Rapport Project BDA

Lydia Benzemrane
Rahil Ghamraci

Introduction

Dans le domaine du traitement des données, les systèmes de gestion de bases de données (SGBD) jouent un rôle essentiel en offrant des solutions pour le stockage, la manipulation et la récupération efficace de l'information. Dans ce rapport de projet, nous explorerons deux approches distinctes de gestion des données : le SQL3 Relationnel-Objet et le modèle orienté "documents" NoSQL.

Partie I : SQL3 Relationnel-Objet

Le SQL3, ou Structured Query Language version 3, représente une évolution majeure dans le domaine des bases de données relationnelles. Cette norme introduit des fonctionnalités avancées permettant de manipuler efficacement les données tout en maintenant la structure relationnelle. L'intégration d'éléments objet dans le modèle relationnel offre une flexibilité accrue, permettant la représentation de données complexes. Dans cette partie, nous examinerons en détail les concepts fondamentaux du SQL3 Relationnel-Objet, ses fonctionnalités avancées et ses applications pratiques.

Partie II : NoSQL – Modèle orienté "documents"

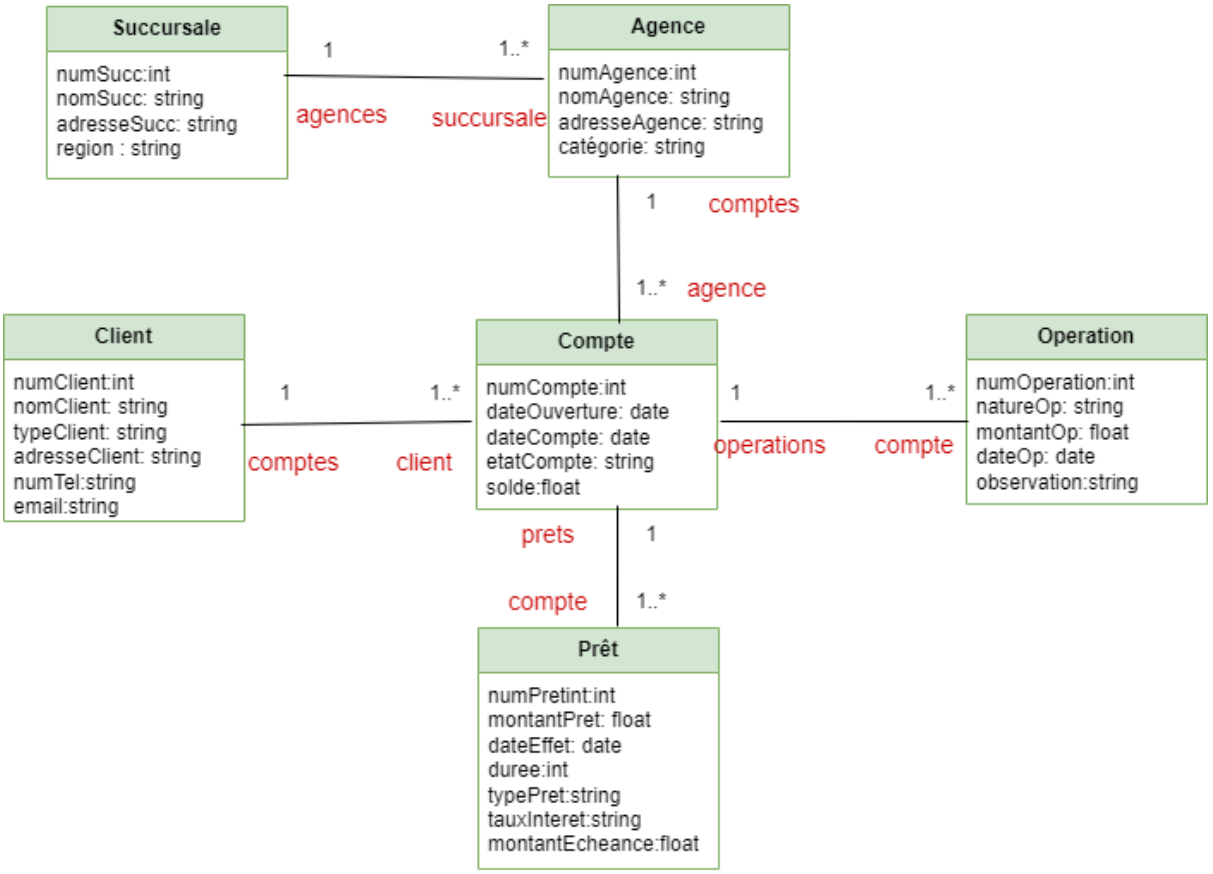
Le modèle NoSQL, contrairement aux bases de données relationnelles traditionnelles, adopte une approche plus flexible et distribuée pour le stockage et la manipulation des données. Parmi les différentes approches NoSQL, le modèle orienté "documents" se distingue par sa capacité à stocker des données semi-structurées sous forme de documents, généralement au format JSON ou XML. Cette approche permet une scalabilité horizontale facile et convient particulièrement aux applications nécessitant une gestion de données dynamique et évolutive. Dans cette partie, nous explorerons les principes fondamentaux du modèle orienté "documents" NoSQL, ses avantages et ses limitations, ainsi que ses cas d'utilisation les plus pertinents.

En combinant l'analyse approfondie de ces deux approches, ce rapport vise à fournir un aperçu complet des technologies de gestion de bases de données modernes.

Partie I : Relationnel-Objet

A-Modélisation orientée objet

Pour la modélisation du schéma relationnel donnée, nous avons utilisé le diagramme de classes d'uml



Et nous avons défini les rôles suivants :

le rôle *agences* pour la classe succursale qui sera une collection de références vers des objets de type agence.

le rôle *succursale* pour la classe agence qui sera une seule référence vers un objet de type succursale.

le rôle *comptes* pour la classe agence qui sera une collection de références vers des objets de type compte.

le rôle *agence* pour la classe compte qui sera une une seule référence vers un objet de type agence.

le rôle *comptes* pour la classe client qui sera une collection de références vers des objets de type compte.

le rôle *client* pour la classe compte qui sera une une seule référence vers un objet de type client.

le rôle *opérations* pour la classe compte qui sera une collection de références vers des objets de type opération.

le rôle *compte* pour la classe opération qui sera une une seule référence vers un objet de type compte.

le rôle *prêts* pour la classe compte qui sera une collection de références vers des objets de type prêt.

le rôle *compte* pour la classe prêt qui sera une une seule référence vers un objet de type compte.

B- Création des TableSpaces et utilisateur

connection à une base de données pluggable

```
SQL> ALTER PLUGGABLE DATABASE orclpdb OPEN;

Base de donnÚes pluggable modifiÚe.

SQL> connect sys@orclpdb as sysdba
Entrez le mot de passe :
ConnectÚ.
```

2. Création des deux TableSpaces SQL3_TBS et SQL3_TempTBS

```
SQL> create tablespace  SQL3_TBS datafile 'C:\tablespaces\tbs_002.dat' size 100M autoextend on;

Tablespace crÚÚ.

SQL> create temporary tablespace SQL3_TempTBS  tempfile 'C:\tablespaces\temp_002.dat' size 100M autoextend on;

Tablespace crÚÚ.
```

3. Création de l'utilisateur SQL3 en lui attribuant les deux tableSpaces créés précédemment

```
SQL> create user  SQL3 identified by sql3 default tablespace SQL3_TBS temporary tablespace SQL3_TempTBS;

Utilisateur crÚÚ.
```

4. Donner tous les privilèges à cet utilisateur.

```
SQL> grant all privileges to SQL3;

Autorisation de privilPges (GRANT) acceptÚe.
```

connection avec les informations de l'utilisateur créé

```
SQL> connect SQL3/sql3@orclpdb;
ConnectÚ.
```

C- Langage de définition de données

5. Définition de tous les types abstraits nécessaires et définition de toutes les associations qui existent

Création des types incomplets:

```
SQL> create type tsuccursale;
  2  /

Type crÚÚ.

SQL> create type tagence;
  2  /

Type crÚÚ.

SQL> create type tclient;
  2  /

Type crÚÚ.

SQL> create type tcompte
  2  /

Type crÚÚ.

SQL> create type toperation;
  2  /

Type crÚÚ.

SQL> create type tpret;
  2  /
```

création des types nécessaires aux associations "les tables imbriquées des références"

```
SQL> Create type tset_ref_agences as table of ref tagence;
  2  /

Type crÚÚ.

SQL> Create type tset_ref_comptes as table of ref tcompte;
  2  /

Type crÚÚ.

SQL> Create type tset_ref_operation as table of ref toperation;
  2  /

Type crÚÚ.

SQL> Create type tset_ref_prets as table of ref tpret;
  2  /

Type crÚÚ.
```

création des types

```
SQL> CREATE OR REPLACE TYPE tsuccursale AS OBJECT (
  2      numSucc NUMBER(3),
  3      nomSucc VARCHAR(30),
  4      adresseSucc VARCHAR(30),
  5      region VARCHAR(30),
  6      agences tset_ref_agences
  7  );
  8  /

Type crÚÚ.
```

```
SQL> CREATE OR REPLACE TYPE tsuccursale AS OBJECT (
  2      numSucc NUMBER(3),
  3      nomSucc VARCHAR(30),
  4      adresseSucc VARCHAR(30),
  5      region VARCHAR(30),
  6      agences tset_ref_agences
  7  );
  8  /

Type crÚÚ.

SQL> create or replace type tagence as object (
  2      numAgence NUMBER(3),
  3      nomAgence VARCHAR(30),
  4      adresseAgence VARCHAR(30),
  5      categorie VARCHAR(30),
  6      succursale ref tsuccursale,
  7  comptes tset_ref_comptes
  8  );
  9  /

Type crÚÚ.
```

```
SQL> create or replace type tclient as object (
  2      numClient NUMBER(5),
  3      nomClient VARCHAR(30),
  4  typeClient VARCHAR(30),
  5      adresseClient VARCHAR(30),
  6  numTel VARCHAR(100),
  7      email VARCHAR(30),
  8      comptes tset_ref_comptes
  9  );
 10  /

Type crÚÚ.

SQL> create or replace type tcompte as object (
  2      numCompte NUMBER(10),
  3      dateOuverture date,
  4  etatCompte VARCHAR(30),
  5      solde float,
  6  client ref tclient,
  7      agence ref tagence,
  8  operations tset_ref_operation,
  9  prets tset_ref_prets
 10  );
 11  /

Type crÚÚ.
```

```
SQL> create or replace type toperation as object (
  2       numOperation NUMBER(10),
  3       natureOp VARCHAR(30),
  4    montantOp float,
  5       dateOp date,
  6    observation VARCHAR(50),
  7       compte ref tcompte
  8    );
  9    /

Type crÚÚ.

SQL> create or replace type tpret as object (
  2       numPret NUMBER(10),
  3       montantPret float,
  4    dateEffet date,
  5       duree INTEGER,
  6    typePret VARCHAR(50),
  7    tauxInteret VARCHAR(50),
  8    montantEcheance float,
  9       compte ref tcompte
 10    );
 11    /

Type crÚÚ.
```

6. Définition des méthodes :

définition de la signature de la méthode qui calcule le nombre de prêts effectuées pour chaque agence.

```
SQL> alter type tagence add member function calcul_prets return Number cascade;

Type modifiÚ.
```

définition de corp de la méthode calcul_prets

```
SQL> CREATE OR REPLACE TYPE BODY tagence AS
  2     MEMBER FUNCTION calcul_prets RETURN NUMBER IS
  3       nb NUMBER := 0;
  4    compte_ref ref tcompte;
  5    pret_ref ref tpret;
  6     BEGIN
  7       select count(*) into nb from table( SELECT value(c).prets FROM TABLE(self.comptes) c) p ;
  8       RETURN nb;
  9     END calcul_prets;
 10    END;
 11    /

Corps de type crÚÚ.
```

définition de la signature de la méthode qui calcule pour une agence , le montant global des prêts effectués durant la période du 01-01-2020 au 01-01-2024.

```
SQL> alter type tagence add member function montant_global return Number cascade;

Type modifiÚ.
```

définition de corp de la méthode montant_global

```
SQL> CREATE OR REPLACE TYPE BODY tagence AS
  2    MEMBER FUNCTION calcul_prets RETURN NUMBER IS
  3      nb NUMBER := 0;
  4    BEGIN
  5     select count(*) into nb from table( SELECT value(c).prets FROM TABLE(self.comptes) c) p ;
  6      RETURN nb;
  7    END calcul_prets;
  8
  9    MEMBER FUNCTION montant_global RETURN NUMBER IS
 10      nb NUMBER := 0;
 11    BEGIN
 12       select sum(value(p).montantPret) into nb from table( SELECT value(c).prets FROM TABLE(self.comptes) c) p
 13      where  value(p).dateEffet >= TO_DATE('01-01-2020', 'DD-MM-YYYY')
 14      AND value(p).dateEffet + value(p).duree <= TO_DATE('01-01-2024', 'DD-MM-YYYY');
 15      RETURN nb;
 16    END montant_global;
 17  END;
 18  /

Corps de type crÚÚ.
```

définition de la signature de la méthode qui calcule pour chaque succursale, le nombre d’agences principales qui lui sont rattachées.

```
SQL> alter type tsuccursale add member function calcul_nb_agences return Number cascade;

Type modifiÚ.
```

définition de corp de la méthode calcul_nb_agences

```
MEMBER FUNCTION calcul_nb_agences RETURN NUMBER IS
  nb NUMBER;
BEGIN
  SELECT COUNT(*) INTO nb
  FROM  TABLE(self.agences) a
  where VALUE(a).categorie = 'Principale';

  RETURN nb;
END calcul_nb_agences;
```

définition des types nécessaire pour la méthode qui liste toutes les agences secondaires (avec la succursale rattachée) ayant au moins un prêt
« ANSEJ »

le type tagencesucc qui contient le nom de l'agence avec le nom de la succursale

```
SQL> create or replace type tagencesucc as object (
  2       nomAgence VARCHAR(50),
  3       nomSucc VARCHAR(50)
  4  );
  5  /

Type crÚÚ.
```

le type testagence qui est une table d'objets tagencesucc , cette table va contenir le résultat de la méthode

```
SQL> create type tsetAgence as table of tagencesucc;
  2  /

Type crÚÚ.
```

définition de la signature de la méthode qui liste toutes agences secondaires (avec la succursale rattachée) ayant au moins un prêt « ANSEJ »

```
SQL> alter type tsuccursale add member function agences_liste return tsetAgence cascade;

Type modifiÚ.
```

définition de corp de la méthode agences_liste

```
4       RETURN nb;
5     END calcul_nb_agences;
6
7       MEMBER FUNCTION agences_liste RETURN tsetAgence  IS
8       agences_secondaires tsetAgence ;
9       nb NUMBER;
0     BEGIN
1       Select CAST(MULTISET( select deref(value(a)).nomAgence,  self.nomSucc
2                         from TABLE(self.agences) a
3                         where deref(VALUE(a)).categorie = 'Secondaire'
4     AND EXISTS (
5                         SELECT 1
6                         FROM TABLE( SELECT value(c).prets FROM TABLE(VALUE(a).comptes) c) p
7                         WHERE value(p).typePret = 'ANSEJ'
8                          )
9     ) as tsetAgence
0      ) into agences_secondaires
1       from dual;
2       RETURN agences_secondaires;
3     END agences_liste;
4
5   END;
6   /

rps de type crÚÚ.
```

7. Définitions des tables nécessaires à la base de données.

Pour chaque table nous avons défini la contrainte de la clé primaire ainsi que les contraintes de check nécessaire pour les attributs de la table.

```
SQL>  CREATE TABLE succursales OF tsuccursale (
  2      constraint pk_succursale primary key(numSucc),
  3      constraint region_check CHECK (region IN ('Sud', 'Nord', 'Est', 'Ouest'))
  4  )
  5   nested table agences store as tableAgence;

Table crÚÚe.

SQL> CREATE TABLE agences OF tagence (
  2      constraint pk_agence primary key(numAgence),
  3      CONSTRAINT categorie_check CHECK (categorie IN ('Principale', 'Secondaire'))
  4  )
  5   nested table comptes store as tableComptesAgence;

Table crÚÚe.

SQL>
SQL>
SQL> CREATE TABLE clients OF tclient (
  2      constraint pk_client primary key(numClient),
  3      CONSTRAINT type_client_check CHECK (typeClient IN ('Particulier', 'Entreprise'))
  4  )
  5   nested table comptes store as tableComptesClient;

Table crÚÚe.
```

```
SQL>
SQL> CREATE TABLE comptes OF tcompte (
  2      constraint pk_compte primary key(numCompte),
  3      CONSTRAINT etat_check CHECK (etatCompte IN ('Actif', 'Bloque'))
  4  )
  5   nested table operations store as tableOperations,
  6    nested table prets store as tablePrets;

Table crÚÚe.

SQL>
SQL>
SQL> CREATE TABLE operations OF toperation (
  2      constraint pk_operation primary key(numOperation),
  3      CONSTRAINT nature_op_check CHECK (natureOp IN ('Credit', 'Debit'))
  4  );

Table crÚÚe.

SQL>
SQL>
SQL> CREATE TABLE prets OF tpret (
  2      constraint pk_pret primary key(numPret),
  3      CONSTRAINT type_pret_check CHECK (typePret IN ('Vehicule', 'Immobilier','ANSEJ', 'ANJEM'))
  4  );

Table crÚÚe.
```

D-Création des instances dans les tables

Insertion des instances dans la table succursales

Requête :

```
INSERT INTO succursales VALUES (001, 'Succursale Sud', 'Rue des Oliviers, VilleA',
'Sud', tset_ref_agences());
```

Résultat d'exécution :

```
SQL> INSERT INTO succursales VALUES ('001', 'Succursale Sud', 'Rue des Oliviers, VilleA', 'Sud', tset_ref_agences());

1 ligne crÚÚe.

SQL> INSERT INTO succursales VALUES ('002', 'Succursale Nord', 'Avenue des Pins, VilleB', 'Nord', tset_ref_agences());

1 ligne crÚÚe.

SQL> INSERT INTO succursales VALUES ('003', 'Succursale Est', 'Boulevard des Roses, VilleC', 'Est', tset_ref_agences());


1 ligne crÚÚe.

SQL> INSERT INTO succursales VALUES ('004', 'Succursale Ouest', 'Chemin des Chênes, VilleD', 'Ouest', tset_ref_agences()
);

1 ligne crÚÚe.

SQL> INSERT INTO succursales VALUES ('005', 'Succursale Centrale', 'Place du Marché, VilleE', 'Sud', tset_ref_agences())
;

1 ligne crÚÚe.

SQL> INSERT INTO succursales VALUES ('006', 'Succursale Principale', 'Rue Principale, VilleF', 'Nord', tset_ref_agences(
));

1 ligne crÚÚe.
```

Insertion des instances dans la table agences

Requête :

```
INSERT INTO agences VALUES (101, 'Agence Principale 1', 'Rue de la Liberté,
VilleA', 'Principale', (SELECT REF(s) FROM succursales s WHERE s.numSucc = 001),
tset_ref_comptes());
```

Résultat d'exécution :

```
SQL>
SQL> -- Agence 023
SQL> INSERT INTO agences VALUES (123, 'Agence Secondaire 23', 'Place des Arts, VilleE', 'Secondaire', (SELECT REF(s) FRO
M succursales s WHERE s.numSucc = 005), tset_ref_comptes());

1 ligne crÚÚe.

SQL>
SQL> -- Agence 024
SQL> INSERT INTO agences VALUES (124, 'Agence Principale 124', 'Rue Saint-Michel, VilleF', 'Principale', (SELECT REF(s)
FROM succursales s WHERE s.numSucc = 006), tset_ref_comptes());

1 ligne crÚÚe.

SQL>
SQL> -- Agence 025
SQL> INSERT INTO agences VALUES (125, 'Agence Principale 125', 'Rue Saint-Michel, VilleF', 'Principale', (SELECT REF(s)
FROM succursales s WHERE s.numSucc = 001), tset_ref_comptes());

1 ligne crÚÚe.

SQL>
```

Maintenant que les agences ont été insérées , nous devons mettre à jour l'attribut agences dans la table succursale qui est une collection de références vers des objets de type agence

En utilisant la requête:

```
insert into table (select s.agences from succursales s where numSucc=001)
(select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s
where numSucc=001));
```

`(select s.agences from succursales s where numSucc=001)`....est utilisée pour spécifier la table où nous allons insérer nos références vers les objets agences.

`(select ref(s) from succursales s where numSucc=001)`....est utilisée pour récupérer la référence vers la succursale numéro 001

`(select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=001));`...est utilisée pour récupérer les références des agences qui ont comme succursale la succursale numéro 001

Résultat d'exécution :

```
SQL> insert into table (select s.agences from succursales s where numSucc=002)
  2  (select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=002));

4 lignes crÚÚes.

SQL>
SQL> insert into table (select s.agences from succursales s where numSucc=003)
  2  (select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=003));

4 lignes crÚÚes.

SQL>
SQL> insert into table (select s.agences from succursales s where numSucc=004)
  2  (select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=004));

4 lignes crÚÚes.

SQL>
SQL> insert into table (select s.agences from succursales s where numSucc=005)
  2  (select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=005));

4 lignes crÚÚes.

SQL>
SQL> insert into table (select s.agences from succursales s where numSucc=006)
  2  (select ref(a) from agences a where  a.succursale=(select ref(s) from succursales s where numSucc=006));

4 lignes crÚÚes.
```

Insertion des instances dans la table clients en utilisant un script

```
SQL> -- Insertion de 100 clients
SQL> BEGIN
  2      FOR i IN 1..100 LOOP
  3          INSERT INTO clients (
  4              numClient,
  5              nomClient,
  6              typeClient,
  7              adresseClient,
  8              numTel,
  9              email,
 10              comptes
 11          ) VALUES (
 12              LPAD(i, 5, '0'),
 13              'Client_' || LPAD(i, 5, '0'),
 14              CASE MOD(i, 2) WHEN 0 THEN 'Particulier' ELSE 'Entreprise' END,
 15              'Adresse_Client_' || LPAD(i, 5, '0'),
 16              '+123456789',
 17              'client' || LPAD(i, 3, '0') || '@example.com',
 18              tset_ref_comptes() -- Crée un ensemble vide de références de comptes
 19          );
 20      END LOOP;
 21  END;
 22  /

ProcÚdure PL/SQL terminÚe avec succPs.
```

Insertion des instances dans la table comptes

Requête:

```
INSERT INTO comptes
VALUES (
    tcompte(1010000001, SYSDATE - 1, 'Actif', 5000.00, (SELECT REF(c) FROM clients
c WHERE c.numClient = 1), (SELECT REF(a) FROM agences a WHERE a.numAgence = 101),
tset_ref_operation(), tset_ref_prets())
);
```

Une fois que les comptes ont été insérées , nous devons mettre à jour l'attribut comptes dans la table client qui est une collection de références vers des objets de type compte

En utilisant la requête:

```
insert into table (select c.comptes from clients c where c.numClient=1)
(select ref(c1) from comptes c1 where  c1.client=(select ref(c) from clients c
where numClient=1));
```

`(select c.comptes from clients c where c.numClient=1)` ....est utilisée pour spécifier la table où nous allons insérer nos références vers les objets comptes.

`(select ref(c) from clients c where numClient=1)` ....est utilisée pour récupérer la référence vers le client numéro 001

`(select ref(c1) from comptes c1 where  c1.client=(select ref(c) from clients c where numClient=1));...` est utilisée pour récupérer les références des comptes qui ont comme client , le client numéro 001

Résultat d'exécution :

```
SQL>
SQL> insert into table (select a.comptes from agences a where a.numAgence=124)
  2  (select ref(c) from comptes c where  c.agence=(select ref(a) from agences a where a.numAgence=124));

4 lignes crÚÚes.

SQL>
SQL> insert into table (select a.comptes from agences a where a.numAgence=125)
  2  (select ref(c) from comptes c where  c.agence=(select ref(a) from agences a where a.numAgence=125));

4 lignes crÚÚes.
```

Insertion des instances dans la table opération

Requête:

```
INSERT INTO operations
VALUES (
    toperation(10000001, 'Credit', 500.00, SYSDATE - 1, 'Paiement', (SELECT REF(c)
FROM comptes c WHERE c.numCompte = 1010000001))
);
```

Résultat d'exécution :

```
SQL> INSERT INTO operations
  2  VALUES (
  3      toperation(10000039, 'Credit', 500.00, SYSDATE - 1, 'Paiement', (SELECT REF(c) FROM comptes c WHERE c.numCompte
 = 1190000019))
  4  );

1 ligne crÚÚe.

SQL>
SQL> INSERT INTO operations
  2  VALUES (
  3      toperation(100000040, 'Debit', 200.00, SYSDATE - 2, 'Retrait', (SELECT REF(c) FROM comptes c WHERE c.numCompte
= 1200000020))
  4  );

1 ligne crÚÚe.

SQL> select count(*) from operations;

  COUNT(*)
----------
        40
```

Une fois que les opérations ont été insérées , nous devons mettre à jour l'attribut opérations dans la table compte qui est une collection de références vers des objets de type opération

En utilisant la requête:

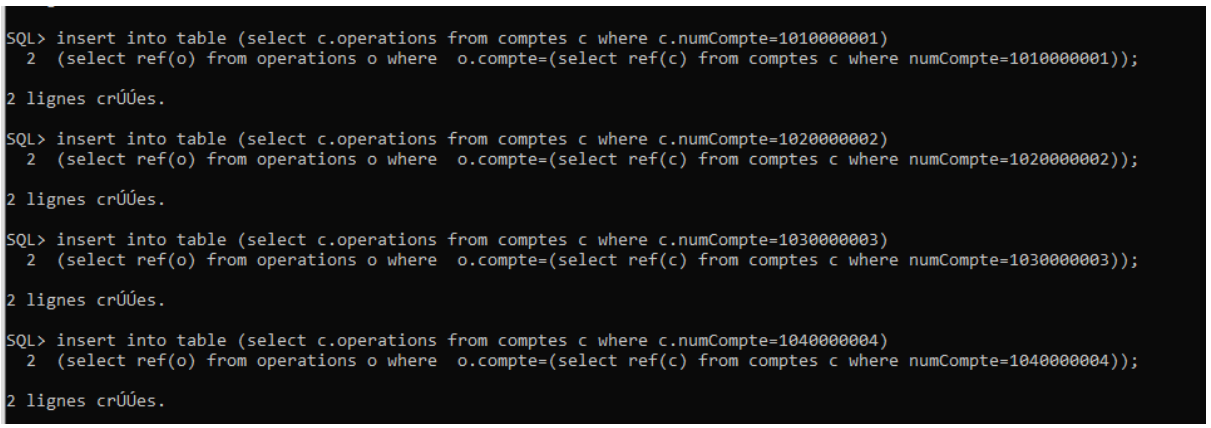
```
insert into table (select c.operations from comptes c where c.numCompte=1010000001)
(select ref(o) from operations o where  o.compte=(select ref(c) from comptes c
where numCompte=1010000001));
```

`(select c.operations from comptes c where c.numCompte=1010000001)`...est utilisée pour spécifier la table où nous allons insérer nos références vers les objets opérations.

`(select ref(c) from comptes c where numCompte=1010000001)`....est utilisée pour récupérer la référence vers le compte numéro 1010000001

`(select ref(o) from operations o where  o.compte=(select ref(c) from comptes c where numCompte=1010000001));`...est utilisée pour récupérer les références des opérations qui ont comme compte , le compte numéro 1010000001

Résultat d'exécution :



```
SQL> insert into table (select c.operations from comptes c where c.numCompte=1010000001)
  2  (select ref(o) from operations o where  o.compte=(select ref(c) from comptes c where numCompte=1010000001));

2 lignes crÚÚes.

SQL> insert into table (select c.operations from comptes c where c.numCompte=1020000002)
  2  (select ref(o) from operations o where  o.compte=(select ref(c) from comptes c where numCompte=1020000002));

2 lignes crÚÚes.

SQL> insert into table (select c.operations from comptes c where c.numCompte=1030000003)
  2  (select ref(o) from operations o where  o.compte=(select ref(c) from comptes c where numCompte=1030000003));

2 lignes crÚÚes.

SQL> insert into table (select c.operations from comptes c where c.numCompte=1040000004)
  2  (select ref(o) from operations o where  o.compte=(select ref(c) from comptes c where numCompte=1040000004));

2 lignes crÚÚes.
```

Insertion des instances dans la table prêts

Requête:

```
INSERT INTO prets
VALUES (
    tpret(1, 5000.00, SYSDATE, 24, 'Vehicule', '5%', 250.00, (SELECT REF(c) FROM
comptes c WHERE c.numCompte = 1010000001))
);
```

Résultat d'exécution :

```
SQL> INSERT INTO prets
  2  VALUES (
  3      tpret(9, 5000.00, SYSDATE, 24, 'Vehicule', '5%', 250.00, (SELECT REF(c) FROM comptes c WHERE c.numCompte = 1090
000009))
  4  );

1 ligne crÚÚe.

SQL>
SQL> INSERT INTO prets
  2  VALUES (
  3      tpret(10, 10000.00, SYSDATE, 36, 'Immobilier', '6%', 300.00, (SELECT REF(c) FROM comptes c WHERE c.numCompte =
1100000010))
  4  );

1 ligne crÚÚe.

SQL>
```

Une fois que les prêts ont été insérées , nous devons mettre à jour l'attribut prêts dans la table compte qui est une collection de références vers des objets de type prêt.

En utilisant la requête:

```
insert into table (select c.prets from comptes c where c.numCompte=1010000001)
(select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where
numCompte=1010000001));
```

`(select c.prets from comptes c where c.numCompte=1010000001)`...est utilisée pour spécifier la table où nous allons insérer nos références vers les objets prets.

`(select ref(c) from comptes c where numCompte=1010000001)`....est utilisée pour récupérer la référence vers le compte numéro 1010000001

`(select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where numCompte=1010000001));`..est utilisée pour récupérer les références des prets qui ont comme compte , le compte numéro 1010000001

Résultat d'exécution :

```
SQL>
SQL> insert into table (select c.prets from comptes c where c.numCompte=1080000008)
  2  (select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where numCompte=1080000008));

1 ligne crÚÚe.

SQL>
SQL> insert into table (select c.prets from comptes c where c.numCompte=1090000009)
  2  (select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where numCompte=1090000009));

1 ligne crÚÚe.

SQL>
SQL> insert into table (select c.prets from comptes c where c.numCompte=1100000010)
  2  (select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where numCompte=1100000010));
```

E- Langage d'interrogation de données

9. La liste de tous les comptes d'une agence donnée, dont les propriétaires sont des entreprises.

```
select value(c).numCompte
from  agences a , table (a.comptes) c
where a.numAgence = 125 and  deref(value(c).client).typeClient='Entreprise';
```

`table (a.comptes) c` récupère la table de référence vers les comptes de l'agence
`value(c)` est une seule valeur de cette table de références
`value(c).client` est la référence vers le client
`deref(value(c).client)` retourne l'objet de type tclient représenté par la référence récupérée

```
SQL> select value(c).numCompte
  2  from  agences a , table (a.comptes) c
  3  where a.numAgence = 125 and  deref(value(c).client).typeClient='Entreprise';

VALUE(C).NUMCOMPTE
------------------
        1250000025
        1250000075
        1250000099
```

10. La liste des prêts effectués auprès des agences rattachées à une succursale donnée.

```
select value(p).numPret,a.numAgence, value(c).numCompte , value(p).montantPret
from  agences a ,table(a.comptes) c ,table (value(c).prets )  p
where deref(a.succursale).numSucc=005;
```

`table(a.comptes) c` récupère la table de référence vers les comptes de l'agence
`table (value(c).prets )  p` récupère la table de référence vers les prêts d'un compte
`deref(a.succursale)` retourne l'objet de type succursale associé à l'agence donnée

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

VALUE(P).NUMPRET  NUMAGENCE VALUE(C).NUMCOMPTE VALUE(P).MONTANTPRET
---------------- ---------- ------------------ --------------------
               5        105         1050000005                 5000
              11        111         1110000011                 5000
```

11.Les comptes sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022

```
SELECT c.numCompte
FROM comptes c
WHERE c.numCompte NOT IN (
    SELECT c1.numCompte
    FROM comptes c1 ,TABLE(c1.operations) o
    WHERE value(o).natureOp = 'Debit'
    AND EXTRACT(YEAR FROM value(o).dateOp) BETWEEN 2000 AND 2022
);
```

`TABLE(c1.operations) o` récupère la table de référence vers les opérations du compte c1

```
 SELECT c1.numCompte
    FROM comptes c1 ,TABLE(c1.operations) o
    WHERE value(o).natureOp = 'Debit'
    AND EXTRACT(YEAR FROM value(o).dateOp) BETWEEN 2000 AND 2022
```

récupère les comptes qui ont effectué des opération de débit entre 2000 et 2022 , nous avons éliminé ces compte par le mot clé **not in**

```
SQL> SELECT c.numCompte
  2  FROM comptes c
  3  WHERE c.numCompte NOT IN (
  4      SELECT c1.numCompte
  5      FROM comptes c1 ,TABLE(c1.operations) o
  6      WHERE value(o).natureOp = 'Debit'
  7      AND EXTRACT(YEAR FROM value(o).dateOp) BETWEEN 2000 AND 2022
  8  );

 NUMCOMPTE
----------
```

```
 NUMCOMPTE
----------
1250000050
1040000029
1040000079
1050000080
1060000031
1160000016
1160000041
1240000098
1080000033
1110000086
1150000100

99 lignes sÚlectionnÚes.
```

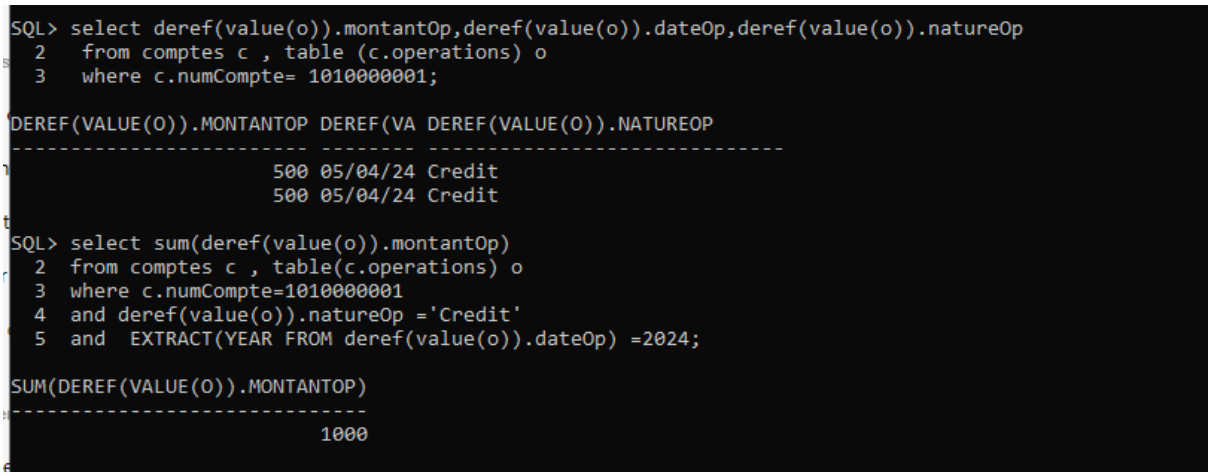
```
SQL> INSERT INTO operations
  2  VALUES (
  3      toperation(100000041, 'Debit', 200.00, TO_DATE('04/04/21'), 'Retrait', (SELECT REF(c) FROM comptes c WHERE c.nu
mCompte = 1200000020))
  4  );

1 ligne crÚÚe.
```

99 lignes retournées car le compte avec le numéro 1200000020 avait effectué une opération de débit en 2021

12. Le montant total des crédits effectués sur un compte en 2023

```
select sum(deref(value(o)).montantOp)
from comptes c , table(c.operations) o
where c.numCompte=1010000001
and deref(value(o)).natureOp ='Credit'
and  EXTRACT(YEAR FROM deref(value(o)).dateOp) =2024;
```

`table(c.operations) o` récupère la table de référence vers les opérations du compte c
`deref(value(o))` retourne l'objet de type toperation qui a la référence value(o)
`EXTRACT(YEAR FROM deref(value(o)).dateOp)` extraire l'année à partir de la date de l'opération



```
SQL> select deref(value(o)).montantOp,deref(value(o)).dateOp,deref(value(o)).natureOp
  2   from comptes c , table (c.operations) o
  3   where c.numCompte= 1010000001;

DEREF(VALUE(O)).MONTANTOP DEREF(VA DEREF(VALUE(O)).NATUREOP
------------------------- -------- ------------------------------
                      500 05/04/24 Credit
                      500 05/04/24 Credit

SQL> select sum(deref(value(o)).montantOp)
  2  from comptes c , table(c.operations) o
  3  where c.numCompte=1010000001
  4  and deref(value(o)).natureOp ='Credit'
  5  and  EXTRACT(YEAR FROM deref(value(o)).dateOp) =2024;

SUM(DEREF(VALUE(O)).MONTANTOP)
------------------------------
                          1000
```

13. Les prêts non encore soldés à ce jour .

```
select p.numPret,deref(deref(p.compte).agence).numAgence as numAgence,
deref(p.compte).numCompte as numCompte, deref(deref(p.compte).client).numClient as
numClient, p.montantPret
from prets p
WHERE (
    SELECT SUM(value(o).montantOp)
    FROM TABLE(p.compte.operations) o
    WHERE value(o).natureOp = 'Debit'
) < p.montantPret;
```

la sous requête: `SELECT SUM(value(o).montantOp)FROM TABLE(p.compte.operations) o WHERE value(o).natureOp = 'Debit'` retourne la somme des montants des opérations de débit effectuées sur le compte du prêt

`< p.montantPret;` vérifier si la somme calculée est inférieur au montant du prêt

`deref(deref(p.compte).agence)` retourne l'agence du compte du prêt
`deref(p.compte)` retourne le compte du prêt
`deref(deref(p.compte).client)` retourne le client du compte du prêt

```
SQL> select p.numPret,deref(deref(p.compte).agence).numAgence as numAgence, deref(p.compte).numCompte as numCompte, dere
f(deref(p.compte).client).numClient as numClient, p.montantPret
  2  from prets p
  3  WHERE (
  4      SELECT SUM(value(o).montantOp)
  5      FROM TABLE(p.compte.operations) o
  6      WHERE value(o).natureOp = 'Debit'
  7  ) < p.montantPret;

   NUMPRET  NUMAGENCE  NUMCOMPTE  NUMCLIENT MONTANTPRET
---------- ---------- ---------- ---------- -----------
         2        102 1020000002          2       10000
         4        104 1040000004          4       10000
         6        106 1060000006          6       10000
         8        108 1080000008          8       10000
        10        110 1100000010         10       10000
```

14. Le compte le plus mouvementé en 2023

```
SELECT c.numCompte
FROM comptes c , table(c.operations) o
WHERE EXTRACT(YEAR FROM value(o).dateOp) =2024
GROUP BY c.numCompte
HAVING COUNT(DISTINCT deref(value(o)).numOperation) = (
    SELECT MAX(operation_count)
    FROM (
        SELECT COUNT(DISTINCT deref(value(ops)).numOperation) AS operation_count
        FROM comptes cs , table(cs.operations) ops
             WHERE EXTRACT(YEAR FROM value(ops).dateOp) =2024
        GROUP BY cs.numCompte
    )
);
```

la sous requête:

```
        SELECT COUNT(DISTINCT deref(value(ops)).numOperation) AS operation_count
        FROM comptes cs , table(cs.operations) ops
             WHERE EXTRACT(YEAR FROM value(ops).dateOp) =2024
        GROUP BY cs.numCompte
```

compte le nombre d’opération effectuées par chaque compte

```
 SELECT MAX(operation_count)
    FROM (
       ……
    )
```

retourne le nombre maximale des opération effectuées par un seul compte

```
HAVING COUNT(DISTINCT deref(value(o)).numOperation) = (
    SELECT MAX(operation_count)
    FROM (
        SELECT COUNT(DISTINCT deref(value(ops)).numOperation) AS operation_count
        FROM comptes cs , table(cs.operations) ops
             WHERE EXTRACT(YEAR FROM value(ops).dateOp) =2024
        GROUP BY cs.numCompte
    )
);
```

sélectionne le compte qui a le nombre d'opérations effectuées égale au nombre maximal des opérations effectuées .

```
SQL> SELECT c.numCompte
  2  FROM comptes c , table(c.operations) o
  3  WHERE EXTRACT(YEAR FROM value(o).dateOp) =2024
  4  GROUP BY c.numCompte
  5  HAVING COUNT(DISTINCT deref(value(o)).numOperation) = (
  6      SELECT MAX(operation_count)
  7      FROM (
  8          SELECT COUNT(DISTINCT deref(value(ops)).numOperation) AS operation_count
  9          FROM comptes cs , table(cs.operations) ops
 10  WHERE EXTRACT(YEAR FROM value(ops).dateOp) =2024
 11          GROUP BY cs.numCompte
 12      )
 13  );

 NUMCOMPTE
----------
1200000020
```

```
SQL> insert into table (select c.prets from comptes c where c.numCompte=1110000011)
  2  (select ref(p) from prets p where  p.compte=(select ref(c) from comptes c where numCompte=1110000011));

2 lignes crÚÚes.

SQL> select a.nomAgence, a.calcul_prets() from agences a where a.numAgence=111;

NOMAGENCE                    A.CALCUL_PRETS()
---------------------------- ----------------
Agence Secondaire 11                        2

SQL>
```

Appel aux méthode

la méthode qui calcule pour chaque agence, le nombre de prêts effectués

```
SQL> select a.nomAgence, a.calcul_prets() from agences a where a.numAgence=111;

NOMAGENCE                     A.CALCUL_PRETS()
----------------------------- ----------------
Agence Secondaire 11                         3
```

la méthode qui calcule pour une agence le montant global des prêts effectués durant la période du 01-01-2020 au 01-01-2024.

```
SQL> select a.nomAgence, a.montant_global() from agences a where a.numAgence=111;

NOMAGENCE                    A.MONTANT_GLOBAL()
---------------------------- ------------------
Agence Secondaire 11                       5000
```

la méthode qui calcule le nombre d'agences d'une succursale donnée

```
SQL> select s.nomSucc, s.calcul_nb_agences() from succursales s where s.numSucc=001;

NOMSUCC                       S.CALCUL_NB_AGENCES()
----------------------------- ---------------------
Succursale Sud                                    5
```

la méthode qui liste toutes agences secondaires (avec la succursale rattachée) ayant au moins un prêt « ANSEJ »

```
SQL> select s.agences_liste() from succursales s where s.numSucc=001;

S.AGENCES_LISTE()(NOMAGENCE, NOMSUCC)
--------------------------------------------------------------------------------
TSETAGENCE(TAGENCESUCC('Agence Secondaire 7', 'Succursale Sud'))

SQL>
```

Partie II : No SQL – Modèle orienté « documents »

A – Modélisation orientée document

modélisation orientée document de la base de données


```
{
 "Prêt": {
  "NumPrêt": 10001,
  "montantPrêt": 20000,
  "dateEffet": "2022-12-15",
  "durée": 36,
  "typePrêt": "Véhicule",
  "tauxIntérêt": 0.05,
  "montantEchéance": 1000,
  "compte": {
   "NumCompte": 1001,
   "dateOuverture": "2023-01-01",
   "étatCompte": "Actif",
   "Solde": 5000,
   "client": {
    "NumClient": 1,
    "NomClient": "Client A",
    "TypeClient": "Particulier",
    "AdresseClient": "Adresse Client A",
    "NumTel": "0123456789",
    "Email": "clienta@example.com"
   },
   "agence": {
    "NumAgence": 101,
    "nomAgence": "Agence A",
    "adresseAgence": "Adresse Agence A",
    "catégorie": "Principale",
    "succursale": {
     "NumSucc": 1,
     "nomSucc": "Succursale Nord",
     "adresseSucc": "Adresse Succursale Nord",
     "région": "Nord"
    }
   }
  }
 }
}
```


Dans cette modélisation, chaque prêt contient des informations sur le compte associé, qui contient à son tour des informations sur le client et l'agence associés. L'agence contient des informations sur la succursale correspondante.

exemple de la modélisation :


```
{
 "Prêt": {
  "NumPrêt": 10001,
  "montantPrêt": 20000,
  "dateEffet": "2022-12-15",
  "durée": 36,
  "typePrêt": "Véhicule",
  "tauxIntérêt": 0.05,
  "montantEchéance": 1000,
  "compte": {
   "NumCompte": 1001,
   "dateOuverture": "2023-01-01",
   "étatCompte": "Actif",
   "Solde": 5000,
   "client": {
    "NumClient": 1,
    "NomClient": "Client A",
    "TypeClient": "Particulier",
    "AdresseClient": "Adresse Client A",
    "NumTel": "0123456789",
    "Email": "clienta@example.com"
   },
   "agence": {
    "NumAgence": 101,
    "nomAgence": "Agence A",
    "adresseAgence": "Adresse Agence A",
    "catégorie": "Principale",
    "succursale": {
     "NumSucc": 1,
     "nomSucc": "Succursale Nord",
     "adresseSucc": "Adresse Succursale Nord",
     "région": "Nord"
    }
   }
  }
 }
}
```


Justification du choix de conception :

Cette modélisation permet une représentation hiérarchique des données, ce qui correspond bien à la structure imbriquée des informations dans la base de données relationnelle. Elle simplifie également la navigation et l'accès aux données, en regroupant les informations connexes dans des documents.

Inconvénients de la conception

– La duplication des données peut survenir si les mêmes informations sont stockées dans plusieurs documents.

– Les mises à jour peuvent être complexes, surtout si les données sont dupliquées et doivent être maintenues cohérentes.

Remarque :

dans le reste du TP, nous allons exploiter une modélisation plus simple ou chaque objet est représenté dans une collection distincte, ce qui simplifie la structure des données et permet des requêtes plus directes sur chaque type d'objet.

- ## Collection "Succursales" :


```
[
  {
    "NumSucc": 1,
    "nomSucc": "Succursale Nord",
    "adresseSucc": "Adresse Succursale Nord",
    "région": "Nord"
  },
  {
    "NumSucc": 2,
    "nomSucc": "Succursale Sud",
    "adresseSucc": "Adresse Succursale Sud",
    "région": "Sud"
  }
]
```


- ## Collection "Agences" :


```
[
  {
    "NumAgence": 101,
    "nomAgence": "Agence A",
    "adresseAgence": "Adresse Agence A",
    "catégorie": "Principale",
    "NumSucc": 1
  },
  {
    "NumAgence": 102,
    "nomAgence": "Agence B",
    "adresseAgence": "Adresse Agence B",
    "catégorie": "Secondaire",
    "NumSucc": 2
  }
]
```

- **Collection "Clients" :**


```
[
  {
    "NumClient": 1,
    "NomClient": "Client A",
    "TypeClient": "Particulier",
    "AdresseClient": "Adresse Client A",
    "NumTel": "0123456789",
    "Email": "clienta@example.com",
    "NumAgence": 101
  },
  {
    "NumClient": 2,
    "NomClient": "Client B",
    "TypeClient": "Entreprise",
    "AdresseClient": "Adresse Client B",
    "NumTel": "0234567890",
    "Email": "clientb@example.com",
    "NumAgence": 102
  }
]
```


- **Collection "Comptes" :**

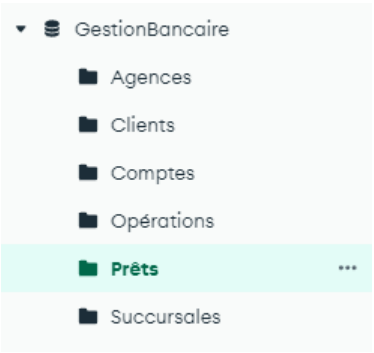

```
[
  {
    "NumCompte": 1001,
    "dateOuverture": "2023-01-01",
    "étatCompte": "Actif",
    "Solde": 5000,
    "NumClient": 1,
    "NumAgence": 101
  },
  {
    "NumCompte": 1002,
    "dateOuverture": "2022-11-01",
    "étatCompte": "Actif",
    "Solde": 8000,
    "NumClient": 2,
    "NumAgence": 102
  }
]
```

- **Collection "Opérations" :**


```
[
  {
    "NumOpération": 1,
    "NatureOp": "Crédit",
    "montantOp": 1000,
    "DateOp": "2023-01-05",
    "Observation": "Observation Opération 1",
    "NumCompte": 1001
  },
  {
    "NumOpération": 2,
    "NatureOp": "Débit",
    "montantOp": 500,
    "DateOp": "2023-02-10",
    "Observation": "Observation Opération 2",
    "NumCompte": 1002
  }
]
```

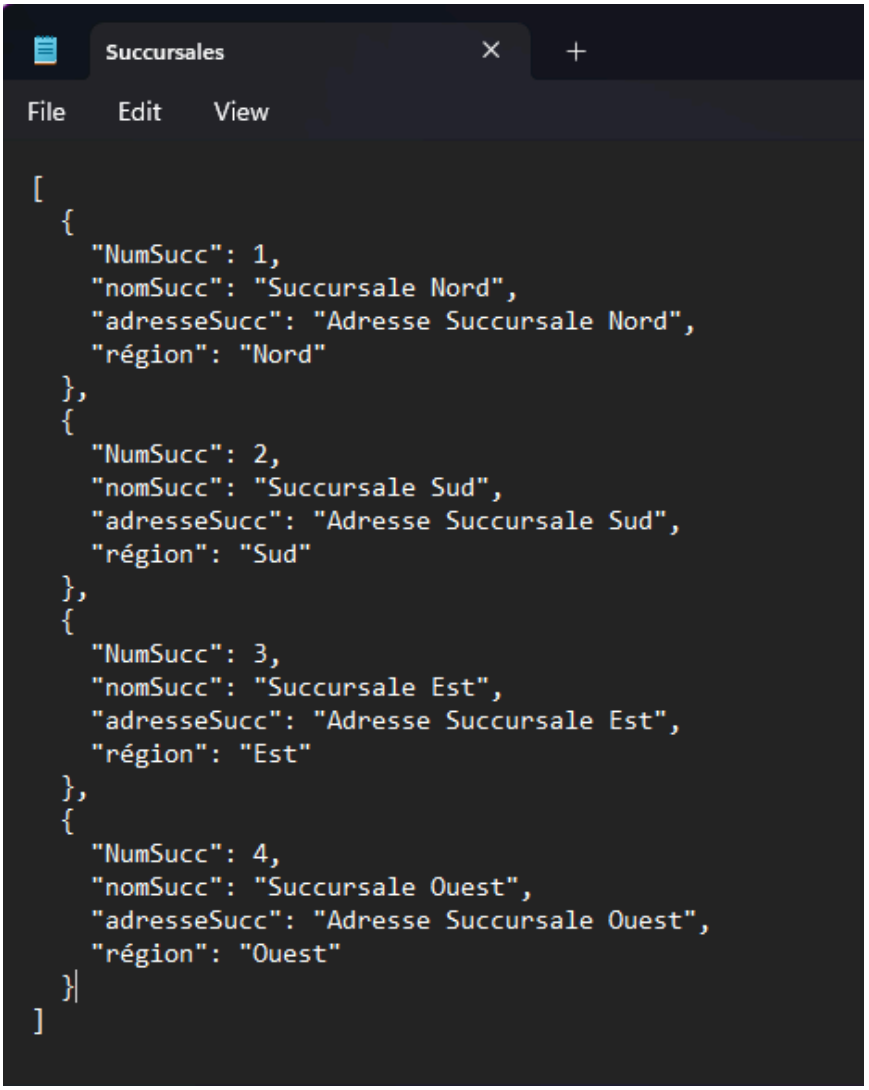

- **Collection "Prêts" :**

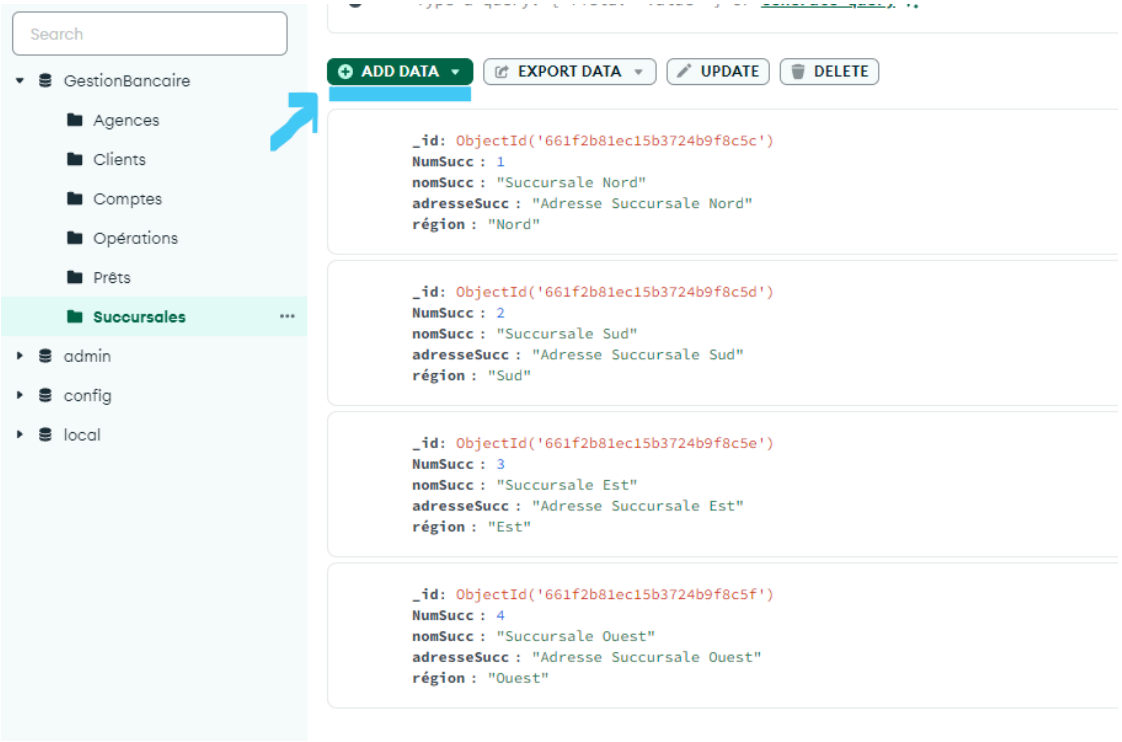

```
[
  {
    "NumPrêt": 10001,
    "montantPrêt": 20000,
    "dateEffet": "2022-12-15",
    "durée": 36,
    "typePrêt": "Véhicule",
    "tauxIntérêt": 0.05,
    "montantEchéance": 1000,
    "NumCompte": 1001
  },
  {
    "NumPrêt": 10002,
    "montantPrêt": 30000,
    "dateEffet": "2023-01-15",
    "durée": 48,
    "typePrêt": "Immobilier",
    "tauxIntérêt": 0.06,
    "montantEchéance": 1200,
    "NumCompte": 1002
  }
]
```



B. Remplissage de la base de données via un script

1. **on va créer des fichiers JSON qui contiennent nos données et les importer dans MongoDB**





2. comme ont peu utilisé la commande load() pour exécuter un fichier.js

Succursales insertion

File Edit View

```
db.Succursales.insertOne({
    "NumSucc": 1,
    "nomSucc": "Succursale Nord",
    "adresseSucc": "Adresse Succursale Nord",
    "région": "Nord"
});

db.Succursales.insertOne({
    "NumSucc": 2,
    "nomSucc": "Succursale Sud",
    "adresseSucc": "Adresse Succursale Sud",
    "région": "Sud"
});


db.Succursales.insertOne({
    "NumSucc": 3,
    "nomSucc": "Succursale Est",
    "adresseSucc": "Adresse Succursale Est",
    "région": "Est"
});
db.Succursales.insertOne({
    "NumSucc": 4,
    "nomSucc": "Succursale Ouest",
    "adresseSucc": "Adresse Succursale Ouest",
    "région": "Ouest"
});
```

on exécute ce fichier dans le terminale avec la commande :
load('/chemin/vers/le/fichier.js')

C.

1. **- Afficher tous prêts effectués auprès de l'agence de numéro 102**

// Étape 1 : Trouver les numéros de compte associés à l'agence de numéro 102

```
GestionBancaire> var numComptes = db.Comptes.find({ "NumAgence": 102 }).toArray(
).map(function (compte) { return compte.NumCompte;})
```

// Étape 2 : Trouver les prêts associés aux numéros de compte trouvés

```
GestionBancaire> db.Prêts.find({ "NumCompte": { $in: numComptes } })
[
  {
    _id: ObjectId('661f30f1ec15b3724b9f8c8e'),
    'NumPrêt': 2,
    'montantPrêt': 30000,
    dateEffet: '2022-02-01',
    'durée': 60,
    'typePrêt': 'Immobilier',
    'tauxIntérêt': 0.06,
    'montantEchéance': 1200,
    NumCompte: 1002
  },
  {
    _id: ObjectId('661f30f1ec15b3724b9f8c92'),
    'NumPrêt': 6,
    'montantPrêt': 32000,
    dateEffet: '2022-04-01',
    'durée': 48,
    'typePrêt': 'Immobilier',
    'tauxIntérêt': 0.06,
    'montantEchéance': 1000,
    NumCompte: 1006
  },
  {
    _id: ObjectId('661f30f1ec15b3724b9f8c96'),
    'NumPrêt': 10,
    'montantPrêt': 28000,
    dateEffet: '2022-06-01',
    'durée': 48,
    'typePrêt': 'ANSEJ',
    'tauxIntérêt': 0.07,
    'montantEchéance': 850,
```

Explication : La fonction toArray() est utilisée pour convertir le curseur retourné par la méthode find() en un tableau JavaScript contenant tous les documents correspondants.

- La fonction map() crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant. Dans cette requête, la fonction map est utilisée pour extraire le champ "NumCompte" de chaque document de la collection de comptes.

2. **Afficher tous prêts effectués auprès des agences rattachées aux succursales de la région « Nord ». Préciser NumPrêt, NumAgence, numCompte, numClient et MontantPrêt.**

```
GestionBancaire> db.Agences.aggregate([{ $lookup: { from: "Succursales", localField: "NumSucc", foreignField: "
NumSucc", as: "succursale" } }, { $match: { "succursale.région": "Nord" } }, { $lookup: { from: "Comptes", loca
lField: "NumAgence", foreignField: "NumAgence", as: "compte" } }, { $lookup: { from: "Prêts", localField: "comp
te.NumCompte", foreignField: "NumCompte", as: "prêt" } }, { $project: { _id: 0, "prêt.NumPrêt": 1, NumAgence: 1
, "compte.NumCompte": 1, "compte.NumClient": 1, "prêt.montantPrêt": 1 } }]);
[
  {
    NumAgence: 101,
    compte: [
      { NumCompte: 1001, NumClient: 1 },
      { NumCompte: 1005, NumClient: 5 },
      { NumCompte: 1009, NumClient: 9 }
    ],
    'prêt': [
      { 'NumPrêt': 1, 'montantPrêt': 20000 },
      { 'NumPrêt': 5, 'montantPrêt': 18000 },
      { 'NumPrêt': 9, 'montantPrêt': 21000 }
    ]
  },
  {
    NumAgence: 102,
    compte: [
      { NumCompte: 1002, NumClient: 2 },
      { NumCompte: 1006, NumClient: 6 },
      { NumCompte: 1010, NumClient: 10 }
    ],
    'prêt': [
      { 'NumPrêt': 2, 'montantPrêt': 30000 },
      { 'NumPrêt': 6, 'montantPrêt': 32000 },
      { 'NumPrêt': 10, 'montantPrêt': 28000 }
    ]
  }
]
GestionBancaire>
```

Explication : Cette requête utilise l'agrégation MongoDB pour effectuer

- $lookup: C'est une étape qui permet de joindre deux collections en utilisant un champ commun entre elles. Dans ce cas, nous joignons la collection des agences avec celle des succursales en utilisant le champ "NumSucc", ce qui nous permet d'obtenir les agences rattachées à chaque succursale.

- $match: Cette étape filtre les documents pour ne garder que ceux qui répondent à un critère spécifique. Ici, nous filtrons les succursales pour ne conserver que celles dont la région est "Nord".

- $lookup : Encore une fois, nous utilisons l'opération de jointure pour cette fois-ci joindre la collection des comptes avec celle des agences. Nous utilisons le champ "NumAgence" pour cette jointure, ce qui nous permet d'obtenir les comptes associés à chaque agence de la région Nord.

- $lookup : Dans cette étape, nous joignons la collection des prêts avec celle des comptes en utilisant le champ "NumCompte". Ainsi, nous obtenons les prêts effectués auprès des agences de la région Nord

- $project : Enfin, cette étape permet de spécifier les champs que nous voulons inclure dans le résultat final. Nous sélectionnons ici le numéro du prêt (NumPrêt), le numéro de l'agence (NumAgence), le numéro du compte (NumCompte), le numéro du client associé au compte (NumClient) et le montant du prêt (montantPrêt).

3. **- Récupérer dans une nouvelle collection Agence-NbPrêts, les numéros des agences et le nombre total de prêts, par agence ; la collection devra être ordonnée par ordre décroissant du nombre de prêts. Afficher le contenu de la collection.**

   // // Opération d'agrégation $lookup pour joindre les collections "Prêts" et "Comptes" en utilisant le champ "NumCompte" comme clé de jointure.

   // $unwind : Décompose les tableaux résultants de la jointure pour qu'ils puissent être traités dans les étapes suivantes.

   // $group : Regroupe les documents par le numéro d'agence du compte et calcule le nombre total de prêts pour chaque groupe à l'aide de l'opérateur d'agrégation $sum.

   // $sort : Trie les résultats par ordre décroissant du nombre total de prêts.

```
GestionBancaire> var pipeline = [{ $lookup: { from: "Comptes", localField: "NumCompte", foreignField: "NumCompte",as: "compte" } }, {$unwind
$unwind: "$compte" }, {$group: {_id: "$compte.NumAgence",totalPrêts: { $sum: 1 } } }, {$sort: { totalPrêts: -1 } }];
```

   // Exécution de l'opération d'agrégation

```
GestionBancaire> var result = db.Prêts.aggregate(pipeline);
```

   // Créer une nouvelle collection "Agence-NbPrêts" et insère les résultats

```
GestionBancaire> db.createCollection("Agence-NbPrêts");result.forEach(function(agence) {db["Agence-NbPrêts"].insertOne({NumAgence: agence._i
d, totalPrêts: agence.totalPrêts });});
ce-NbPrêts:");
```

// Afficher le contenu de la nouvelle collection

```
{
  _id: ObjectId('661f456dfa051b3a7a117b7d'),
  NumAgence: 102,
  'totalPrêts': 3
}
{
  _id: ObjectId('661f456dfa051b3a7a117b7e'),
  NumAgence: 101,
  'totalPrêts': 3
}
{
  _id: ObjectId('661f456dfa051b3a7a117b7f'),
  NumAgence: 103,
  'totalPrêts': 2
}
{
  _id: ObjectId('661f456dfa051b3a7a117b80'),
  NumAgence: 104,
  'totalPrêts': 2
}
```

4. **Dans une collection Prêt-ANSEJ, récupérer tous les prêts liés à des dossiers ANSEJ. Préciser NumPrêt, numClient, MontantPrêt et dateEffet.**

// Créez la collection "Prêt-ANSEJ"

```
GestionBancaire> db.createCollection("Prêt-ANSEJ");
{ ok: 1 }
```

// Requête pour récupérer les prêts liés à des dossiers ANSEJ

```
GestionBancaire> var pipeline = [{ $match: { typePrêt: "ANSEJ" }},{ $lookup: {from: "Comptes"
,localField: "NumCompte", foreignField: "NumCompte", as: "compteInfo"}},{ $unwind: "$compteIn
fo" }, { $project: { _id: 0,NumPrêt: 1, NumClient: "$compteInfo.NumClient",montantPrêt: 1,dat
eEffet: 1}}];
```

// Exécutez l'agrégation

```
GestionBancaire> var result = db.Prêts.aggregate(pipeline);
```

// Insérez les résultats dans la collection "Prêt-ANSEJ"

```
GestionBancaire> result.forEach(function(prêt) {db["Prêt-ANSEJ"].insertOne(prêt);});
```

// Affichez le contenu de la collection "Prêt-ANSEJ"

```
GestionBancaire> var contenuCollection = db["Prêt-ANSEJ"].find();

GestionBancaire> contenuCollection.forEach(printjson);
{
  _id: ObjectId('661f4a06fa051b3a7a117b83'),
  'NumPrêt': 3,
  'montantPrêt': 25000,
  dateEffet: '2022-02-15',
  NumClient: 3
}
{
  _id: ObjectId('661f4a06fa051b3a7a117b84'),
  'NumPrêt': 10,
  'montantPrêt': 28000,
  dateEffet: '2022-06-01',
  NumClient: 10
}
{
  _id: ObjectId('661f4caefa051b3a7a117b85'),
  'NumPrêt': 3,
  'montantPrêt': 25000,
  dateEffet: '2022-02-15',
  NumClient: 3
}
{
  _id: ObjectId('661f4caefa051b3a7a117b86'),
  'NumPrêt': 10,
  'montantPrêt': 28000,
  dateEffet: '2022-06-01',
  NumClient: 10
}
```

5. Afficher tous les prêts effectués par des clients de type « Particulier ». On affichera le NumClient, NomClient, NumPrêt, montantPrêt.

// Requête pour afficher tous les prêts effectués par des clients de type "Particulier"

1. On Recherche des prêts avec la fonction $lookup
2. On Dérouler les informations client avec la fonction $unwind
3. On filtre les clients de type “Particulier” avec la fonction $match
4. Puis, on projette les champs requis avec la fonction $project

```
GestionBancaire> var pipeline = [{$lookup: { from: "Comptes",localField: "NumCompte",
foreignField: "NumCompte",as: "compteInfo"} }, {$unwind: "$compteInfo"},{$lookup: {fro
m: "Clients",localField: "compteInfo.NumClient", foreignField: "NumClient",as: "client
Info" }}, {$unwind: "$clientInfo" },{$match: { "clientInfo.TypeClient": "Particulier"
}},{$project: {_id: 0,NumClient: "$clientInfo.NumClient", NomClient: "$clientInfo.NomC
lient", NumPrêt: 1,  montantPrêt: 1}}]
```

// // Exécution de l'agrégation

```
GestionBancaire> var result = db.Prêts.aggregate(pipeline);
```

// Affichage des résultats

```
GestionBancaire> result.forEach(printjson);
{
  'NumPrêt': 1,
  'montantPrêt': 20000,
  NumClient: 1,
  NomClient: 'Client A'
}
{
  'NumPrêt': 3,
  'montantPrêt': 25000,
  NumClient: 3,
  NomClient: 'Client C'
}
{
  'NumPrêt': 5,
  'montantPrêt': 18000,
  NumClient: 5,
  NomClient: 'Client E'
}
{
  'NumPrêt': 7,
  'montantPrêt': 23000,
  NumClient: 7,
  NomClient: 'Client G'
}
{
  'NumPrêt': 9,
  'montantPrêt': 21000,
  NumClient: 9,
  NomClient: 'Client I'
}
```

6. Augmenter de 2000 DA, le montant de l'échéance de tous les prêts non encore soldés, dont la date d'effet est antérieure à (avant) janvier 2021.

```
GestionBancaire> db.Prêts.updateMany({ "montantEchéance": { $exists:
 true }, "dateEffet": { $lt: ISODate("2021-01-01") },"montantEchéanc
e": { $ne: null, $gt: 0 } },{ $inc: { "montantEchéance": 2000 } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
  {
    _id: ObjectId('661f30f1ec15b3724b9f8c95'),
    'NumPrêt': 9,
    'montantPrêt': 21000,
    dateEffet: ISODate('2020-05-15T00:00:00.000Z'),
    'durée': 24,
    'typePrêt': 'Véhicule',
    'tauxIntérêt': 0.06,
    'montantEchéance': 2900,
    NumCompte: 1009
  },
```

7. Reprendre la 3ème requête à l'aide du paradigme Map-Reduce

// Étape 1 : Définition de la fonction Map Function

La fonction map prend chaque document de la collection Comptes et émet une paire clé-valeur où la clé est le numéro d'agence (NumAgence) et la valeur est toujours 1.

```
GestionBancaire> var mapFunction = function() { emit (this.NumAgence, 1); };
```

// Étape 2 : Définition de la fonction Reduce Function

la fonction reduce prend une clé (dans ce cas, le numéro d'agence) et une liste de valeurs (dans ce cas, une liste de 1), puis retourne la somme de ces valeurs, donnant ainsi le nombre total de prêts par agence.

```
GestionBancaire> var reduceFunction = function(key, values) { return Array.sum(values); };
```

// Étape 3 : Exécution de la fonction MapReduce

```
GestionBancaire> var mapReduceResult = db.Comptes.mapReduce(mapFunction, reduceFunction,{ out: "Agence-NbPrêts" });
```

// Étape 4 : Affiche du résultat trié par ordre décroissant

```
GestionBancaire> db["Agence-NbPrêts"].find().sort({ value: -1 }
).forEach(function(item) {print("NumAgence:", item._id, ",total
Prêts:", item.value);});
NumAgence: 102 ,totalPrêts: 3
NumAgence: 101 ,totalPrêts: 3
NumAgence: 103 ,totalPrêts: 2
NumAgence: 104 ,totalPrêts: 2
```

8. Avec votre conception, peut-on répondre à la requête suivante : Afficher toutes les opérations de crédit effectuées sur les comptes des clients de type « Entreprise » pendant l'année 2023. Justifiez votre réponse.

   Oui, avec la conception actuelle, il est possible de répondre à la requête suivante : Afficher toutes les opérations de crédit effectuées sur les comptes des clients de type « Entreprise » durant l'année 2023. Voici comment cela peut être fait :

   1. **Identification des collections nécessaires :**

   Nous avons besoin des collections Opération, Compte et Client.

   2. **Filtrage des opérations de crédit :**

   Nous filtrons les opérations pour ne sélectionner que celles qui sont des opérations de crédit

   3. **Filtrage des clients de type « Entreprise » :**

   Nous filtrons les opérations pour ne sélectionner que celles effectuées sur les comptes des clients de type « Entreprise ».

   4. **Filtrage par année 2023 :**

   Nous filtrons les opérations pour ne sélectionner que celles qui ont eu lieu en 2023.

   5. **Affichage des résultats :**

   Une fois que toutes les opérations répondant aux critères ont été identifiées, elles peuvent être affichées.

D- Analyse

1. **Collections séparées :**

   - **Avantages :**

     - Offre une flexibilité pour consulter et mettre à jour des objets individuellement, facilitant ainsi la gestion des données.

     - Prise en charge de la scalabilité et des performances lors du traitement de grandes collections, ce qui permet de gérer efficacement les données volumineuses.

     - Permet des indexations et des requêtes plus efficaces sur des champs spécifiques, améliorant ainsi les performances des requêtes.

   - **Inconvénients :**
     - Complexité accrue lors de la gestion des relations entre les objets, ce qui peut rendre la maintenance plus difficile

     - Nécessite plusieurs requêtes ou jointures pour récupérer des données liées, ce qui peut augmenter la complexité des requêtes.

2. **Objets imbriqués :**

   - **Avantages :**

     - Regroupe des données liées dans un seul document, réduisant ainsi le besoin de jointures et simplifiant la récupération de hiérarchies d'objets complets.

     - Peut améliorer les performances en lecture

   - **Inconvénients :**

     -Flexibilité limitée lors de la consultation et de la mise à jour d'objets spécifiques imbriqués, car toute modification peut nécessiter la mise à jour du document entier.

     -Les grands tableaux imbriqués peuvent avoir un impact sur les performances en écriture et les limites de taille des documents, car les documents MongoDB ont une limite de 16 Mo.

     - Complexité accrue lors de la modification ou de la mise à jour de structures imbriquées

Conclusion

En conclusion, le choix entre des collections séparées et des objets imbriqués dans MongoDB dépend des besoins spécifiques de chaque application. Les collections séparées offrent une flexibilité accrue pour gérer des relations complexes entre les données, tandis que les objets imbriqués simplifient la récupération de données liées et peuvent améliorer les performances de lecture.

Une approche hybride, combinant ces deux méthodes, peut être avantageuse pour tirer parti des forces de chaque approche tout en atténuant leurs faiblesses respectives. Cela permet de concevoir des schémas de données flexibles et évolutifs, adaptés à divers cas d'utilisation dans MongoDB.