

# Sommaire

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Partie I : Relationnel-Objet</b>	<b>2</b>
2.1	Modélisation orientée objet . . . . .	2
2.2	Création des TableSpaces et utilisateur . . . . .	3
2.3	Langage de définition de données . . . . .	4
2.4	Création des instances dans les tables . . . . .	8
2.5	Langage d'interrogation de données . . . . .	13

## Liste des Figures

1	Diagramme de classes représentant le schéma relationnel en un schéma Objet . . . . .	2
2	Résultats de la création des TableSpaces SQL3_TBS et SQL3_TempTBS . . . . .	3
3	Résultats de la création de l'utilisateur SQL3 . . . . .	4
4	Résultats de l'attribution de tous les privilèges à SQL3 . . . . .	4
5	Résultats de la création des types incomplets . . . . .	4
6	Résultats de la création des tables imbriquées des références . . . . .	5
7	Définition des méthodes du type tagence . . . . .	5
8	Corps méthodes de tagence . . . . .	5
9	Ajout des signatures des méthodes de tsuccursale . . . . .	6
10	Corps méthodes de tsuccursale . . . . .	6
11	Définition des tables avec les contraintes nécessaires . . . . .	7
12	Insertion des succursales . . . . .	8
13	Insertion des agences . . . . .	9
14	mise à agences des ref d'agences dans la tables succursales . . . . .	10
15	Résultats d'insertion des 100 clients . . . . .	10
16	mise à jour comptes dans la tables clients . . . . .	11
17	resultat representant la mise à jour de comptes dans la tables clients . . . . .	11
18	mise à jour les collections d'opérations pour les comptes . . . . .	11
19	resultat representant la mise à jour les collections d'opérations pour les comptes . . . . .	12
20	mise à jour les collections de prêts pour les comptes . . . . .	12
21	resultat representant la mise à jour les collections de prêts pour les comptes . . . . .	12
22	mise à jour les collections de compte pour les agence . . . . .	12
23	resultat epresentant la mise à jour les collections de compte pour les agence . . . . .	13
24	Liste des comptes d'une agence donnée, dont les propriétaires sont des entreprises . . . . .	13
25	Les prêts effectués auprès des agences rattachées à une succursale donnée . . . . .	13
26	Liste des comptes sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022 . . . . .	14
27	montant total des crédits effectués sur un compte en 2023 . . . . .	15
28	les prêts non encore soldés à ce jour . . . . .	16

# 1 Introduction

Les systèmes de gestion de bases de données revêtent un rôle primordial dans le monde moderne, où les données représentent un atout stratégique pour les entreprises et organisations. Que ce soit dans le domaine bancaire, financier ou tout autre secteur d'activité, la maîtrise des technologies liées aux bases de données est devenue un prérequis indispensable. Le présent projet a pour objectif d'explorer les concepts fondamentaux des bases de données relationnelles-objet ainsi que les modèles NoSQL orientés documents, en prenant pour cas d'étude la gestion des opérations et des prêts bancaires.

Dans un premier temps, une modélisation orientée objet sera réalisée à partir du schéma relationnel fourni, permettant de mettre en évidence les entités, leurs attributs et les relations qui les lient. Cette approche permettra de représenter de manière fidèle la structure de la base de données et facilitera la compréhension de son fonctionnement.

Par la suite, une attention particulière sera accordée à la définition des types abstraits, des associations et des méthodes nécessaires à l'implémentation de la base de données. Ces éléments constitueront la pierre angulaire de la conception et de la manipulation des données dans un environnement relationnel-objet.

Dans un second volet, le projet explorera le paradigme NoSQL orienté documents, en proposant une modélisation adaptée à ce modèle de données. Cette étape cruciale permettra d'appréhender les spécificités de ce type de base de données et d'en évaluer les avantages et les limites dans le cadre de l'application étudiée.

Enfin, une série de requêtes sera mise en œuvre, tant dans l'environnement relationnel-objet que dans le modèle NoSQL orienté documents. Ces requêtes, issues de cas d'utilisation concrets, permettront de valider la pertinence des modélisations proposées et d'en analyser les performances respectives.

Ce projet, à la croisée des chemins entre la théorie et la pratique, représente une opportunité unique d'approfondir nos connaissances dans le domaine des bases de données avancées et de consolider notre expertise dans la manipulation des différents paradigmes de stockage et de traitement des données.

## 2 Partie I : Relationnel-Objet

### 2.1 Modélisation orientée objet

Pour représenter le schéma relationnel, une modélisation orientée objet a été réalisée par le biais d'un diagramme de classes UML. Ce formalisme graphique normalisé permet de mettre en évidence les entités, leurs attributs et les relations les liant.

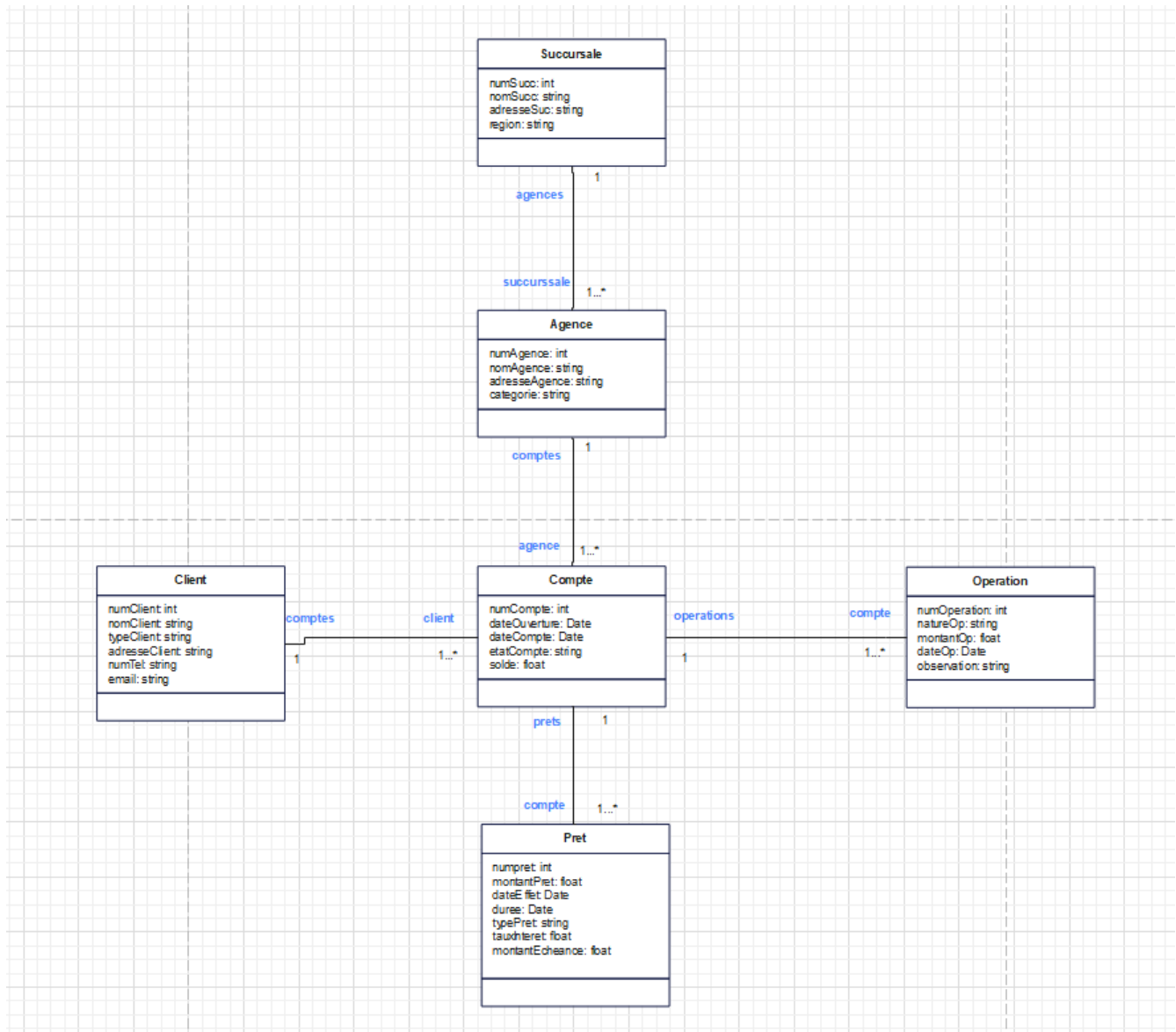


Figure 1: Diagramme de classes représentant le schéma relationnel en un schéma Objet

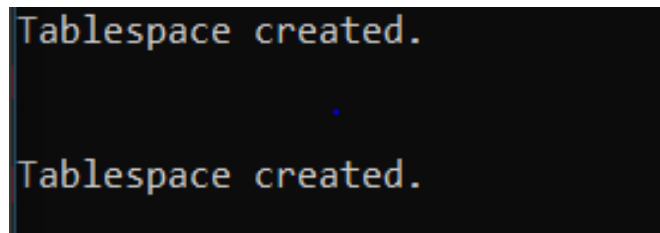
Dans la modélisation orientée objet, les relations entre les différentes entités ont été représentées à travers des rôles spécifiques:

- La classe Succursale possède un rôle "agences" qui est une collection de références vers des objets de type Agence. Réciproquement, la classe Agence définit un rôle "succursale" qui est une référence unique vers un objet Succursale.
- La classe Agence détient également un rôle "comptes" constitué d'une collection de références vers des objets Compte, tandis que la classe Compte établit un rôle "agence" en référençant un unique objet Agence.
- la classe Client gère un rôle "comptes" répertoriant les références vers les objets Compte associés, et la classe Compte définit un rôle "client" faisant correspondre chaque instance à un seul objet Client.
- Les opérations effectuées sur les comptes sont modélisées à travers le rôle "opérations" de la classe Compte, collection de références vers des objets Opération, chaque instance d'Opération possédant un rôle "compte" la liant à un unique objet Compte.
- les prêts sont représentés par le rôle "prêts" de la classe Compte, ensemble de références vers des objets Prêt, chacun d'eux étant associé à un seul Compte par le biais du rôle "compte".

## 2.2 Création des TableSpaces et utilisateur

Vous trouverez les scripts de cette partie en suivant ce lien:[ici](#)

- **Création des TableSpaces SQL3\_TBS et SQL3\_TempTBS**



```
Tablespace created.  
.  
Tablespace created.
```

Figure 2: Résultats de la création des TableSpaces SQL3\_TBS et SQL3\_TempTBS

- Création de l'utilisateur SQL3 avec attribution des deux tablespaces précédemment créés

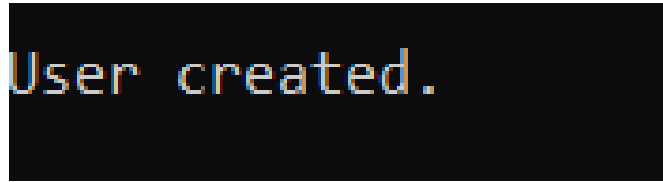


Figure 3: Résultats de la création de l'utilisateur SQL3

- Accorder tous les privilèges à l'utilisateur en question

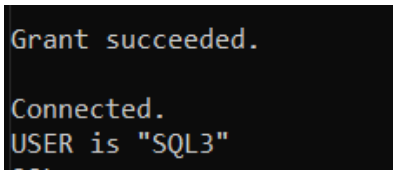


Figure 4: Résultats de l'attribution de tous les privilèges à SQL3

## 2.3 Langage de définition de données

Vous trouverez les scripts de cette partie en suivant ce lien:[ici](#)

- Définition de tous les types abstraits nécessaires et de toutes les associations qui existent

Nous commençons d'abord par créer les types de manière incomplète

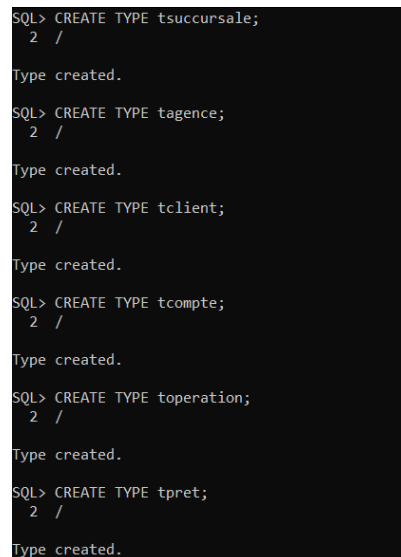


Figure 5: Résultats de la création des types incomplets

```

SQL> CREATE TYPE tset_ref_agences AS TABLE OF REF tagence;
2 /

Type created.

SQL> CREATE TYPE tset_ref_comptes AS TABLE OF REF tcompte;
2 /

Type created.

SQL> CREATE TYPE tset_ref_operations AS TABLE OF REF toperation;
2 /

Type created.

SQL> CREATE TYPE tset_ref_prets AS TABLE OF REF tpreset;
2 /

Type created.

```

Figure 6: Résultats de la création des tables imbriquées des références

Puis nous créons les tables imbriquées des références qui vont nous aider à représenter les associations

Maintenant nous pouvons créer les types on y ajoutant les références où c'est nécessaire (vous trouverez le script sur le lien précédent)

- **Définition des méthodes**

Nous commençons par définir la signature des méthodes nous permettant de le nombre de prêts pour chaque agence et le montant global des prêts effectués durant la période du 01-01-2020 au 01-01-2024 pour une agence.

```

SQL> ALTER TYPE tagence ADD MEMBER FUNCTION nbr_pret RETURN NUMBER CASCADE;

Type altered.

SQL>
SQL> ALTER TYPE tagence ADD MEMBER FUNCTION montant_global_pret(numagence NUMBER) RETURN NUMBER CASCADE;

Type altered.

```

Figure 7: Définition des méthodes du type tagence

puis nous définissons le corps des deux méthodes

```

SQL> CREATE OR REPLACE TYPE BODY tagence AS
2   MEMBER FUNCTION nbr_pret RETURN NUMBER IS
3     nb NUMBER;
4     BEGIN
5       SELECT COUNT(*) INTO nb FROM TABLE(SELf.comptes) c, TABLE(VALUE(c).prets) p;
6       RETURN nb;
7     END nbr_pret;
8   MEMBER FUNCTION montant_global_pret(numagence NUMBER) RETURN NUMBER IS
9     total_montant NUMBER;
10    BEGIN
11      SELECT SUM(VALUE(p).montantPret) INTO total_montant FROM
12        TABLE(comptes) c, TABLE(VALUE(c).prets) p
13        WHERE VALUE(c).agence.NumAgence = numagence
14        AND value(p).dateEffet >= TO_DATE('01-01-2020', 'DD-MM-YYYY')
15        AND value(p).dateEffet + value(p).duree <= TO_DATE('01-01-2024', 'DD-MM-YYYY')
16      RETURN total_montant;
17    END montant_global_pret;
18  END;
19 /

Type body created.

```

Figure 8: Corps méthodes de tagence

Nous faisons de même pour les méthodes de calcul de nombre d'agences principales pour chaque succursale et la liste de toutes les agences secondaires (avec la succursale rattachée) ayant au moins un prêt ANSEJ

```
SQL> CREATE OR REPLACE TYPE agence_succ AS OBJECT(
2     nom_agence VARCHAR2(50),
3     nom_succursale VARCHAR2(50)
4 );
5 /

Type created.

SQL> CREATE TYPE liste_agence_succ AS TABLE OF agence_succ;
2 /

Type created.

SQL> ALTER TYPE tsuccursale ADD MEMBER FUNCTION nbr_agences_principales RETURN NUMBER CASCADE;

Type altered.

SQL> ALTER TYPE tsuccursale ADD MEMBER FUNCTION show_agences_secondaires RETURN liste_agence_succ CASCADE;

Type altered.
```

Figure 9: Ajout des signatures des méthodes de tsuccursale

```
SQL> CREATE OR REPLACE TYPE BODY tsuccursale AS
2     MEMBER FUNCTION nbr_agences_principales RETURN NUMBER IS
3         nb NUMBER;
4         BEGIN
5             SELECT COUNT(*) INTO nb FROM TABLE(SELF.agences) a
6             WHERE VALUE(a).categorie = 'Principale';
7             RETURN nb;
8         END nbr_agences_principales;
9
10        MEMBER FUNCTION show_agences_secondaires RETURN liste_agence_succ IS
11        liste liste_agence_succ;
12
13        BEGIN
14            SELECT CAST(MULTISET(
15                SELECT DISTINCT deref(value(a)).nomAgence ,self.nomSucc
16                FROM TABLE(self.agences) a
17                WHERE deref(VALUE(a)).categorie = 'Secondaire'
18                AND EXISTS (
19                    SELECT 1
20                    FROM TABLE(VALUE(a).comptes) c, TABLE(value(c).prets) p
21                    WHERE value(p).typePret = 'ANSEJ'
22                )
23            ) AS liste_agence_succ)
24            INTO liste
25            FROM dual;
26        RETURN liste;
27    END show_agences_secondaires;
28
29    END;
30 /

Type body created.
```

Figure 10: Corps méthodes de tsuccursale

- **Définition des tables nécessaires à la base de données**

Puisque la taille des numéros identifiant de chaque table a été faite lors de la création des types, nous y avons ajouté que les contraintes manquantes comme montre la figure ci-dessous:

```

SQL> CREATE TABLE Succursale OF tsuccursale(
  2     CONSTRAINT pk_succursale PRIMARY KEY (numSucc),
  3     CONSTRAINT ch_region CHECK (region IN ('Nord', 'Sud', 'Est', 'Ouest'))
  4 )
  5 )
  6 nested table agences STORE AS agence_succursale;

Table created.

SQL>
SQL> CREATE TABLE Agence OF tagence(
  2     CONSTRAINT pk_agence PRIMARY KEY (numAgence),
  3     CONSTRAINT ch_categorie CHECK (categorie IN ('Principale', 'Secondaire'))
  4 )
  5 nested table comptes STORE AS compte_agence;

Table created.

SQL>
SQL> CREATE TABLE Compte OF tcompte(
  2     CONSTRAINT pk_compte PRIMARY KEY (numCompte),
  3     CONSTRAINT ch_etat CHECK (etatCompte IN ('Actif', 'BLOQUE'))
  4 )
  5 NESTED TABLE prets STORE AS pret_compte, NESTED TABLE operations STORE AS operation_compte;

Table created.

SQL>
SQL>
SQL> CREATE TABLE Client OF tclient(
  2     CONSTRAINT pk_client PRIMARY KEY (numClient),
  3     CONSTRAINT ch_type_client CHECK (typeClient IN ('Particulier', 'Entreprise'))
  4 )
  5 nested table comptes STORE AS compte_client;

Table created.

SQL>
SQL> CREATE TABLE operation OF toperation(
  2     CONSTRAINT pk_operation PRIMARY KEY (numOperation),
  3     CONSTRAINT ch_nature_op CHECK (natureOp IN ('Credit', 'Debit'))
  4 );

Table created.

SQL>
SQL>
SQL> CREATE TABLE Pret OF tpret(
  2     CONSTRAINT pk_pret PRIMARY KEY (numPret),
  3     CONSTRAINT ch_type_pret CHECK (typePret IN ('Vehicule', 'Immobilier', 'ANSEJ', 'ANJEM'))
  4 );

Table created.

```

Figure 11: Définition des tables avec les contraintes nécessaires



## 2.4 Création des instances dans les tables

- **insertion des succursales**

pour insérer une succursales nous tapons la commande suivantes:

```
INSERT INTO Succursale VALUES ( tsuccursale(numSucc, 'nomSucc', 'adrSucc', 'region', tset_ref_agences()) );
```

```
SQL> -- insertion des succursales
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(1, 'Succursale Alger Centre', '12 Rue Didouche Mourad, Alger', 'Nord', tset_ref_agences())
  3 );
1 row created.

SQL>
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(2, 'Succursale Oran', '25 Boulevard de la Soummam, Oran', 'Ouest', tset_ref_agences())
  3 );
1 row created.

SQL>
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(3, 'Succursale Constantine', '5 Rue Zighoud Youcef, Constantine', 'Est', tset_ref_agences())
  3 );
1 row created.

SQL>
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(4, 'Succursale Ouargla', '8 Avenue du 1er Novembre, Ouargla', 'Sud', tset_ref_agences())
  3 );
1 row created.

SQL>
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(5, 'Succursale Tizi Ouzou', '15 Boulevard Maurice Audin, Tizi Ouzou', 'Nord', tset_ref_agences())
  3 );
1 row created.

SQL>
SQL> INSERT INTO Succursale VALUES (
  2   tsuccursale(6, 'Succursale Sétif', '3 Rue des Frères Bouadou, Sétif', 'Est', tset_ref_agences())
  3 );
1 row created.
```

Figure 12: Insertion des succursales

- **insertion des agences**

pour insérer une agence nous tapons la commande suivantes:

```
INSERT INTO agences VALUES (numAgence, 'nomAgence', 'adresseAgence',
'categorie',
(SELECT REF(s) FROM succursales s
WHERE s.numSucc = numSucc_auquel_elle_appartient), tset_ref_comptes());
);
```

```

SQL> -- Succursale Alger Centre (numSucc = 1)
SQL> INSERT INTO Agence VALUES (tagence(101, 'Agence Alger Centre', '10 Rue Didouche Mourad, Alger', 'Principale', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 1), tset_ref_comptes()));
1 row created.

SQL> INSERT INTO Agence VALUES (tagence(102, 'Agence Bab El Oued', '5 Rue Ahmed Ouaked, Alger', 'Secondaire', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 1), tset_ref_comptes()));
1 row created.

SQL> INSERT INTO Agence VALUES (tagence(103, 'Agence Belcourt', '2 Rue Belcourt, Alger', 'Secondaire', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 1), tset_ref_comptes()));
1 row created.

SQL> INSERT INTO Agence VALUES (tagence(104, 'Agence Bab Ezzouar', '18 Boulevard Bab Ezzouar, Alger', 'Secondaire', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 1), tset_ref_comptes()));
1 row created.

SQL> INSERT INTO Agence VALUES (tagence(105, 'Agence El Harrach', '25 Rue Hassiba Ben Bouali, El Harrach', 'Secondaire', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 1), tset_ref_comptes()));
1 row created.

SQL>
SQL> -- Succursale Oran (numSucc = 2)
SQL> INSERT INTO Agence VALUES (tagence(201, 'Agence Oran Centre', '20 Boulevard de la Soummam, Oran', 'Principale', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 2), tset_ref_comptes()));
1 row created.

SQL> INSERT INTO Agence VALUES (tagence(202, 'Agence Oran El Mokrani', '8 Rue El Mokrani, Oran', 'Secondaire', (SELECT REF(s) FROM Succursale s WHERE s.numSucc = 2), tset_ref_comptes()));
1 row created.

```

Figure 13: Insertion des agences

une fois fini, nous devons mettre à jour l'attribut agences de la table succursales qui est une collection de références vers des objets de type agence. Pour cela nous tappons la commande suivante:

```

insert into table (select s.agences from Succursale s where s.numSucc= n)
(select ref(a) from Agence a
where a.succursale=(select ref(s) from Succursale s where s.numSucc=n)); );

```

Cette commande SQL effectue deux opérations :

1. La sous-requête:
 

```
SELECT REF(a) FROM Agence a WHERE
a.succursale=(SELECT REF(s) FROM Succursale s WHERE numSucc=n)
```

 récupère les références des agences appartenant à la succursale dont le numéro est
2. La requête principale:
 

```
INSERT INTO TABLE \ (SELECT s.agences FROM Succursale s WHERE numSucc=n)
```

 insère les références des agences obtenues à l'étape précédente dans la table de données objet imbriquée **agences** de la succursale dont le numéro est n.

```

SQL> insert into table (select s.agences from Succursale s where numSucc=1)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=1));
5 rows created.

SQL>
SQL> insert into table (select s.agences from Succursale s where numSucc=2)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=2));
5 rows created.

SQL>
SQL> insert into table (select s.agences from Succursale s where numSucc=3)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=3));
5 rows created.

SQL>
SQL> insert into table (select s.agences from Succursale s where numSucc=4)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=4));
5 rows created.

SQL>
SQL> insert into table (select s.agences from Succursale s where numSucc=5)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=5));
3 rows created.

SQL>
SQL> insert into table (select s.agences from Succursale s where numSucc=6)
  2 (select ref(a) from Agence a where a.succursale=(select ref(s) from Succursale s where numSucc=6));
2 rows created.

SQL>

```

Figure 14: mise à agences des ref d'agences dans la tables succursales

- **Insertions des 100 clients:**

pour cela nous avons utiliser la procedure se trouvant dans le lien cité au dessus.

```

PL/SQL procedure successfully completed.

```

Figure 15: Résultats d'insertion des 100 clients

Ce script génère des données pour une simulation bancaire en effectuant les actions suivantes:

- Création de 100 clients avec des informations aléatoires telles que leur nom, adresse, numéro de téléphone et adresse e-mail.
- Pour certains clients, il crée un ou plusieurs comptes avec des numéros de compte, des dates d'ouverture et de fermeture aléatoires, ainsi qu'un solde et un état aléatoires.
- Pour certains comptes, des opérations (crédit ou débit) sont ajoutées avec des numéros d'opération, des montants et des dates aléatoires.
- Pour certains autres comptes, des prêts sont ajoutés avec des numéros de prêt, des montants, des dates d'effet, des durées et des types de prêt aléatoires, parmi Vehicule, Immobilier, ANSEJ et ANJEM.

par la suite, il faut mettre à jour les collections imbriquées pour les comptes, opérations et prêts

```
SQL> UPDATE Client c
  2 SET c.comptes = (SELECT CAST(MULTISET(
  3                     SELECT REF(co)
  4                     FROM Compte co
  5                     WHERE co.client = REF(c)
  6                     ) AS tset_ref_comptes)
  7                     FROM dual)
  8 WHERE EXISTS (SELECT 1
  9                FROM Compte co
 10                WHERE co.client = REF(c));

33 rows updated.
```

Figure 16: mise à jour comptes dans la tables clients

```
Entreprise
Adresse_099
0651297287 client099@example.com

NUMCLIENT NOMCLIENT
-----
TYPECLIENT
-----
ADRESSECLIENT
-----
NUMTEL      EMAIL
-----
COMPTES
-----
TSET_REF_COMPTES(00002202089C2B4621DEFA4550907883FC5578D22D095F6187DCCE4B5BB2A87
019D9CEE086, 00002202086713847E8EB04D75A9A67E8D3BBCD827095F6187DCCE4B5BB2A87019D
9CEE086)
```

Figure 17: resultat representant la mise à jour de comptes dans la tables clients

```
SQL> UPDATE Compte c
  2 SET c.operations = (SELECT CAST(MULTISET(
  3                     SELECT REF(op)
  4                     FROM Operation op
  5                     WHERE op.compte = REF(c)
  6                     ) AS tset_ref_operations)
  7                     FROM dual)
  8 WHERE EXISTS (SELECT 1
  9                FROM Operation op
 10                WHERE op.compte = REF(c));

19 rows updated.
```

Figure 18: mise à jour les collections d'opérations pour les comptes

```
PRETS
-----
CLIENT
-----
AGENCE
-----
3037428423 28-MAY-20 28-SEP-16 Actif 248601
TSET_REF_OPERATIONS(00002202084F0CEC1353164BFA93F0CAA17D6161E7B1E936FD53DB4FED9B
730A051B136CBA, 0000220208D4D872700EBF463CA8FE9B3CAF1B5DF5B1E936FD53DB4FED9B730A
```

Figure 19: resultat representant la mise à jour les collections d'opérations pour les comptes

```
SQL> UPDATE Compte c
  2 SET c.prets = (SELECT CAST(MULTISET(
  3 SELECT REF(p)
  4 FROM Pret p
  5 WHERE p.compte = REF(c)
  6 ) AS tset_ref_prets)
  7 FROM dual)
  8 WHERE EXISTS (SELECT 1
  9 FROM Pret p
 10 WHERE p.compte = REF(c));

9 rows updated.
```

Figure 20: mise à jour les collections de prêts pour les comptes

```
NUMCOMPTE DATEOUVER DATECOMPT ETATCOMPTE SOLDE
-----
OPERATIONS
-----
PRETS
-----
CLIENT
-----
AGENCE
-----
TSET_REF_PRETS()
0000220208D160316E5BB940C9AA4F2669FE0B3445D9F5A2EBBB89436599C3829AEE05B724
0000220208F4B9817E03644AC1AE4ECAB28888A8643C65FE7D06444020BEF1FC7965F4C822
```

Figure 21: resultat representant la mise à jour les collections de prêts pour les comptes

```
SQL> -- mise à jour les collections de compte pour les agence
SQL> UPDATE Agence a
  2 SET a.comptes = (SELECT CAST(MULTISET(
  3 SELECT REF(c)
  4 FROM compte c
  5 WHERE c.agence = REF(a)
  6 ) AS tset_ref_comptes)
  7 FROM dual)
  8 WHERE EXISTS (SELECT 1
  9 FROM Compte c
 10 WHERE c.agence = REF(a));

18 rows updated.
```

Figure 22: mise à jour les collections de compte pour les agence

```

CATEGORIE
-----
SUCCURSALE
-----
COMPTES
-----
Secondaire
000022020830DA52C1895748079C299AAD5206A66EE05AE1F63AF34B19B5950482631B7AE7
TSET_REF_COMPTES(0000220208A8F3773E3B7547509103DCBB01E74BE3095F6187DCCE4B5BB2A87

```

Figure 23: resultat epresentant la mise à jour les collections de compte pour les agence

## 2.5 Langage d'interrogation de données

- Liste des comptes d'une agence donnée, dont les propriétaires sont des entreprises

```

SQL> select value(c).numCompte
2  from agence a , table (a.comptes) c
3  where a.numAgence = 101 and deref(value(c).client).typeClient='Entreprise';

VALUE(C).NUMCOMPTE
-----
1012836687
1013040367

```

Figure 24: Liste des comptes d'une agence donnée, dont les propriétaires sont des entreprises

- Les prêts effectués auprès des agences rattachées à une succursale donnée

```

SQL> select value(p).numPret,a.numAgence, value(c).numCompte , value(p).montantPret
2  from agence a ,table(a.comptes) c ,table (value(c).prets ) p
3  where deref(a.succursale).numSucc=005;

no rows selected

SQL> select value(p).numPret,a.numAgence, value(c).numCompte , value(p).montantPret
2  from agence a ,table(a.comptes) c ,table (value(c).prets ) p
3  where deref(a.succursale).numSucc=003;

VALUE(P).NUMPRET  NUMAGENCE  VALUE(C).NUMCOMPTE  VALUE(P).MONTANTPRET
-----
4013759          304          3042368831          3272386
8158803          304          3042368831          3468421
6777980          303          3038929107          2599546
4251682          301          3012519800          1580738
7143341          301          3012519800          1905605

```

Figure 25: Les prêts effectués auprès des agences rattachées à une succursale donnée

Il semble que l'absence de prêts auprès des agences rattachées à la succursale numéro 005 soit attribuable à l'insertion aléatoire des clients, qui a pu ne pas inclure ceux affiliés à ces agences.

### – Sélection des colonnes :

- \* `value(p).numPret`: Récupère le numéro du prêt à partir de la table de prêts (stockée dans `value(p)`).
- \* `a.numAgence`: Récupère le numéro d'agence de la table d'agence.
- \* `value(c).numCompte`: Récupère le numéro de compte de la table de comptes (stockée dans `value(c)`).

- \* `value(p).montantPret`: Récupère le montant du prêt à partir de la table de prêts (stockée dans `value(p)`).
- **Tables utilisées :**
  - \* `agence a`: Utilise la table "agence" et utilise "a" comme alias pour cette table.
  - \* `table(a.comptes) c`: Utilise la table "comptes" dans chaque agence (stockée dans `a.comptes`). La fonction `table()` est utilisée pour débiller les collections stockées dans la table.
  - \* `table(value(c).prets) p`: Utilise la table "prets" dans chaque compte (stockée dans `value(c).prets`). Encore une fois, la fonction `table()` est utilisée pour débiller les collections stockées dans la table.
- **Clause WHERE :**
  - \* `deref(a.succursale)`: Cette partie déréférence la colonne "succursale" dans la table "agence", ce qui signifie qu'elle accède aux données de la référence de succursale.
  - \* `.numSucc`: Cette partie extrait le numéro de succursale de la succursale déréférée.
  - \* `:numSucc`: Cela semble être un paramètre ou une variable. La requête sélectionne uniquement les données où le numéro de succursale correspond à la valeur fournie dans cette variable ou paramètre.
- **liste des comptes sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022**

```

NUMCOMPTE
-----
3034153976
5028330563
1014328117
4014573117
1038314426
3041670831
2048305867
2027848156
4024249417
3037619559
3038929107

NUMCOMPTE
-----
2036446350
3038940788

46 rows selected.
```

Figure 26: Liste des comptes sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022

- montant total des crédits effectués sur un compte en 2023

```
SQL> SELECT SUM(deref(VALUE(op)).montantOp)
2 FROM COMPTE C, TABLE(C.OPERATIONS) op
3 WHERE numCompte = 1180005564
4 AND EXTRACT(YEAR FROM deref(VALUE(op)).dateOp) = 2023
5 AND deref(VALUE(op)).natureOp = 'Credit';

SUM(DEREF(VALUE(OP)).MONTANTOP)
-----

SQL> SELECT SUM(deref(VALUE(op)).montantOp)
2 FROM COMPTE C, TABLE(C.OPERATIONS) op
3 WHERE numCompte = 3023540212
4 AND EXTRACT(YEAR FROM deref(VALUE(op)).dateOp) = 2023
5 AND deref(VALUE(op)).natureOp = 'Credit';

SUM(DEREF(VALUE(OP)).MONTANTOP)
-----
38515
```

Figure 27: montant total des crédits effectués sur un compte en 2023

Explication de la requête :

1. `SELECT SUM(deref(VALUE(op)).montantOp)` : Cette partie de la requête sélectionne la somme des montants (`montantOp`) des opérations de crédit.
2. `FROM COMPTE C, TABLE(C.OPERATIONS) op` : Cette partie spécifie les tables à partir desquelles les données sont extraites. `COMPTE` est la table principale, tandis que `OPERATIONS` est une table imbriquée dans la table `COMPTE`. `op` est l'alias de la table `OPERATIONS`.
3. `WHERE numCompte = :numcompte` : Cela filtre les données en ne considérant que le compte spécifié par la variable `:numcompte`.
4. `AND EXTRACT(YEAR FROM deref(VALUE(op)).dateOp) = 2023` : Cela extrait l'année de la date de chaque opération (`dateOp`) et ne sélectionne que celles qui ont eu lieu en 2023.
5. `AND deref(VALUE(op)).natureOp = 'Credit'` : Ceci filtre davantage les opérations pour ne considérer que celles de nature "Crédit".

- Les prêts non encore soldés à ce jour



```

SQL> SELECT p.numPret,
2      deref(deref(p.compte).agence).numAgence AS numAgence,
3      deref(p.compte).numCompte AS numCompte,
4      deref(deref(p.compte).client).numClient AS numClient,
5      p.montantPret - COALESCE((SELECT SUM(op.montantOp)
6                                FROM Operation op
7                                WHERE op.natureOp = 'Debit'
8                                AND op.compte = p.compte), 0) AS montantRestant
9  FROM Pret p
10 WHERE (p.dateEffet + p.duree <= TRUNC(SYSDATE))
11       AND (p.montantPret - COALESCE((SELECT SUM(op.montantOp)
12                                       FROM Operation op
13                                       WHERE op.natureOp = 'Debit'
14                                       AND op.compte = p.compte), 0)) > 0;

```

NUMPRET	NUMAGENCE	NUMCOMPTE	NUMCLIENT	MONTANTRESTANT
4251682	301	3012519800	10003	1580738
7143341	301	3012519800	10003	1905605
4013759	304	3042368831	10012	3272386
8158803	304	3042368831	10012	3468421
3021562	103	1038314426	10018	1136733
5294804	103	1038314426	10018	4824336
1009450	103	1038314426	10018	3238563
6423816	101	1014328117	10030	4506496
7859499	401	4019296834	10039	2928409
1138737	401	4019296834	10039	3132083
5908528	201	2011943424	10042	2582761
6777980	303	3038929107	10051	2599546
4408744	204	2046036983	10066	579060
6701677	204	2046036983	10066	3794942
5510912	202	2024041675	10087	3852571
4675005	202	2024041675	10087	4037406

16 rows selected.

Figure 28: les prêts non encore soldés à ce jour

- **SELECT p.numPret**: Sélectionne le numéro du prêt.
- **deref(deref(p.compte).agence).numAgence AS numAgence**: Récupère le numéro de l'agence à laquelle le compte est associé.
- **deref(p.compte).numCompte AS numCompte**: Récupère le numéro de compte.
- **deref(deref(p.compte).client).numClient AS numClient**: Récupère le numéro du client associé au compte.
- **p.montantPret - COALESCE((SELECT SUM(op.montantOp) FROM Operation op WHERE op.natureOp = 'Debit' AND op.compte = p.compte), 0) AS montantRestant**: Calcule le montant restant dû en soustrayant la somme des débits effectués sur le compte du montant initial du prêt.
- **FROM Pret p**: Spécifie la table des prêts à partir de laquelle les données sont sélectionnées.
- **WHERE ...**: Conditions pour filtrer les prêts : vérifie si la période de remboursement est terminée et si le montant restant dû est supérieur à zéro.