXINT
----- ---------- ----------
CE1          600        2,7
CE2         4500        2,9
CE3          500        2,9
CE4          500        2,4
CE5          500        3,4
CE6         3300        3,3

DECLARE
  unCompteEpargne cptEpargne_type;
BEGIN
  SELECT VALUE(cep) INTO unCompteEpargne
      FROM cptEpargne cep
      WHERE cep.nCompte = 'CE5';
  DBMS_OUTPUT.PUT_LINE('Epargne cumulée de CE5 : ' ||
                       unCompteEpargne.calculeInterets());
EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('CE5, compte épargne inexistant');
END;
/
Epargne cumulée de CE5 : 517
Procédure PL/SQL terminée avec succès.
```

31. Méthode booléenne `estTitulaire(cli IN NUMBER)` qui renvoie `TRUE` si le client de numéro passé en paramètre est titulaire du compte courant donné.

```
ALTER TYPE cptCourant_type
      ADD MEMBER FUNCTION estTitulaire(cli IN NUMBER) RETURN BOOLEAN CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type AS
…
  MEMBER FUNCTION estTitulaire(cli IN NUMBER) RETURN BOOLEAN IS
      resultat NUMBER := 0;
   BEGIN
      SELECT COUNT(*) INTO resultat FROM cptCourant cou
            WHERE cou.ref_Client.num = cli
            AND cou.nCompte          = SELF.nCompte;
```

```
      IF (resultat = 1) THEN
            RETURN TRUE;
      ELSE
            RETURN FALSE;
      END IF;
 END estTitulaire;
END;
/
```

Testons cette méthode dans un bloc en cherchant le titulaire du compte CC4.

```
DECLARE
  unCptCourant cptCourant_type;
BEGIN
  SELECT VALUE(cou) INTO unCptCourant
      FROM cptCourant cou WHERE cou.nCompte = 'CC4';
  IF ( unCptCourant.estTitulaire(4) ) THEN
      DBMS_OUTPUT.PUT_LINE('Le client 4 est titulaire du compte CC4');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Le client 4 n''est pas titulaire du compte CC4');
  END IF;
  IF ( unCptCourant.estTitulaire(5) ) THEN
      DBMS_OUTPUT.PUT_LINE('Le client 5 est titulaire du compte CC4');
  ELSE
      DBMS_OUTPUT.PUT_LINE('Le client 5 n''est pas titulaire du compte CC4');

  END IF;
END;
/
```

```
Le client 4 est titulaire du compte CC4
Le client 5 n'est pas titulaire du compte CC4

Procédure PL/SQL terminée avec succès.
```

32. Méthode booléenne `estSignataire(cli IN NUMBER, droitparam IN CHAR)` qui renvoie `TRUE` si le client de numéro passé en paramètre est signataire du droit passé en paramètre pour un compte courant donné.

```
ALTER TYPE cptCourant_type
      ADD MEMBER FUNCTION estSignataire(cli IN NUMBER, droitparam IN CHAR)
            RETURN BOOLEAN CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type AS
…
 MEMBER FUNCTION estSignataire(cli IN NUMBER, droitparam IN CHAR)
                  RETURN BOOLEAN IS
      resultat NUMBER := 0;
   BEGIN
      SELECT COUNT(ant.num) INTO resultat
            FROM TABLE (SELECT cou.signataire_nt
                  FROM cptCourant cou WHERE cou.nCompte = SELF.nCompte) ant
      WHERE ant.droit = droitparam
      AND   ant.num   = cli;
```

```
      IF (resultat = 0) THEN
            RETURN FALSE;
      ELSE
            RETURN TRUE;
      END IF;
 END estSignataire;
END;
/
```

Testons cette méthode dans un bloc en cherchant des signataires au droit D du compte CC7.

```
DECLARE
  unCptCourant cptCourant_type;
BEGIN
  SELECT VALUE(cou) INTO unCptCourant
      FROM cptCourant cou WHERE cou.nCompte = 'CC7';
   IF ( unCptCourant.estSignataire(5,'D') ) THEN
      DBMS_OUTPUT.PUT_LINE('Le client 5 est signataire (D)
                            pour le compte CC7');
  ELSE
      DBMS_OUTPUT.PUT_LINE('Le client 5 n''est pas signataire (D)
                        pour le compte CC7');
  END IF;
  IF ( unCptCourant.estSignataire(3,'D') ) THEN
      DBMS_OUTPUT.PUT_LINE('Le client 3 est signataire (D)
                         pour le compte CC7');
  ELSE
      DBMS_OUTPUT.PUT_LINE('Le client 3 n''est pas signataire (D)
                         pour le compte CC7');
  END IF;
END;
/

Le client 5 est signataire (D) pour le compte CC7
Le client 3 n'est pas signataire (D) pour le compte CC7

Procédure PL/SQL terminée avec succès.
```

Procédures

33. Méthode `supprimeTel(indice IN NUMBER)` qui supprime pour un client, le téléphone d'indice passé en paramètre. Si aucun téléphone n'est présent à cet indice, la méthode le signale.

```
ALTER TYPE client_type
      ADD MEMBER PROCEDURE supprimeTel(indice IN NUMBER) CASCADE;

CREATE OR REPLACE TYPE BODY client_type
  …
  MEMBER PROCEDURE supprimeTel(indice In NUMBER) IS
    liste_tel telephone_vry_type;
  BEGIN
      SELECT telephone_vry INTO liste_tel
```

```
            FROM Client WHERE num = SELF.num FOR UPDATE;
      IF (liste_tel(indice).numTel IS NULL) THEN
        DBMS_OUTPUT.PUT_LINE('Pas de téléphone répertorié l''indice '
                             || indice);
      ELSE
        liste_tel(indice) := telephone_elt_vry_type(NULL);
        DBMS_OUTPUT.PUT_LINE('Téléphone ( ' || indice || ' ) supprimé');
      END IF;
      UPDATE Client SET telephone_vry = liste_tel WHERE num = SELF.num;
  END supprimeTel;
END;
/
```

Testons cette méthode en supprimant les deuxième et troisième numéro de téléphone du client 4. Avant d'exécuter cette méthode, consultons la table. Nous la consulterons aussi près exécution de la méthode.

```
SQL> SELECT avry.numTel
            FROM TABLE(SELECT telephone_vry FROM Client WHERE num = 4) avry;

NUMTEL
--------------
05-61-75-98-44

06-46-45-72-30

DECLARE
  unClient  client_type;
BEGIN
  SELECT VALUE(cli) INTO unClient FROM client cli WHERE cli.num = 4;
  unClient.supprimeTel(2);
  unClient.supprimeTel(3);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Client inexistant');
END;
/
Pas de téléphone répertorié l'indice 2
Téléphone ( 3 ) supprimé
Procédure PL/SQL terminée avec succès.

SQL> SELECT avry.numTel
            FROM TABLE(SELECT telephone_vry FROM Client WHERE num = 4) avry;

NUMTEL
--------------
05-61-75-98-44

```

34. Méthode `supprimeSign(numcli IN NUMBER)` qui supprime pour un compte courant, le signataire de numéro passé en paramètre. Si aucun signataire n'est présent, la méthode signale l'anomalie.

```
ALTER TYPE cptCourant_type
      ADD MEMBER PROCEDURE supprimeSign (nclient IN NUMBER) CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type AS
  …
  MEMBER PROCEDURE supprimeSign(nclient In NUMBER) IS
    num_sign signataire_tabnt.num%TYPE;
  BEGIN
    SELECT DISTINCT(ant.num) INTO num_sign
      FROM TABLE(SELECT signataire_nt
            FROM cptCourant WHERE nCompte=SELF.nCompte) ant
      WHERE ant.num = nclient;
    DELETE FROM TABLE
      (SELECT signataire_nt FROM cptCourant WHERE nCompte=SELF.nCompte) ant
      WHERE ant.num = nclient;
    DBMS_OUTPUT.PUT_LINE('Suppression du signataire '|| nclient ||
                         ' effectuée pour le compte ' || SELF.nCompte);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Le signataire '|| nclient ||
                           ' n''existe pas pour le compte ' || SELF.nCompte);
  END supprimeSign;
END;
/
```

Testons cette méthode en supprimant pour le compte CC7 le signataire de numéro 2. Avant d'exécuter cette méthode, consultons la table. Nous la consulterons aussi près exécution de la méthode. Testons aussi un cas d'erreur en tentant de supprimer le signataire de numéro 6 pour ce compte qui ne l'inclut pas.

```
SQL> SELECT ant.num "Signataires CC7", ant.droit FROM
         TABLE(SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7') ant;

Signataires CC7 DROIT
--------------- -----
              2 D
              2 R
              2 X
              1 D
              5 D

DECLARE
 unCptCourant cptCourant_type;
BEGIN
  SELECT VALUE(cou) INTO unCptCourant
           FROM cptCourant cou WHERE cou.nCompte = 'CC7';
  unCptCourant.supprimeSign(2);
  unCptCourant.supprimeSign(6);
END;
/
Suppression du signataire 2 effectuée pour le compte CC7
Le signataire 6 n'existe pas pour le compte CC7
```

```
Procédure PL/SQL terminée avec succès.

SQL> SELECT ant.num "Signataires CC7", ant.droit FROM
        TABLE(SELECT signataire_nt FROM CptCourant WHERE nCompte = 'CC7') ant;

Signataires CC7 DROIT
--------------- -----
              1 D
              5 D
```

35. Méthode `afficheInterets` qui affiche l'épargne cumulée d'un client donné en utilisant la fonction `calculeInterets`.

```
ALTER TYPE client_type
      ADD MEMBER PROCEDURE afficheInterets CASCADE;

CREATE OR REPLACE TYPE BODY client_type
      …
  MEMBER PROCEDURE afficheInterets IS
    epargneTotale NUMBER := 0;
    CURSOR LesCptEpargnes IS SELECT VALUE(cep)
            FROM CptEpargne cep WHERE cep.ref_Client.num = SELF.num;
    unCptEpargne CptEpargne_type;
  BEGIN
    OPEN LesCptEpargnes;
    FETCH LesCptEpargnes INTO unCptEpargne;
    WHILE (LesCptEpargnes%FOUND) LOOP
      epargneTotale := epargneTotale + unCptEpargne.calculeInterets();
      DBMS_OUTPUT.PUT_LINE('Cumul du compte : ' || unCptEpargne.nCompte ||
                           ' intérêts : ' || unCptEpargne.calculeInterets());
        FETCH LesCptEpargnes INTO unCptEpargne;
    END LOOP;
    CLOSE LesCptEpargnes;
    DBMS_OUTPUT.PUT_LINE('L''épargne totale est de  : '||epargneTotale||' €');
  END afficheInterets;
END;
/
```

Testons cette méthode pour le client 4 qui possède trois comptes épargnes.

```
DECLARE
  unClient client_type;
BEGIN
  SELECT VALUE(cli) INTO unClient FROM Client cli WHERE cli.num = 4;
  unClient.afficheInterets();
END;
/
Cumul du compte : CE3 intérêts : 514,5
Cumul du compte : CE4 intérêts : 512
Cumul du compte : CE5 intérêts : 517
L'épargne totale est de  : 1543,5 €
```

```
Procédure PL/SQL terminée avec succès.
```

36. Méthode `STATIC fermeCompte` clôture un compte donné en affichant son solde avant de le détruire. La méthode vider la table des mouvements relatifs à ce compte si le compte est un compte courant.

```
ALTER TYPE compte_type
      ADD STATIC PROCEDURE fermeCompte (cpt IN VARCHAR2) CASCADE;

CREATE OR REPLACE TYPE BODY compte_type AS
 STATIC PROCEDURE fermeCompte (cpt IN VARCHAR2) IS
  nbMouvement     NUMBER;
  unCompteCourant CptCourant_type;
  unCptEpargne    CptEpargne_type;
BEGIN
   BEGIN
    SELECT VALUE(cou) INTO unCompteCourant
           FROM cptCourant cou WHERE nCompte = cpt;
    DBMS_OUTPUT.PUT_LINE('Le solde du compte '|| cpt ||
           ' est : ' || unCompteCourant.solde);
    SELECT COUNT(*) INTO nbMouvement
           FROM Mouvement m WHERE m.ref_CptCourant.nCompte = cpt;
    DELETE FROM Mouvement m  WHERE m.ref_CptCourant.nCompte = cpt;
    DELETE FROM cptCourant   WHERE nCompte = cpt;
    DBMS_OUTPUT.PUT_LINE('Compte courant détruit, ' || nbMouvement ||
            ' mouvements supprimé(s)');
    COMMIT;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        BEGIN
         SELECT VALUE(cep) INTO unCptEpargne
            FROM CptEpargne cep WHERE nCompte = cpt;
         DBMS_OUTPUT.PUT_LINE('Le solde du compte '|| cpt || ' est : ' ||
             unCptEpargne.solde);
         DELETE FROM CptEpargne WHERE nCompte = cpt;
         DBMS_OUTPUT.PUT_LINE('Compte épargne supprimé.');
         COMMIT;
         EXCEPTION
         WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Le compte '|| cpt ||' n''existe pas!');
         WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Gros Problème...');
          END;
      WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Gros Problème...');
      END;
  END fermeCompte;
END;
/
```

Testons cette méthode en supprimant le comptes épargne CE4, le compte courant CC1 et testons les cas d'erreurs.

État de la base avant :

```
SQL> SELECT nCompte, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", solde, txInt
            FROM CptEpargne;

NCOMP DATEOUV         SOLDE      TXINT
----- ---------- ---------- ----------
CE1   05/02/1965        600        2,7
CE2   04/12/1998       4500        2,9
CE3   05/03/2000        500        2,9
CE4   05/02/2001        500        2,4
CE5   13/05/1995        500        3,4
CE6   23/08/1997       3300        3,3

SQL> SELECT nCompte, solde, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", nbOpCB
            FROM CptCourant;

NCOMP      SOLDE DATEOUV        NBOPCB
----- ---------- ---------- ----------
CC1          4030 01/02/2001        509
CC2          3000 15/02/2002          0
CC3           460 13/05/2000        678
…

SQL> SELECT m.ref_Client.num, m.ref_CptCourant.nCompte, m.dateOp, m.montant
           FROM Mouvement m;

REF_CLIENT.NUM REF_C DATEOP      MONTANT
-------------- ----- -------- ----------
             1 CC1   25/03/03        100
             1 CC1   25/03/03        -65
             1 CC1   27/03/03         40
             2 CC1   27/03/03        -80
             2 CC1   29/03/03        -50
             2 CC6   25/03/03         30
…

BEGIN
      CptEpargne_type.fermeCompte('CE4');
      CptEpargne_type.fermeCompte('CE?');
      CptCourant_type.fermeCompte('CC1');
      CptCourant_type.fermeCompte('CC?');
END;
/
Le solde du compte CE4 est : 500
Compte épargne supprimé.
Le compte CE? n'existe pas!
Le solde du compte CC1 est : 4030
Compte courant détruit, 5 mouvements supprimé(s)
Le compte CC? n'existe pas!

Procédure PL/SQL terminée avec succès.
```

37. Méthode `passeMvt(cli IN NUMBER, somme IN NUMBER )` qui réalise un débit (`somme` négatif) ou un crédit débit (`somme` positif) à la date du jour sur un compte courant donné. Opération faite par le client de numéro `cli`. Cette méthode met à jour le solde du compte et vérifie aussi que pour les retraits ne sont permis qu'au titulaire du compte ou à un signataire muni du droit `D`. On utilise un `PRAGMA EXCEPTION_INIT` pour dérouter dans la section exception une erreur du type « `ORA-02290: violation de contraintes (SOUTOU.NN_MVT_REF_CLIENT) de vérification` » à l'insertion d'un objet de `Mouvement` n'ayant pas une référence correcte vers la table `Client` (voir la section *À propos de l'intégrité référentielle*).

```
ALTER TYPE cptCourant_type
      ADD MEMBER PROCEDURE passeMvt (cli IN NUMBER, somme IN NUMBER) CASCADE;

CREATE OR REPLACE TYPE BODY cptCourant_type AS
…
  MEMBER PROCEDURE passeMvt (cli IN NUMBER, somme IN NUMBER) AS
    clientInconnu EXCEPTION;
    PRAGMA EXCEPTION_INIT(clientInconnu,-02290);
   BEGIN
    IF (somme < 0 AND
        NOT (SELF.estSignataire(cli, 'D') OR SELF.estTitulaire(cli))) THEN
      DBMS_OUTPUT.PUT_LINE('Le client '|| cli ||' n''est ni titulaire,
                  ni signataire pour le compte ' || SELF.nCompte ||
                  ' retrait interdit.');
      ELSE
      INSERT INTO Mouvement VALUES (mouvement_type(
            (SELECT REF(cl) FROM Client cl WHERE cl.num = cli),
            (SELECT REF(cou) FROM CptCourant cou WHERE
                        cou.nCompte = SELF.NCompte), SYSDATE, somme) );
        UPDATE cptCourant
            SET solde = solde + somme WHERE nCompte = SELF.NCompte;
        COMMIT;
    END IF;
  EXCEPTION
      WHEN clientInconnu THEN
            DBMS_OUTPUT.PUT_LINE('Client ' || cli || ' inconnu,
                                  opération impossible.');
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Compte ' || SELF.nCompte || ' inconnu,
                                  opération impossible.');
  END passeMvt;
END;
/
```

Testons cette méthode en passant différents mouvement, le dernier étant un débit interdit. Avant d'excuter le bloc l'état des tables :

```
SQL> SELECT nCompte, solde, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", nbOpCB
            FROM CptCourant;

NCOMP      SOLDE DATEOUV        NBOPCB
----- ---------- ---------- ----------
```

```
CC1            0 01/02/2001        509
CC2            0 15/02/2002          0
CC3            0 13/05/2000        678
CC4            0 17/09/2002          0
CC5            0 10/12/1998       1390
CC6            0 16/01/1965       2400
CC7            0 04/03/1976       5600

SQL> SELECT m.ref_Client.num, m.ref_CptCourant.nCompte, m.dateOp, m.montant
            FROM Mouvement m;

aucune ligne sélectionnée
```

```
DECLARE
 unCptCourant cptCourant_type;
BEGIN
  SELECT VALUE(cou) INTO unCptCourant
      FROM cptCourant cou WHERE cou.nCompte = 'CC3';
  --titulaire
  unCptCourant.passeMvt(4,50);
  unCptCourant.passeMvt(4,-10);
  --signataire droit D
  unCptCourant.passeMvt(1,10);
  unCptCourant.passeMvt(1,-5);
  --non signataire
  unCptCourant.passeMvt(3,5);
  unCptCourant.passeMvt(3,-10);
END;
/
```

Après exécution du bloc :

```
Le client 3 n'est ni titulaire, ni signataire pour le compte CC3 retrait
interdit.
Procédure PL/SQL terminée avec succès.

SQL> SELECT nCompte, solde, TO_CHAR(dateOuv,'DD/MM/YYYY') "DATEOUV", nbOpCB
            FROM CptCourant;

NCOMP      SOLDE DATEOUV        NBOPCB
----- ---------- ---------- ----------
CC1            0 01/02/2001        509
CC2            0 15/02/2002          0
CC3           50 13/05/2000        678
…

SQL> SELECT m.ref_Client.num, m.ref_CptCourant.nCompte, m.dateOp, m.montant
            FROM Mouvement m;

REF_CLIENT.NUM REF_C DATEOP      MONTANT
-------------- ----- -------- ----------
             4 CC3   01/04/03         50
             4 CC3   01/04/03        -10
```

```
             1 CC3   01/04/03         10
             1 CC3   01/04/03         -5
             3 CC3   01/04/03          5
```

Récapitulatif des méthodes

En récapitulant, si on voulait écrire un script de création des types qui contiennent au préalable les méthodes, ce serait le suivant.

```
CREATE TYPE client_type AS OBJECT
 (num NUMBER(5), nom VARCHAR2(30), adresse VARCHAR2(30),
  telephone_vry telephone_vry_type,
  MEMBER FUNCTION nbCepargne RETURN NUMBER,
  MEMBER PROCEDURE supprimeTel(indice In NUMBER),
  MEMBER PROCEDURE afficheInterets)
/
CREATE TYPE compte_type AS OBJECT
 (nCompte VARCHAR2(5), solde NUMBER(10,2), dateOuv DATE,
  ref_Client REF client_type,
  STATIC PROCEDURE fermeCompte (cpt IN VARCHAR2) )
 NOT FINAL NOT INSTANTIABLE
/
CREATE TYPE signataire_elt_nt_type AS OBJECT
 (num NUMBER(5), droit CHAR(1))
/
CREATE TYPE signataire_nt_type AS TABLE OF signataire_elt_nt_type
/
CREATE TYPE cptCourant_type UNDER compte_type
 (nbOpCB NUMBER(5), signataire_nt signataire_nt_type,
  MEMBER FUNCTION  nbSignataire RETURN NUMBER,
  MEMBER FUNCTION  nbSignataire (droitparam IN CHAR) RETURN NUMBER,
  MEMBER FUNCTION estTitulaire(cli IN NUMBER) RETURN BOOLEAN,
  MEMBER FUNCTION estSignataire(cli IN NUMBER, droitparam IN CHAR)
                  RETURN BOOLEAN,
  MEMBER PROCEDURE supprimeSign (nclient IN NUMBER),
  MEMBER PROCEDURE passeMvt     (cli IN NUMBER, somme IN NUMBER))
/
CREATE TYPE cptEpargne_type UNDER compte_type
 (txInt NUMBER(2,1),
  MEMBER FUNCTION calculeInterets RETURN NUMBER)
/
```

Le diagramme de classes complété aux méthodes est le suivant.

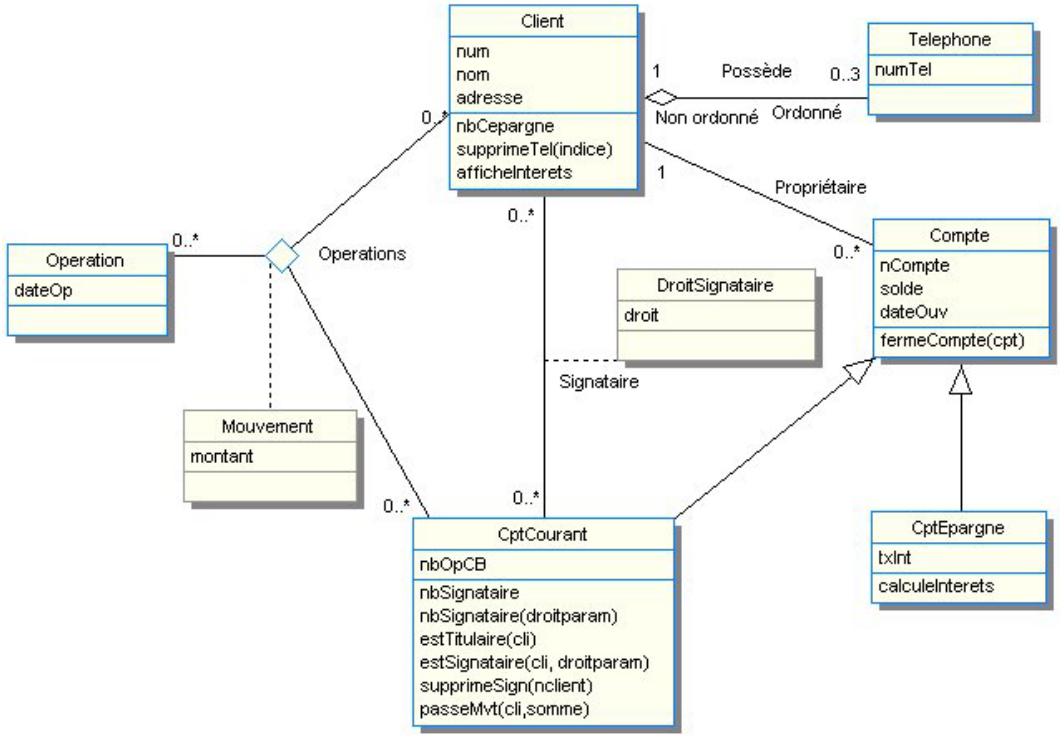

Figure S-2. Diagramme complet UML

Type récursifs

38. La modification du type consiste à ajouter une référence récursive.

```
ALTER TYPE client_type
      ADD ATTRIBUTE ref_ParrainClient REF client_type CASCADE;
```

Les mises à jour des références ne posent pas de problème particuler.

```
UPDATE Client c
  SET c.ref_ParrainClient =
          (SELECT REF(c) FROM Client c WHERE c.nom = 'Albaric')
   WHERE c.nom = 'Bidal' OR c.nom = 'Miranda';
UPDATE Client c
   SET c.ref_ParrainClient =
         (SELECT REF(c) FROM Client c WHERE c.nom = 'Miranda')
   WHERE c.nom = 'Payrissat';
```

La particularité de l'extraction des parrains des parrains des clients réside dans l'utilisation de la double notation pointée.

```
SELECT c.num, c.nom,
       c.ref_ParrainClient.ref_ParrainClient.nom "Parrain du parrain"
  FROM Client c WHERE c.ref_ParrainClient.ref_ParrainClient IS NOT NULL;

       NUM NOM                            Parrain du parrain
---------- ------------------------------ ------------------------------
         4 Payrissat                      Albaric
```

Collection multi niveau

39. La collection multi niveaux à définir est contenue dans la table relationnelle illustrée à la figure suivante :

Statistiques

num	{**rendez_vous_vry**}			
	conseiller	mois	{**produits_vendus_nt**}	
			nomProd	valeur
1	Filou	Janv 2004	PEL	150
			CEL	500
	Screau	Avr 2004	CODEVI	400
	Manteur	Mai 2004	SICAV_A	700
			ACTIONS_B	400
	Screau	Aout 2004	EPARVIE	300

Figure S-3. Collection multi niveaux

La création des types et de la table est la suivante.

```
CREATE TYPE produits_vendus_elt_nt_type AS OBJECT
 (nomProd VARCHAR2(10), valeur NUMBER(7,2))
/
CREATE TYPE produits_vendus_nt_type AS TABLE OF produits_vendus_elt_nt_type
/
CREATE TYPE rendez_vous_elt_vry_type AS OBJECT
 (conseiller VARCHAR2(15), mois VARCHAR2(20),
  produits_vendus_nt produits_vendus_nt_type)
/
CREATE TYPE rendez_vous_vry_type AS VARRAY(12) OF rendez_vous_elt_vry_type
/
CREATE TABLE Statistiques
 (num NUMBER(5) PRIMARY KEY, rendez_vous_vry rendez_vous_vry_type)
  VARRAY rendez_vous_vry STORE AS LOB tabqualifs_LOB
  (ENABLE STORAGE in ROW STORAGE
     (INITIAL 100K NEXT 100K MINEXTENTS 2 MAXEXTENTS 5 PCTINCREASE 10));
```

L'initialisation fait intervenir les quatre constructeurs.

```
INSERT INTO Statistiques
 VALUES (1,rendez_vous_vry_type(
```

```
                rendez_vous_elt_vry_type('Filou','Janvier 2004',
                  produits_vendus_nt_type (
                    produits_vendus_elt_nt_type ('PEL',150),
                    produits_vendus_elt_nt_type ('CEL',500))),
                rendez_vous_elt_vry_type('Screau','Avril 2004',
                  produits_vendus_nt_type (
                    produits_vendus_elt_nt_type ('CODEVI',400))),
                rendez_vous_elt_vry_type('Manteur','Mai 2004',
                  produits_vendus_nt_type (
                    produits_vendus_elt_nt_type ('SICAV_A',700),
                    produits_vendus_elt_nt_type ('ACTIONS_B',400))),
                rendez_vous_elt_vry_type('Screau','Aout 2004',
                  produits_vendus_nt_type (
                    produits_vendus_elt_nt_type ('EPARVIE',300))) ));
```

L'ajout d'un rendez-vous avec le conseiller 'Larnac' au cours du mois de Septembre 2004 (sans vente de produit) est programmé dans le bloc suivant. Il s'agit d'étendre un tableau chargé en mémoire puis de mettre à jour la colonne de la table avec ce dernier.

```
DECLARE
  tableau_rdv rendez_vous_vry_type;
BEGIN
  SELECT rendez_vous_vry INTO tableau_rdv FROM Statistiques WHERE num = 1
     FOR UPDATE OF rendez_vous_vry;
  IF tableau_rdv.COUNT < 12 THEN
      tableau_rdv.EXTEND;
      tableau_rdv(tableau_rdv.LAST) := rendez_vous_elt_vry_type
            ('Larnac','Septembre 2004',produits_vendus_nt_type());
       UPDATE Statistiques SET rendez_vous_vry = tableau_rdv WHERE num = 1;
  END IF;
END;
/
```

L'ajout d'une vente d'un produit au rendez-vous précédemment inséré est programmée dans le bloc suivant. Il s'agit d'étendre un tableau (représentant la collection *nested table*) et mettre à jour un élément entier du *varray*.

```
DECLARE
  tableau_rdv rendez_vous_vry_type;
  tableau_prod produits_vendus_nt_type;
BEGIN
  SELECT rendez_vous_vry INTO tableau_rdv FROM Statistiques WHERE num = 1
     FOR UPDATE OF rendez_vous_vry ;
  SELECT produits_vendus_nt INTO tableau_prod FROM TABLE
    (SELECT rendez_vous_vry FROM Statistiques WHERE num = 1)
       WHERE conseiller = 'Larnac' AND mois ='Septembre 2004';
  tableau_prod.EXTEND;
  tableau_prod (tableau_prod.LAST) :=
            produits_vendus_elt_nt_type('LIVRET-A',800);
  tableau_rdv(tableau_rdv.LAST).produits_vendus_nt :=tableau_prod;
  UPDATE Statistiques SET rendez_vous_vry = tableau_rdv WHERE num = 1;
END;
/
```

Les pseudo jointures réalisées par l'opérateur `TABLE` permettent d'extraire aisément la somme des produits vendus par conseiller. L'élection de celui qui a dépossédé le plus de clients dans l'année est sans appel !

```
SELECT r.conseiller, SUM(p.valeur)
FROM Statistiques s, TABLE(s.rendez_vous_vry) r, TABLE(r.produits_vendus_nt) p
GROUP BY r.conseiller ORDER BY 2 DESC;
```

```
CONSEILLER      SUM(P.VALEUR)
--------------- -------------
Manteur                 1100
Larnac                   800
Screau                   700
Filou                    650
```

Vues

Définition de la vue

Avant de définir la vue objet, il faut définir les vues avec l'option précisant la clé primaire des tables, ceci pour permettre aux références de la vue objet principale de pouvoir cibler tout enregistrement.

```
CREATE VIEW Employes_VOR OF employes_type
 WITH OBJECT IDENTIFIER(numEmp) AS SELECT * FROM Employes;

CREATE VIEW Client_VOR OF client_type
 WITH OBJECT IDENTIFIER(num) AS SELECT * FROM Client;
```

La structure de la vue objet principale nécessite de définir plusieurs types (deux collections distinctes à créer).

```
CREATE TYPE employes_type AS OBJECT
 (numEmp VARCHAR2(5), nomEmp VARCHAR(20), age NUMBER(2), resp VARCHAR2(6))
/
CREATE TYPE elt_nt_equipe_type AS OBJECT
 (refEmp REF employes_type, dateEnt DATE)
/
CREATE TYPE equipe_nt_type AS TABLE OF elt_nt_equipe_type
/
CREATE TYPE elt_nt_contrats_type AS OBJECT
 (refEmp REF employes_type, refCli REF client_type,
  nomProd VARCHAR2(10), dateCont DATE)
/
CREATE TYPE contrats_nt_type AS TABLE OF elt_nt_contrats_type
/
CREATE TYPE agenceVOR_type AS OBJECT
 (numAg VARCHAR2(6), nomAg VARCHAR(20),
  equipe_nt equipe_nt_type, contrats_nt contrats_nt_type)
/
```

La première collection est définie à l'aide des directives `CAST` et `MULTISET`, la jointure avec la table `Travaille` permet d'extraire les salariés (en cours) pour chaque agence.

La seconde collection contient deux directives MAKE_REF qui créeent des références en minimisant les jointures. Cette directive associe automatiquement un objet cible (d'une vue objet) à l'aide de la valeur de la clé primaire de la table source (ici `numEmp` et `numCli`).

```
CREATE VIEW Agence_VOR OF agenceVOR_type
 WITH OBJECT IDENTIFIER(numAg)
 AS SELECT a.numAg, a.nomAg,
           CAST( MULTISET (SELECT REF(e), t.dateDeb
                           FROM  Employes_VOR e, Travaille t
                           WHERE t.numEmp = e.numEmp
                           AND   t.dateFin IS NULL
                           AND   t.numAg  = a.numAg) AS equipe_nt_type
               ) AS equipeCAST,
           CAST( MULTISET (SELECT MAKE_REF(Employes_VOR, cv.numEmp),
                                  MAKE_REF(Client_VOR, cv.numCli),
                                  cv.nomProd, cv.dateCont
                           FROM ContratsVendus cv
                           WHERE cv.numAg = a.numAg) AS contrats_nt_type
               ) AS contratsCAST
 FROM Agence a;
```

Interrogation de la vue

Les trois requêtes programment des pseudo jointures à l'aide de l'opérateur `TABLE`. Il est aussi fait usage de jointure implicite (notation pointée d'une référence). La dernière met en œuvre en plus une jointure traditionnelle (ici avec une table relationnelle).

40.

```
SELECT   a.numAg, a.nomAg, ant.refEmp.nomEmp, ant.refEmp.age, ant.dateEnt
FROM     Agence_VOR a, TABLE(a.equipe_nt) ant
ORDER BY 1,4,5;
```

41.

```
SELECT   ant.nomProd, a.numAg, ant.refEmp.nomEmp, ant.refCli.nom, ant.dateCont
FROM     Agence_VOR a, TABLE(a.contrats_nt) ant
ORDER BY 1,2;
```

42.

```
SELECT  ant.nomProd, a.numAg, ant.refEmp.nomEmp, ant.refCli.nom, ant.dateCont
 FROM   Agence_VOR a, TABLE(a.contrats_nt) ant, Employes e
 WHERE  e.resp = a.numAg
 AND    ant.refEmp.numEmp = e.numEmp
 ORDER BY 1,2;
```

Programmation de méthodes

Avant de coder les méthodes, il faut modifier la spécification du type de la vue.

```
ALTER TYPE agenceVOR_type
      ADD MEMBER PROCEDURE passeContrat(nEmp IN VARCHAR2, nCli IN NUMBER,
                                  nomProduit IN VARCHAR2) CASCADE;

ALTER TYPE agenceVOR_type
           ADD MEMBER PROCEDURE mutation(nEmp IN VARCHAR2,
                                  nAgCible IN VARCHAR2) CASCADE;
```

43. La méthode membre `passeContrat(nEmp, nCli, nomProduit)` mémorise la vente d'un produit au client par l'employé d'une agence donnée.

```
CREATE TYPE BODY agenceVOR_type AS
  MEMBER PROCEDURE passeContrat(nEmp IN VARCHAR2, nCli IN NUMBER, nomProduit IN
VARCHAR2) IS
    trace    NUMBER(1) := 1;
    v_nom    VARCHAR2(30);
    v_nombre NUMBER(4);
  BEGIN
   SELECT nom INTO v_nom FROM Client WHERE num = nCli;
   trace := 2;
   SELECT nomEmp INTO v_nom FROM Employes WHERE numEmp = nEmp;
   SELECT COUNT(*) INTO v_nombre FROM ContratsVendus
          WHERE numAg = SELF.numAg AND  numEmp = nEmp
          AND  numCli = nCli AND nomProd = nomProduit;
   IF (v_nombre <> 0) THEN
      DBMS_OUTPUT.PUT_LINE('Produit déjà vendu par cet employé à ce client');
   ELSE
   INSERT INTO ContratsVendus VALUES(SELF.numAg,nEmp,nCli,nomProduit,SYSDATE);
   COMMIT;
   END IF;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
     IF trace = 1 THEN
       DBMS_OUTPUT.PUT_LINE('Pas de client de ce numéro : ' || nCli);
     ELSE
       DBMS_OUTPUT.PUT_LINE('Pas d''employé de ce numéro : ' || nEmp);
     END IF;
  WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Erreur Oracle ' || SQLERRM);
  END passeContrat;
MEMBER PROCEDURE mutation(nEmp IN VARCHAR2, nAgCible IN VARCHAR2) IS
 …
  END mutation;
END;
/
```

44. La méthode membre `mutation(nEmp, nAgCible)` opère la mutation de l'employé dans l'agence.

```
MEMBER PROCEDURE mutation(nEmp IN VARCHAR2, nAgCible IN VARCHAR2) IS
   trace    NUMBER(1) := 1;
   v_nom    VARCHAR2(30);
   v_nombre NUMBER(4);
 BEGIN
   SELECT nomAg INTO v_nom FROM Agence WHERE numAg = nAgCible ;
   trace := 2;
   SELECT nomEmp INTO v_nom FROM Employes WHERE numEmp = nEmp;
   trace := 3;
   SELECT numEmp INTO v_nom FROM Travaille
          WHERE numEmp =  nEmp  AND numAg = SELF.numAg AND dateFin IS NULL;
   SELECT COUNT(*) INTO v_nombre FROM Employes
          WHERE resp = SELF.numAg AND numEmp = nEmp;
   IF (v_nombre <> 0) THEN
      DBMS_OUTPUT.PUT_LINE('Il faut nommer un nouveau responsable pour
                           l''agence ' || SELF.numAg);
   END IF;
   UPDATE Travaille SET dateFin = SYSDATE
          WHERE numAg = SELF.numAg AND numEmp = nEmp;
   UPDATE Employes SET resp = NULL WHERE numEmp = nEmp;
   INSERT INTO Travaille VALUES(nAgCible, nEmp, SYSDATE, NULL);
   DBMS_OUTPUT.PUT_LINE('Mutation de ' || SELF.numAg || ' à ' || nAgCible ||
                        ' enregistrée.');
   COMMIT;
 EXCEPTION
    WHEN NO_DATA_FOUND THEN
     IF trace = 1 THEN
       DBMS_OUTPUT.PUT_LINE('Pas d''agence de ce numéro : ' || nAgCible);
     ELSIF trace = 2 THEN
       DBMS_OUTPUT.PUT_LINE('Pas d''employé de ce numéro : ' || nEmp);
     ELSIF trace = 3 THEN
       DBMS_OUTPUT.PUT_LINE('L''employé  n''appartient  pas  à  l''agence  '  ||
SELF.numAg);
     END IF;
  WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Erreur Oracle ' || SQLERRM);
END mutation;
```