



Rapport du Projet de Base de Données

SQL3-Oracle et NoSQL (MongoDB)

Master 1 Systèmes Informatiques Intelligents

Chargée de cours : Mme S. BOUKHEDOUMA

Chargée de TP : Mme Z. CHALLAL

SAYOUD Maissa **191931040670**

BOULKABOUL Amira **202031043294**

Mai 2024

Table des matières

Introduction	1
Partie I Relationnel Objet.....	2
1. Modélisation Orientée Objet	3
2. Création des TableSpaces et Utilisateur	4
3. Langage de Définition de Données	5
4. Création des Instances dans les Tables.....	16
5. Langage d'Interrogation de Données	22
Partie II NoSQL – Modèle orienté « documents »	34
1. Modélisation Orientée Document	35
2. Remplir la base de données	38
3. Répondre aux requêtes	40
4. Analyse.....	48
Conclusion.....	50

Introduction

Nous nous intéressons dans ce projet à une base de données de gestion des opérations et des prêts bancaires auprès d'une banque donnée. Nous couvrons deux aspects de la gestion de données : le modèle relationnel orienté objet et le modèle NoSQL orienté documents. À travers ces deux parties, nous explorerons la conception, l'implémentation et l'interrogation de bases de données avancées, en expliquant en détail chaque tâche réalisée avec des captures montrant les résultats de l'exécution .

Partie I : Relationnel Objet

Dans cette première partie, nous commençons par la modélisation en transformant un schéma relationnel en un diagramme de classes UML pour représenter clairement les entités et leurs relations. Ensuite, nous procédons la création des tablespaces et des utilisateurs, la définition des types abstraits et des méthodes pour répondre finalement aux requêtes.

Partie II : NoSQL – Modèle orienté « documents »

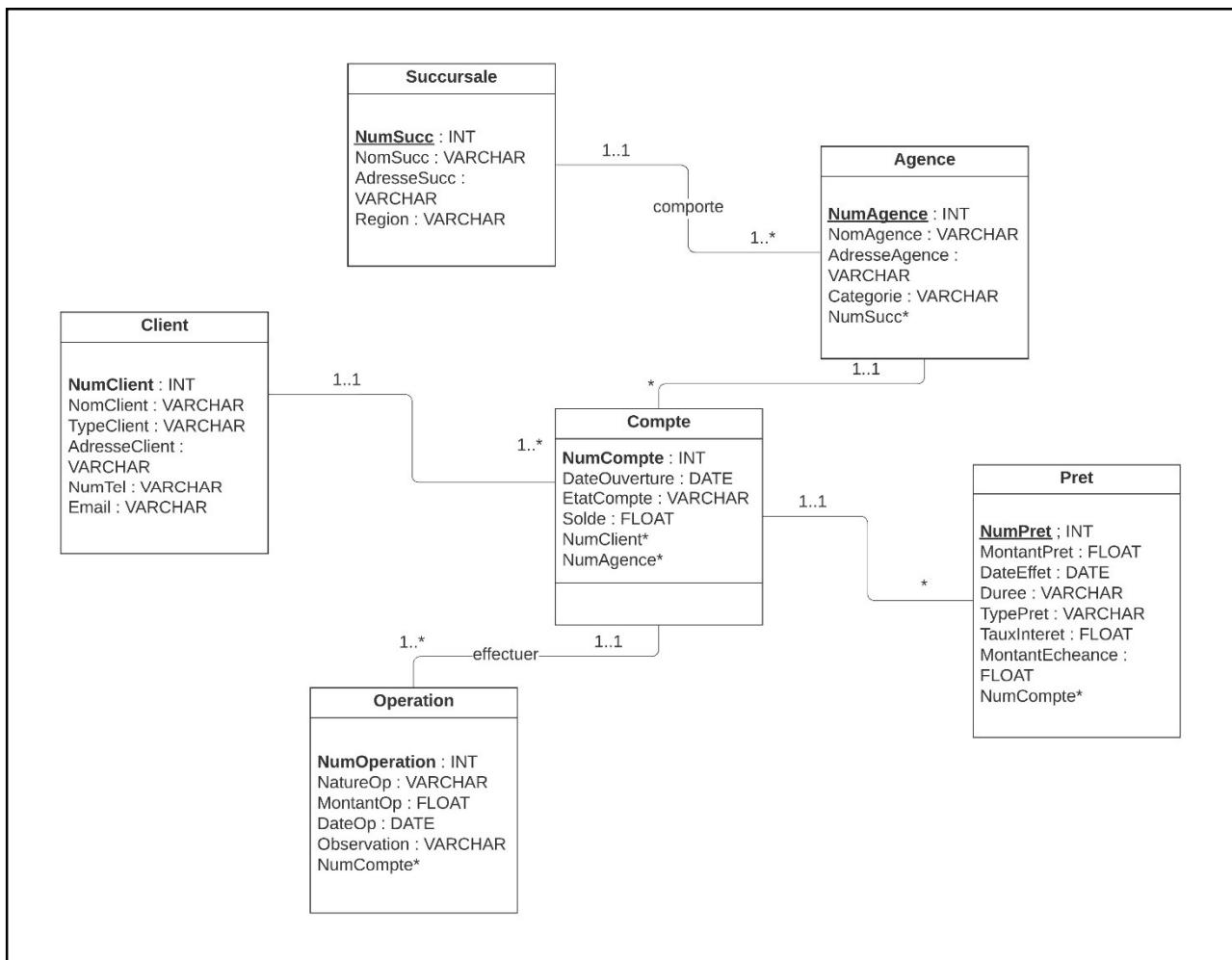
La deuxième partie du rapport explore le modèle NoSQL orienté documents. Nous proposons alors une modélisation orientée document de la base de données précédente, nous remplissons, via un script, cette base et nous l'interrogeons à travers des requêtes pertinentes. Enfin, nous analysons les performances et implications de chaque approche.

Partie I

Relationnel Objet

1. Modélisation Orientée Objet

1. Transformez ce schéma relationnel en un schéma Objet (diagramme de classes UML)



2. Création des TableSpaces et Utilisateur

2. Créer deux TableSpaces SQL3_TBS et SQL3_TempTBS

```
SQL> CREATE TABLESPACE SQL3_TBS
  2  DATAFILE 'C:\Oracle19c\ tablespaces\SQL3_TBS.dat'
  3  SIZE 100M AUTOEXTEND
  4  ON;
Tablespace crée.
```

```
SQL> CREATE TEMPORARY TABLESPACE SQL3_TempTBS
  2  TEMPFILE 'C:\Oracle19c\ tablespaces\sql3_tempTBS.dat'
  3  SIZE 100M AUTOEXTEND
  4  ON;
Tablespace crée.
```

3. Créer un utilisateur SQL3 en lui attribuant les deux TableSpaces créés précédemment

```
SQL> CREATE USER SQL3
  2  IDENTIFIED BY 0000
  3  DEFAULT TABLESPACE SQL3_TBS
  4  TEMPORARY TABLESPACE SQL3_TempTBS;
Utilisateur créé.
```

4. Donner tous les privilèges à cet utilisateur.

```
SQL> GRANT ALL PRIVILEGES TO SQL3;
Autorisation de priviléges (GRANT) acceptée.
```

3. Langage de Définition de Données

5. En se basant sur le diagramme de classes établi, définir tous les types abstraits nécessaires. Définir toutes les associations qui existent.

En définissant ces types abstraits et en spécifiant les associations appropriées entre eux, nous pouvons mieux représenter la structure et les relations de la base de données de gestion des opérations et des prêts bancaires.

```
SQL> connect SQL3/0000
Connecté.
SQL>
```

En relationnel objet, une classe est définie par un type, alors toutes nos classes sont des types abstraits :

```
SQL> CREATE OR REPLACE TYPE Type_Client;
  2 /
Type crée.

SQL>
SQL> CREATE OR REPLACE TYPE Type_Succ;
  2 /
Type crée.

SQL>
SQL> CREATE OR REPLACE TYPE Type_Agence;
  2 /
Type crée.

SQL>
SQL> CREATE OR REPLACE TYPE Type_Compte;
  2 /
Type crée.

SQL>
SQL> CREATE OR REPLACE TYPE Type_Operation;
  2 /
Type crée.

SQL>
SQL> CREATE OR REPLACE TYPE Type_Pret;
  2 /
Type crée.
```

Nous créons les types nécessaires aux associations "les tables imbriquées des références" :

```
SQL> CREATE TYPE t_set_ref_Agence AS TABLE OF REF Type_Agence;
2 /
Type crÚÚ.

SQL>
SQL> CREATE TYPE t_set_ref_Succ AS TABLE OF REF Type_Succ;
2 /
Type crÚÚ.

SQL>
SQL> CREATE TYPE t_set_ref_Client AS TABLE OF REF Type_Client;
2 /
Type crÚÚ.

SQL>
SQL> CREATE TYPE t_set_ref_Compte AS TABLE OF REF Type_Compte;
2 /
Type crÚÚ.

SQL>
SQL> CREATE TYPE t_set_ref_Operation AS TABLE OF REF Type_Operation;
2 /
Type crÚÚ.

SQL>
SQL> CREATE TYPE t_set_ref_Pret AS TABLE OF REF Type_Pret;
2 /
Type crÚÚ.
```

Vérification :

```
SQL> SELECT type_name FROM user_types;
```

```
TYPE_NAME
```

```
--  
TYPE_PRET  
T_SET_REF_SUCC  
T_SET_REF_CLIENT  
T_SET_REF_COMPTE  
T_SET_REF_OPERATION  
T_SET_REF_PRET  
TYPE_SUCC  
TYPE_AGENCE  
T_SET_REF_AGENCE  
TYPE_CLIENT  
TYPE_COMPTE
```

```
TYPE_NAME
```

```
--  
TYPE_OPERATION
```

```
12 lignes sélectionnées.
```

Nous mettons enfin les types à jour en prenons en considération les associations :

```
SQL> CREATE OR REPLACE TYPE Type_Succ AS OBJECT  
2  (  
3      NumSucc      INT          ,  
4      NomSucc      VARCHAR(50)   ,  
5      AdresseSucc  VARCHAR(100)  ,  
6      Region       VARCHAR(10)   ,  
7      Succursale_Agence t_set_ref_Agence  
8  );  
9 /
```

```
Type créé.
```

```
SQL> CREATE OR REPLACE TYPE Type_Agence AS OBJECT
 2  (
 3      NumAgence      INT      ,
 4      NomAgence      VARCHAR(50) ,
 5      AdresseAgence  VARCHAR(100),
 6      Categorie      VARCHAR(10) ,
 7      Agence_Succursale REF Type_Succ,
 8      Agence_Compte  t_set_ref_Compte
 9  );
10 /
```

Type crûû.

```
SQL> CREATE OR REPLACE TYPE Type_Compte AS OBJECT
 2  (
 3      NumCompte      INT      ,
 4      DateOuverture DATE      ,
 5      EtatCompte     VARCHAR(10) ,
 6      Solde          FLOAT    ,
 7      Compte_Agence  REF Type_Agence,
 8      Compte_Client  REF Type_Client,
 9      Compte_Operation t_set_ref_Operation,
10      Compte_Pret    t_set_ref_Pret
11  );
12 /
```

Type crûû.

```
SQL> CREATE OR REPLACE TYPE Type_Client AS OBJECT
 2  (
 3      NumClient      INT      ,
 4      NomClient      VARCHAR(50) ,
 5      TypeClient     VARCHAR(15) ,
 6      AdresseClient  VARCHAR(100),
 7      NumTel         VARCHAR(12) ,
 8      Email          VARCHAR(100),
 9      Client_Compte  t_set_ref_Compte
10  );
11 /
```

Type crûû.

```
SQL> CREATE OR REPLACE TYPE Type_Operation AS OBJECT
 2  (
 3      NumOperation   INT      ,
 4      NatureOp       VARCHAR(10) ,
 5      MontantOp      FLOAT    ,
 6      DateOp         DATE      ,
 7      Observation    VARCHAR(100) ,
 8      Operation_Compte REF Type_Compte
 9  );
10 /
```

Type crûû.

```
SQL> CREATE OR REPLACE TYPE Type_Pret AS OBJECT
  2  (
  3      NumPret          INT          ,
  4      MontantPret      FLOAT        ,
  5      DateEffet       DATE         ,
  6      Duree            VARCHAR(10)   , -- tout le reste est un VARCHAR selon l'annonce
  7      TypePret         VARCHAR(15)   ,
  8      TauxInteret     FLOAT        ,
  9      MontantEcheance FLOAT        ,
 10      Pret_Compte     REF Type_Compte
 11  );
 12 /  
Type crÚÚ.
```

6. Définir les méthodes permettant de :

La définition d'une méthode se fait en deux étapes :

1/ Définition de la signature de la méthode.

2/ Déclaration du corps de la méthode (body).

- Calculer pour chaque agence, le nombre de prêts effectuées.

```
SQL> ALTER TYPE Type_Agence ADD MEMBER FUNCTION Nombre_Prets_Eff RETURN NUMBER
  CASCADE;  
Type modifiÚ.
```

```

SQL>
SQL> ALTER TYPE Type_Agence ADD MEMBER FUNCTION Nombre_Prets_Eff RETURN NUMBER
CASCADE;

Type modifi .

SQL> CREATE OR REPLACE TYPE BODY Type_Agence
  2  AS
  3    MEMBER FUNCTION Nombre_Prets_Eff
  4      RETURN NUMBER
  5      IS
  6        n NUMBER;
  7      BEGIN
  8
  9        SELECT COUNT(deref(value (p)).NumPret)
 10        INTO n
 11        FROM TABLE(self.Agence_Compte) c , Table(deref(value(c)).Compte_P
et) p;
 12        RETURN n;
 13      END;
 14  END;
 15 /

Corps de type cr .

```

V rification :

Nom	NULL ?	Type

NUMAGENCE		NUMBER(38)
NOMAGENCE		VARCHAR2(50)
ADRESSEAGENCE		VARCHAR2(100)
CATEGORIE		VARCHAR2(10)
AGENCE_SUCCURSALE		REF OF TYPE_SUCC
AGENCE_COMPTE		T_SET_REF_COMPTE
METHOD		

MEMBER FUNCTION NOMBRE_PRETS_EFF RETURNS NUMBER		

- Calculer pour chaque succursale, le nombre d'agences principales qui lui sont rattach es.

```

SQL> ALTER TYPE Type_Succ ADD MEMBER FUNCTION Nombre_Agences_Principales RETURN
NUMBER CASCADE;

Type modifi .

```

```

SQL> CREATE OR REPLACE TYPE BODY Type_Succ
  2  AS
  3  MEMBER FUNCTION Nombre_Agences_Principales
  4    RETURN NUMBER
  5  IS
  6    n NUMBER;
  7  BEGIN
  8    SELECT concat(self.NumSucc,
  9      (SELECT COUNT(*)
 10       FROM TABLE(self.Succursale_Agence) a
 11       WHERE deref(value(a)).Categorie = 'principale'
 12     )) into n
 13    FROM Succursale s;
 14    RETURN n;
 15  END;
 16 END;
 17 /

```

Corps de type crû.

Vérification :

```

SQL> desc type_succ
   Nom                               NULL ?    Type
   -----
   -
NUMSUCC                                NUMBER(38)
NOMSUCC                                 VARCHAR2(50)
ADRESSESUCC                             VARCHAR2(100)
REGION                                  VARCHAR2(10)
SUCCURSALE_AGENCE                         T_SET_REF_AGENCE

METHOD
-----
 MEMBER FUNCTION NOMBRE_AGENCES_PRINCIPALES RETURNS NUMBER

```

- Calculer pour une agence (de numéro donné), le montant global des prêts effectués durant la période du 01-01-2020 au 01-01-2024.

```

SQL> ALTER TYPE Type_Agence ADD MEMBER FUNCTION Montant_Global_Prets_Periode RE
TURN NUMBER CASCADE;

Type modifié.

```

```

SQL> CREATE OR REPLACE TYPE BODY Type_Agence
  2  AS
  3  MEMBER FUNCTION Nombre_Prets_Eff
  4    RETURN NUMBER
  5  IS
  6    n NUMBER;
  7  BEGIN
  8
  9    SELECT COUNT(deref(value (p)).NumPret)
10      INTO n
11      FROM TABLE(self.Agence_Compte) c , Table(deref(value(c)).Compte_P
et) p;
12    RETURN n;
13  END Nombre_Prets_Eff;
14
15  MEMBER FUNCTION Montant_Global_Prets_Periode
16    RETURN NUMBER
17  IS
18    nb NUMBER;
19  BEGIN
20    SELECT SUM(deref(value(p)).MontantPret)
21      INTO nb
22      FROM table (self.Agence_Compte) a , table (deref(value(a)).Compte_
Pret) p
23      WHERE self.NumAgence = 101
24      AND value(p).DateEffet BETWEEN '01/01/2020' AND '01/01/2024';
25
26    RETURN nb;
27  END Montant_Global_Prets_Periode;
28
29 END;
30 /

```

Corps de type crÚÚ.

Vérification :

```

SQL> desc type_agence
   Nom                                NULL ?    Type
   -
   -
NUMAGENCE                           NUMBER(38)
NOMAGENCE                            VARCHAR2(50)
ADRESSEAGENCE                         VARCHAR2(100)
CATEGORIE                            VARCHAR2(10)
AGENCE_SUCCURSALE                     REF OF TYPE_SUCC
AGENCE_COMPTE                         T_SET_REF_COMPTE

METHOD
-----
 MEMBER FUNCTION NOMBRE_PRETS_EFF RETURNS NUMBER

METHOD
-----
 MEMBER FUNCTION MONTANT_GLOBAL_PRETS_PERIODE RETURNS NUMBER

```

- Lister toutes agences secondaires (avec la succursale rattachée) ayant au moins un prêt « ANSEJ »

```
SQL> alter type type_Succ add member function Agences_Ayant_Pret return number
cascade;
```

Type modifié.

```
SQL> CREATE OR REPLACE TYPE BODY Type_Succ
  2  AS
  3
  4      MEMBER FUNCTION Nombre_Agences_Principales
  5          RETURN NUMBER
  6      IS
  7          n NUMBER;
  8      BEGIN
  9          SELECT concat(self.NumSucc,
 10                  (SELECT COUNT(*)
 11                      FROM TABLE(self.Succursale_Agence) a
 12                      WHERE deref(value(a)).Categorie = 'principale'
 13                  )) into n
 14          FROM Succursale s;
 15          RETURN n;
 16      END Nombre_Agences_Principales;
 17
 18      MEMBER FUNCTION Agences_Ayant_Pret
 19          RETURN NUMBER
 20      IS
 21          nbr NUMBER;
 22      BEGIN
 23          SELECT CONCAT(deref(value(a)).NumAgence, self.NumSucc)
 24              INTO nbr
 25              FROM TABLE (self.Succursale_Agence) a, TABLE (deref(VALUE(a)).Agen
ce_Compte) c, TABLE (deref(VALUE(c)).Compte_Pret) p
 26              WHERE VALUE(p).TypePret = 'ANSEJ'
 27              AND VALUE(a).Categorie = 'Secondaire';
 28          RETURN nbr;
 29      END Agences_Ayant_Pret;
 30
 31
 32  END;
 33 /
```

Corps de type créé.

Vérification :

```

SQL> desc type_succ
   Nom                                NULL ?    Type
   -
   -
  NUMSUCC                               NUMBER(38)
  NOMSUCC                               VARCHAR2(50)
  ADRESSESUCC                            VARCHAR2(100)
  REGION                                 VARCHAR2(10)
  SUCCURSALLE_AGENCE                     T_SET_REF_AGENCE

METHOD
-----
 MEMBER FUNCTION NOMBRE_AGENCES_PRINCIPALES RETURNS NUMBER

METHOD
-----
 MEMBER FUNCTION AGENCES_AYANT_PRET RETURNS NUMBER

```

7. Définir les tables nécessaires à la base de données.

```

SQL> CREATE TABLE Succursale OF Type_Succ
  2  (
  3      PRIMARY KEY (NumSucc),
  4      CONSTRAINT ck_region_succursale CHECK (Region IN ('Nord', 'Sud', 'Est',
  ', 'Ouest'))
  5  )
  6      NESTED TABLE Succursale_Agence STORE AS table_Succursale_Agence;

Table crée.

```

```

SQL> CREATE TABLE Agence OF Type_Agence
  2  (
  3      PRIMARY KEY (NumAgence),
  4      FOREIGN KEY (Agence_Succursale) REFERENCES Succursale,
  5      CONSTRAINT ck_categorie_agence CHECK (Categorie IN ('Principale', 'Se
condaire'))
  6  )
  7      NESTED TABLE Agence_Compte STORE AS table_Agence_Compte
  8  ;

Table crée.

```

```

SQL> CREATE TABLE Client OF Type_Client
  2  (
  3      PRIMARY KEY (NumClient),
  4      CONSTRAINT ck_type_client CHECK (TypeClient IN ('Particulier', 'Entre
prise'))
  5  )
  6      NESTED TABLE Client_Compte STORE AS table_Client_Compte
  7  ;

Table crée.

```

```
SQL> CREATE TABLE Compte OF Type_Compte
  2  (
  3      PRIMARY KEY (NumCompte),
  4      FOREIGN KEY (Compte_Client) REFERENCES Client,
  5      FOREIGN KEY (Compte_Agence) REFERENCES Agence,
  6      CONSTRAINT ck_etat_compte CHECK (EtatCompte IN ('Actif', 'Bloque'))
  7  )
  8      NESTED TABLE Compte_Operation STORE AS table_Compte_Operation,
  9      NESTED TABLE Compte_Pret STORE AS table_Compte_Pret
10  ;
```

Table crée.

```
SQL> CREATE TABLE Operation OF Type_Operation
  2  (
  3      PRIMARY KEY (NumOperation),
  4      FOREIGN KEY (Operation_Compte) REFERENCES Compte,
  5      CONSTRAINT ck_nature_operation CHECK (NatureOp IN ('Credit', 'Debit'))
  )
  6  ;
```

Table créée.

```
SQL>
SQL> CREATE TABLE Pret OF Type_Pret
  2  (
  3      PRIMARY KEY (NumPret),
  4      FOREIGN KEY (Pret_Compte) REFERENCES Compte,
  5      CONSTRAINT ck_type_pret CHECK (TypePret IN ('Vehicule', 'Immobilier',
  'ANSEJ', 'ANJEM'))
  6  );
```

Table créée.

4. Création des Instances dans les Tables

8. Remplir toutes les tables par des instances (6 succursales, 25 agences réparties sur les succursales, 100 clients dont 40 ont effectué des prêts auprès de différentes agences rattachées à différentes succursales et un nombre considérable d'opérations de débit/crédit sur les différents comptes)

Respecter les contraintes entre attributs

Les valeurs des attributs doivent être sensées et significatives

Numérotez les succursales par 001, 002, ...

et les agences par 101, 102, 103, ...

Les numéros de comptes sont sur 10 chiffres et commencent par un numéro d'agence par exemple 1180005564. Les numéros des clients sont sur 5 chiffres.

6 Succursales :

```
SQL> INSERT INTO Succursale
  2  VALUES
  3  (
  4    Type_Succ
  5    (
  6      001, 'Succursale1', 'Adresse 1 Nord', 'Nord', t_set_ref_Agence()
  7    )
  8  );
1 ligne créée.
```

```
SQL> SELECT COUNT(*) FROM Succursale;
          COUNT(*)
-----
          6
```

25 Agences :

```
SQL> INSERT INTO Agence
  2  VALUES
  3    (
  4      Type_Agence
  5      (
  6          101,
  7          'Agence 1',
  8          'Adresse Agence 1',
  9          'Principale',
 10          (SELECT REF(s) FROM Succursale s WHERE s.NumSucc = 001),
 11          t_set_ref_Compte()
 12      )
 13    )
 14 ;
1 ligne crée.
```

```
SQL> SELECT COUNT(*) FROM Agence;
COUNT(*)
-----
25
```

100 Clients :

```
SQL> INSERT INTO Client
  2  VALUES
  3    (
  4      Type_Client
  5      (
  6          10001,
  7          'Client 1',
  8          'Particulier',
  9          'Adresse Client 1',
 10          '051 234 567',
 11          'client1@email.com',
 12          t_set_ref_Compte()
 13      )
 14    )
 15 ;
1 ligne créée.
```

```
SQL> SELECT COUNT(*) FROM Client;
COUNT(*)
-----
100
```

141 Comptes :

```
SQL> INSERT INTO Compte
  2  VALUES
  3  (
  4    Type_Compte
  5    (
  6      1010000001,
  7      '01/01/2000',
  8      'Actif',
  9      100.00,
 10      (SELECT REF(a) FROM Agence a WHERE a.NumAgence = 101),
 11      (SELECT REF(c) FROM Client c WHERE c.NumClient = 10001),
 12      t_set_ref_Operation(),
 13      t_set_ref_Pret()
 14    )
 15  );
1 ligne crée.
```

```
SQL> select count(*) from Compte;
          COUNT(*)
-----
        141
```

41 Prêts :

```
SQL>
SQL> INSERT INTO Pret
  2  VALUES
  3  (
  4    Type_Pret
  5    (
  6      1,
  7      3000.00,
  8      '11/09/2001',
  9      '2 ans',
 10      'Vehicule',
 11      0.10,
 12      3150.00,
 13      (SELECT REF(c) FROM Compte c WHERE c.NumCompte = 1090000138)
 14    )
 15  );
1 ligne créée.
```

```
SQL> select count(*) from Pret;
          COUNT(*)
-----
        41
```

Un nombre considérable d'opérations de débit/crédit sur les différents comptes :

30 Opération :

```
SQL> INSERT INTO Operation
  2  VALUES
  3  (
  4    Type_Operation
  5    (
  6      1,
  7      'Debit',
  8      300.00,
  9      '01/06/2023',
 10      'Operation non Justifiée',
 11      (SELECT REF(c) FROM Compte c WHERE c.NumCompte = 1180005564)
 12    )
 13 );
```

1 ligne crée.

```
SQL> SELECT COUNT(*) FROM Operation;
```

COUNT(*)
50

Insertion dans les tables imbriquées :

Succursale_Agence :

```
SQL>
SQL> INSERT INTO TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s.numSucc=1) VALUES ((SELECT REF(a) FROM Agence a WHERE a.NumAgence=101));
1 ligne créée.
```

```

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=1);

      COUNT(*)
-----
      5

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=2);

      COUNT(*)
-----
      4

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=3);

      COUNT(*)
-----
      4

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=4);

      COUNT(*)
-----
      4

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=5);

      COUNT(*)
-----
      4

SQL> SELECT COUNT(*) FROM TABLE (SELECT s.Succursale_Agence FROM Succursale s WHERE s
.NumSucc=6);

      COUNT(*)
-----
      4

```

En tout nous avons les 25 références des 25 agences.

Note : Nous faisons la même chose pour les autres tables imbriquées.

Agence_Compte :

```

SQL> INSERT INTO TABLE (SELECT s.Agence_Compte FROM Agence s WHERE s.numAgence=
123) VALUES ((SELECT REF(c) FROM Compte c WHERE c.NumCompte=1230000100));

1 ligne crée.

```

Nous avons inséré les références de 141 Comptes pour « Agence » (voir l'annexe : SAYOUD_BOULKABOUL.sql).

Client_Compte :

```
SQL> INSERT INTO TABLE (SELECT s.Client_Compte FROM Client s WHERE s.numClient=10001) VALUES ((SELECT REF(c) FROM Compte c WHERE c.NumCompte=1010000001));  
1 ligne crée.
```

Les références de 141 Comptes pour « Client ».

Compte_Pret :

```
SQL> INSERT INTO TABLE (SELECT s.Compte_Pret FROM Compte s WHERE s.numCompte=1010000001) VALUES ((SELECT REF(c) FROM Pret c WHERE c.NumPret=1));  
1 ligne crée.
```

Les références de 41 Prêts pour « Compte ».

Compte_Operation :

```
SQL> INSERT INTO TABLE (SELECT s.Compte_Operation FROM Compte s WHERE s.NumCompte=1110000111) VALUES ((SELECT REF(c) FROM Operation c WHERE c.NumOperation=1))  
;  
1 ligne créée.
```

Les références de 50 Opérations pour « Compte ».

5. Langage d'Interrogation de Données

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

Prenons pour exemple l'agence dont le NumAgence est : 101. Dans cette requête nous récupérons tous les numéros de compte qui appartiennent à une agence spécifique, et dont les propriétaires sont des entreprises. Nous utilisons la table de référence (Agence_Compte) entre les tables Agence et Compte pour récupérer les NumComptes tout en vérifiant que ces comptes sont vraiment de l'agence 101.

```
SQL> select deref(value (c)).NumCompte
  2  FROM Agence a, table(a.Agence_Compte) c
  3  WHERE a.NumAgence = 101;

DREF(VALUE(C)).NUMCOMPTE
-----
1010000001
1010000026
1010000051
1010000076
1010000101
1010000126
1010000130

7 lignes sÚlectionnÚes.
```

10. Lister les prêts effectués auprès des agences rattachées à une succursale donnée (numSuccursale = 005). Préciser NumPrêt, NumAgence, numCompte et MontantPrêt)

Dans cette requête nous listons les prêts effectués auprès des agences rattachées à une succursale donnée (numSuccursale = 005) en affichant pour chaque prêt : le numéro de prêt (NumPrêt), le numéro de l'agence (NumAgence), le numéro de compte (NumCompte), et le montant du prêt (MontantPrêt). Nous utilisons les tables imbriquées entre les tables Succursale, Agence, Compte et Prêt tout en vérifiant que seuls les prêts associés à l'agence de la succursale 005 sont affichés.

```

SQL> SELECT deref(value(p)).NumPret, deref(value(a)).NumAgence, deref(value(c)).NumCompte, deref(value(p)).montantpret
  2  FROM Succursale s, table(s.Succursale_Agence) a, table(deref(value(a)).Agence_Compte) c, table(deref(value(c)).Compte_Pret) p
  3  WHERE s.NumSucc = 005;

DREF(VALUE(P)).NUMPRET DREF(VALUE(A)).NUMAGENCE DREF(VALUE(C)).NUMCOMPTE
-----
DREF(VALUE(P)).MONTANTPRET
-----
          7           105           1050000030
          8000

          13           105           1050000080
          3000

          33           105           1050000105
          5000

DREF(VALUE(P)).NUMPRET DREF(VALUE(A)).NUMAGENCE DREF(VALUE(C)).NUMCOMPTE
-----
DREF(VALUE(P)).MONTANTPRET
-----
          16           111           1110000086
          4000

          41           111           1110005564
          2200

          22           117           1170000092
          2000

DREF(VALUE(P)).NUMPRET DREF(VALUE(A)).NUMAGENCE DREF(VALUE(C)).NUMCOMPTE
-----
DREF(VALUE(P)).MONTANTPRET
-----
          28           123           1230000100
          4000

```

11. Quels sont les comptes sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022 ?

Dans cette requête nous récupérons les numéros de compte sur lesquels aucune opération de débit n'a été effectuée entre 2000 et 2022. Nous utilisons le NOT EXISTS pour vérifier s'il n'y a pas d'opérations de débit à chaque compte dans la table des opérations entre 2000 et 2022.

Si nous ne trouvons aucune opération de débit, nous sélectionnons le numéro de compte et nous l'affichons.

```
SQL> SELECT c.NumCompte
  2  FROM Compte c
  3 WHERE NOT EXISTS
  4  (
  5      SELECT *
  6        FROM table (c.Compte_Operation) o
  7       WHERE value(o).NatureOp = 'Debit'
  8       AND value(o).DateOp BETWEEN '01/01/2000' AND '31/12/2022'
  9  );
```

NUMCOMPTE
1150000090
1110000061
1240000074
1100000128
1150000065
1190000094
1110005564
1060000006
1220000022
1220000047
1110000086

NUMCOMPTE
1090000138
1070000032
1210000134
1010000001
1020000052
1170000067
1220000097
1250000025
1160000041
1230000123
1080000008

NUMCOMPTE
1200000020

SQL Plus	SQL Plus	SQL Plus
NUMCOMPTE	NUMCOMPTE	NUMCOMPTE
-----	-----	-----
1200000020	1040000029	1140000039
1030000053	1120000062	1220000072
1100000085	1080000108	1130000113
1210000096	1240000024	1060000081
1010000130	1200000045	1250000125
1090000109	1180000068	1040000004
1210000121	1010000101	1210000021
1250000050	1120000112	1240000049
1230000100	1220000122	1050000005
1100000010	1170000042	1010000126
1170000017	1190000019	1050000105
NUMCOMPTE	NUMCOMPTE	NUMCOMPTE
-----	-----	-----
1190000069	1210000071	1140000114
1170000092	1080000083	1080000058
1150000015	1010000051	1030000078
1160000016	1090000059	1020000131
1130000088	1140000089	1070000107
1090000009	1020000027	1140000064
1210000046	1230000073	1140000014
1070000057	1250000075	1060000031
1080000129	1230000023	1150000040
1240000136	1230000099	1200000070
1010000026	1240000124	1020000077
NUMCOMPTE	NUMCOMPTE	NUMCOMPTE
-----	-----	-----
1120000037	1020000002	1180000018
1050000030	1130000038	1030000028
1030000132	1040000054	1180000043
1160000091	1060000106	1040000104
1090000034	1230000098	1200000120
1230000135	1200000127	1090000084
1100000139	1110000036	1180005564
1080000033	1120000087	1030000003
1100000035	1030000103	1120000012
1130000013	1130000063	1010000076
1160000066	1070000007	1040000079

SQL Plus
NUMCOMPTE

1190000044
1180000093
1230000048
1110000111
1200000133
1050000080
1080000137
1110000011
1050000055
1070000082
1020000102
NUMCOMPTE

1060000056
1100000110
1100000060
1200000095
136 lignes sÚlectionnÙes.
SQL>

12. Quel est le montant total des crédits effectués sur un compte (numCompte = 1180005564) en 2023 ?

Dans cette requête nous calculons le montant total des crédits effectués sur le compte (NumCompte = 1180005564) au cours de l'année 2023. Nous utilisons la table de références imbriquées Compte_opérations pour filtrer les opérations de crédit effectuées sur ce compte pendant l'année 2023. Ensuite, nous sommes le tout pour obtenir le total des montants des opérations de crédit.

```
SQL> SELECT SUM(deref(value(o)).MontantOp)
  2  FROM Compte c, TABLE (c.Compte_Operation) o
  3  WHERE NumCompte = 1180005564
  4  AND value(o).NatureOp = 'Credit'
  5  AND value(o).DateOp BETWEEN '01/01/2023' AND '01/01/2024';

SUM(DEREF(VALUE(O)).MONTANTOP)
-----
55
```

13. Quels sont les prêts non encore soldés à ce jour ? Préciser NumPrêt, NumAgence, numCompte, numClient et MontantPrêt)

Dans cette requête, nous récupérons les numéros de prêt, les numéros d'agence, les numéros de compte, les numéros de client (nom de client aussi pour une meilleure visualisation des données) et le montant du prêt pour chaque prêt non encore soldé à ce jour. Nous filtrons les prêts non soldés en vérifiant que le montant de l'échéance est différent de zéro tout, que les comptes clients sont associés aux prêts non soldés (vue que la table client n'a pas de table de référence directe avec Prêt) avec les numéros de compte correspondants.

Remarque :

La majorité des Prêts insérés dans la table « Prêt » sont des prêts non soldés, ce qui justifie le fait d'avoir un total de 40 prêts non soldés après l'exécution de la requête suivante :

SQL>

```

SQL> SELECT deref(value(p)).NumPret,
  2      a.NumAgence,
  3      deref(value(c)).NumCompte,
  4      l.NumClient,
  5      l.NomClient,
  6      deref(value(p)).MontantPret
  7  FROM Agence a,
  8      Client l,
  9      TABLE(a.Agence_Compte) c,
 10     TABLE(deref(value(c)).Compte_Pret) p,
 11     TABLE(l.Client_Compte) x
 12 WHERE value(x).NumCompte = value(c).NumCompte
 13 AND value(p).MontantEcheance <> 0;

```

	DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMpte	NUMCLIENT	
NOMCLIENT					DREF(VALUE(P)).MONTANTPRET
-----	-----	-----	-----	-----	-----
Client 2	2	102	1020000002	10002	6000
Client 3	3	104	1040000004	10003	5000
Client 4	4	110	1100000010	10004	4000

	DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMpte	NUMCLIENT	
NOMCLIENT					DREF(VALUE(P)).MONTANTPRET
-----	-----	-----	-----	-----	-----
Client 5	5	116	1160000016	10005	3000
Client 6	6	120	1200000020	10006	2000
Client 7	7	105	1050000030	10007	8000

SQL Plus

DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 8	8	119	1190000044	10008 9000
Client 9	9	104	1040000054	10009 10000
Client 10	10	120	1200000070	10010 8000
<hr/>				
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 11	11	122	1220000072	10011 9000
Client 12	12	124	1240000074	10012 5000
Client 13	13	105	1050000080	10013 3000
<hr/>				
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 14	14	107	1070000082	10014 2000
Client 15	15	109	1090000084	10015 6000
Client 16	16	111	1110000086	10016 4000

SQL Plus

DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 17	17	112	1120000087	10017 5000
Client 18	18	113	1130000088	10018 2000
Client 19	19	114	1140000089	10019 1000
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 20	20	115	1150000090	10020 10000
Client 21	21	116	1160000091	10021 5000
Client 22	22	117	1170000092	10022 2000
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 23	23	118	1180000093	10023 8000
Client 24	24	119	1190000094	10024 9000
Client 25	25	120	1200000095	10025 4000

SQL Plus

DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	DREF(VALUE(P)).MONTANTPRET
NOMCLIENT				
Client 26	26	121	1210000096	10026 6000
Client 27	27	122	1220000097	10027 2000
Client 1	1	101	1010000001	10001 3000
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	DREF(VALUE(P)).MONTANTPRET
NOMCLIENT				
Client 31	29	101	1010000101	10031 3000
Client 32	30	102	1020000102	10032 4000
Client 33	31	103	1030000103	10033 2000
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	DREF(VALUE(P)).MONTANTPRET
NOMCLIENT				
Client 34	32	104	1040000104	10034 10000
Client 35	33	105	1050000105	10035 5000
Client 36	34	106	1060000106	10036 7000

SQL Plus

DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 37	35	107	1070000107	10037 8000
Client 38	36	108	1080000108	10038 9000
Client 39	37	109	1090000109	10039 1000
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 40	38	110	1100000110	10040 2000
Client 40	39	120	1200000120	10050 3000
Client 37	41	111	1110005564	10037 2200
DREF(VALUE(P)).NUMPRET	NUMAGENCE	DREF(VALUE(C)).NUMCOMPTE	NUMCLIENT	
NOMCLIENT				DREF(VALUE(P)).MONTANTPRET
Client 4	40	118	1180005564	10004 5000

40 lignes sélectionnées.

14. Quel est le compte le plus mouvementé en 2023 ? (Max d'opérations de débit/crédit)

Dans cette requête nous récupérerons les numéros de compte avec le nombre total d'opérations effectuées pendant l'année 2023, ensuite nous utilisons la fonction RANK() pour attribuer un classement à chaque compte selon le nombre d'opérations. Enfin, nous sélectionnons le premier premier compte dans le classement (celui qui a effectué le plus grand nombre d'opérations).

```
SQL> SELECT NumCompte
  2  FROM (
  3      SELECT NumCompte,
  4             COUNT(*) AS Nombre_Op,
  5             RANK() OVER (ORDER BY COUNT(*) DESC) AS Classement
  6      FROM (
  7          SELECT c.NumCompte
  8              FROM Compte c, TABLE(c.Compte_Operation) o
  9                 WHERE value(o).DateOp BETWEEN '01/01/2023' AND '01/01/2024'
 10      )
 11     GROUP BY NumCompte
 12  )
 13 WHERE Classement = 1;

NUMCOMPTE
-----
1180005564
```

Conclusion

Dans la **modélisation relationnelle**, nous n'avons pas la possibilité de définir les différents concepts d'héritage, ses types sont simples (NUMBER, VARCHAR, etc.) et il sépare entre les méthodes et les données ce qui rend la tâche d'interroger la base de données complexes car nous aurons besoin d'un autre langage pour créer des méthodes. **L'orienté objet** par contre, nous donne la possibilité de définir pour chaque classe les différentes méthodes dont elle a besoin. Nous constatons alors que le passage du relationnel vers l'Orienté Objet est principalement pour profiter des concepts liés à la modélisation orienté objet (héritage, polymorphisme, etc.) et la possibilité d'utiliser des types plus complexes pour une meilleure gestion des données.

Partie II

NoSQL – Modèle orienté « documents »

1. Modélisation Orientée Document

On suppose que la plupart des requêtes sur la base vont porter sur les prêts (voir exemples de requêtes plus bas).

- Proposer une modélisation orientée document de la base de données décrite dans la partie I, dans ce cas.

```
{  
    "NumPrêt": ,  
    "montantPrêt": ,  
    "dateEffet": ,  
    "durée": ,  
    "typePrêt": ,  
    "tauxIntérêt": ,  
    "montantEchéance": ,  
    "NumCompte": ,  
    "compte": {  
        "NumCompte": ,  
        "dateOuverture": ,  
        "étatCompte": ,  
        "Solde": ,  
        "NumClient": ,}  
    "client": {  
        "NumClient": ,  
        "NomClient": ,  
        "TypeClient": ,  
        "AdresseClient": ,  
        "NumTel": ,  
        "Email":  
        "NumAgence": }  
    "agence": {  
        "NumAgence": ,  
        "nomAgence": ,  
        "adresseAgence": ,  
        "catégorie": ,  
        "NumSucc": }  
    "succursale": {  
        "NumSucc": ,  
        "nomSucc": ,
```

```

        "adresseSucc": ,
        "région": 
    }
}

```

- Illustriez votre modélisation sur un exemple (ou plus) de la BD que vous avez générée

```
{
    "NumPrêt": 1,
    "montantPrêt": 10000,
    "dateEffet": "2020-01-01",
    "durée": 36,
    "typePrêt": "Véhicule",
    "tauxIntérêt": 0.05,
    "montantEchéance": 350,
    "NumCompte": 1180005564,
    "compte": {
        "NumCompte": 1180005564,
        "dateOuverture": "2020-01-01",
        "étatCompte": "Actif",
        "Solde": 10000,
        "NumClient": 1,
        "client": {
            "NumClient": 1,
            "NomClient": "AMIRA",
            "TypeClient": "Particulier",
            "AdresseClient": " bab el oued , alger centre ",
            "NumTel": "0123456789",
            "Email": "dupont@example.com"
        },
        "NumAgence": 101
    },
    "agence": {
        "NumAgence": 101,
        "nomAgence": "Agence principale Nord",
        "adresseAgence": " hydra , alger centre",
        "catégorie": "Principale",
        "NumSucc": 1
    },
    "succursale": {
        "NumSucc": 1,
        "nomSucc": "Succursale Nord",
        "adresseSucc": "hydra , alger centre"
    }
}
```

```
        "adresseSucc": "10 rue dely brahim, alger centre",
        "région": "Nord"
    }
}
```

- Justifiez votre choix de conception

La conception orientée document de la base de données, axée sur les prêts, est justifiée par plusieurs facteurs. Tout d'abord, elle répond à la nature des requêtes centrées sur les prêts, ce qui facilite la récupération rapide et efficace des informations pertinentes. En structurant les données autour de l'entité principale des prêts, elle permet une analyse plus aisée des relations et des dépendances avec les comptes, les clients, les agences et les succursales

(les comptes identifiés par NumCompte et liés à NumClient et NumAgence, les clients identifiés par NumClient, les agences identifiées par NumAgence et liées à NumSucc, et enfin les succursales identifiées par NumSucc).

Cette approche optimise la lecture des données, en particulier lors de l'analyse des prêts et de leur contexte, en les organisant de manière logique et cohérente. De plus, cette conception est évolutive, capable de prendre en charge de nouveaux types de prêts ou de fonctionnalités additionnelles sans modifications majeures. En simplifiant la compréhension et la maintenance de la base de données.

- Quelles sont les inconvénients de votre conception

Une redondance des données, surtout lorsque des informations sont répétées dans plusieurs documents, ce qui peut augmenter la taille de la base de données et compliquer la gestion des données.

La mise à jour de données communes à plusieurs documents peut être complexe et propice aux erreurs.

La flexibilité des requêtes peut également être limitée, en particulier pour les requêtes traversant plusieurs niveaux de la hiérarchie documentaire.

La modélisation des données peut être plus complexe que dans un modèle relationnel classique, nécessitant une expertise approfondie.

La gestion des données sous forme de documents peut consommer davantage de ressources, notamment en termes de stockage et de traitement, surtout avec des bases de données volumineuses.

2. Remplir la base de données

(via un script, ajouter d'autres données afin d'augmenter le volume de la base)

- Pour commencer le remplissage, il est nécessaire de se connecter à MongoDB en utilisant la commande suivante : mongo

```
C:\Users\Home>mongo
MongoDB shell version: 2.6.4
connecting to: test
Error while trying to show server startup warnings: Unsupported OP_QUERY command: getLog. The client driver may require
an upgrade. For more details see https://dochub.mongodb.org/core/legacy-opcode-removal
>
```

La connexion à MongoDB est réussie, comme indiqué par le message "connecting to: test".

- Pour créer une base de données dans MongoDB, il suffit d'utiliser la commande "use" suivie du nom de la base de données souhaitée.

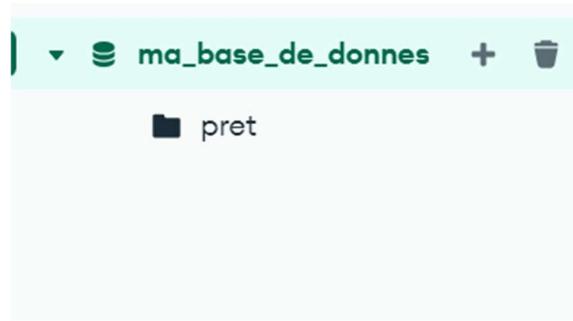
```
> use ma_base_de_donnees
switched to db ma_base_de_donnees
> -
```

- création de la collection :

La décision de regrouper toutes les données relatives aux prêts dans une seule collection, pret, découle de la structure imbriquée de notre modèle. Cette approche simplifie la gestion des données en les centralisant, ce qui facilite les requêtes et l'analyse ultérieure. Chaque document dans cette collection contient non seulement les détails du prêt, mais aussi les informations sur le compte associé, le client, l'agence et la succursale. Cette conception cohérente et structurée permet une gestion efficace des prêts bancaires dans notre système.

```
ma_base_de_donnees> db.createCollection("pret")
{ ok: 1 }
ma_base_de_donnees>
```

La collection "pret" a été créée avec succès dans la base de données et aussi on peut vérifier les créations :



insertion de différentes données :

```

acknowledged: true,
insertedIds: {
  '0': ObjectId('66467988e73f7c6b5846b799'),
  '1': ObjectId('66467988e73f7c6b5846b79a'),
  '2': ObjectId('66467988e73f7c6b5846b79b')
}
}

ma base de donnees>

{
  "succursale": {
    "adresseSucc": "Cité 20 Août, Bab Ezzouar, Alger",
    "région": "Est"
  }
}
]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66467b30e73f7c6b5846b79c'),
    '1': ObjectId('66467b30e73f7c6b5846b79d'),
    '2': ObjectId('66467b30e73f7c6b5846b79e'),
    '3': ObjectId('66467b30e73f7c6b5846b79f')
  }
}

{
  succursale : [
    {
      "NumSucc": 2,
      "nomSucc": "Succursale Est",
      "adresseSucc": "Cité 20 Aout, Bab Ezzouar, Alger",
      "région": "Est"
    }
  ]
};

{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66467d15e73f7c6b5846b7a0'),
    '1': ObjectId('66467d15e73f7c6b5846b7a1'),
    '2': ObjectId('66467d15e73f7c6b5846b7a2')
  }
}

{
  "adresseSucc": "10 rue dely brahim, Alger centre",
  "région": "Nord"
}
]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66467d76e73f7c6b5846b7a3'),
    '1': ObjectId('66467d76e73f7c6b5846b7a4'),
    '2': ObjectId('66467d76e73f7c6b5846b7a5')
  }
}

```

3. Répondre aux requêtes

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

```
ma_base_de_donnees> db.prets.find({"agence.NumAgence": 102})
```

Cette requête recherche tous les documents dans la collection prets où le champ agence.NumAgence est égal à 102, ce qui correspond à l'agence de numéro 102.résultats :

Quelques captures du le résultat :

```
ma_base_de_donnees> db.prets.find({"agence.NumAgence": 102})
[
  {
    _id: ObjectId('66467988e73f7c6b5846b79b'),
    'NumPrêt': 5,
    'montantPrêt': 20000,
    dateEffet: '2020-05-10',
    'durée': 60,
    'typePrêt': 'Immobilier',
    'tauxIntérêt': 0.035,
    'montantEchéance': 400,
    NumCompte: 1180005568,
    compte: {
      NumCompte: 1180005568,
      dateOuverture: '2020-04-10',
      'étatCompte': 'Actif',
      Solde: 10000,
      NumClient: 5
    },
    client: {
      NumClient: 5,
      NomClient: 'Lina',
      TypeClient: 'Particulier',
      AdresseClient: 'Hydra, Alger',
      NumTel: '0312345678',
      Email: 'lina@example.com',
      NumAgence: 102
    },
    agence: {
      NumAgence: 102, ←
      nomAgence: 'Agence centrale Est',
      adresseAgence: 'Bab Ezzouar, Alger',
      'catégorie': 'Principale',
      NumSucc: 2
    },
    succursale: {
      NumSucc: 2,
      nomSucc: 'Succursale Est',
    }
  }
]
```

```
{  
    _id: ObjectId('66467b30e73f7c6b5846b79c'),  
    'NumPrêt': 6,  
    'montantPrêt': 12000,  
    dateEffet: '2020-03-10',  
    'durée': 48,  
    'typePrêt': 'Consommation',  
    'tauxIntérêt': 0.06,  
    'montantEchéance': 300,  
    NumCompte: 1180005569,  
    compte: {  
        NumCompte: 1180005569,  
        dateOuverture: '2020-02-10',  
        'étatCompte': 'Actif',  
        Solde: 2000,  
        NumClient: 6  
    },  
    client: {  
        NumClient: 6,  
        NomClient: 'Hakim',  
        TypeClient: 'Particulier',  
        AdresseClient: 'Bab El Oued, Alger',  
        NumTel: '0312345678',  
        Email: 'hakim@example.com',  
        NumAgence: 102  
    },  
    agence: {  
        NumAgence: 102, ←  
        nomAgence: 'Agence centrale Est',  
        adresseAgence: 'Bab Ezzouar, Alger',  
        'catégorie': 'Principale',  
        NumSucc: 2  
    },  
    succursale: [  
        NumSucc: 2,  
        nomSucc: 'Succursale Est',  
        adresseSucc: 'Cité 28 Août - Bab Ezzouar - Alg'.
```

```

_id: ObjectId('66467b30e73f7c6b5846b79f'),
'NumPrêt': 9,
'montantPrêt': 30000,
dateEffet: '2020-08-15',
'durée': 60,
'typePrêt': 'Études',
'tauxIntérêt': 0.02,
'montantEchéance': 500,
NumCompte: 1180005572,
compte: {
  NumCompte: 1180005572,
  dateOuverture: '2020-07-15',
  'étatCompte': 'Actif',
  Solde: 20000,
  NumClient: 9
},
client: {
  NumClient: 9,
  NomClient: 'Sara',
  TypeClient: 'Particulier',
  AdresseClient: 'Bab Ezzouar, Alger',
  NumTel: '0312345678',
  Email: 'sara@example.com',
  NumAgence: 102
},
agence: {
  NumAgence: 102, ←
  nomAgence: 'Agence centrale Est',
  adresseAgence: 'Bab Ezzouar, Alger',
  'catégorie': 'Principale',
  NumSucc: 2
},
succursale: {
  NumSucc: 2,
  nomSucc: 'Succursale Est',
}
]}

_id: ObjectId('66467d15e73f7c6b5846b7a2'),
'NumPrêt': 12,
'montantPrêt': 30000,
dateEffet: '2021-03-20',
'durée': 60,
'typePrêt': 'Études',
'tauxIntérêt': 0.02,
'montantEchéance': 500,
NumCompte: 1180005575,
compte: {
  NumCompte: 1180005575,
  dateOuverture: '2021-02-20',
  'étatCompte': 'Actif',
  Solde: 15000,
  NumClient: 12
},
client: {
  NumClient: 12,
  NomClient: 'Karima',
  TypeClient: 'Particulier',
  AdresseClient: 'El Harrach, Alger',
  NumTel: '0312345678',
  Email: 'karima@example.com',
  NumAgence: 102
},
agence: {
  NumAgence: 102, ←
  nomAgence: 'Agence centrale Est',
  adresseAgence: 'Bab Ezzouar, Alger',
  'catégorie': 'Principale',
  NumSucc: 2
},
succursale: {
  NumSucc: 2,
  nomSucc: 'Succursale Est',
  adresseSucc: 'Cité 20 Août Bab Ezzouar'
}
]

```

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

Pour afficher tous les prêts effectués auprès des agences rattachées aux succursales de la région "Nord" et préciser les champs NumPrêt, NumAgence, NumCompte, NumClient et MontantPrêt, on utilise la requête suivante :

```

ma_base_de_donnees> db.prets.find({"succursale.région": "Nord"}, {"NumPrêt": 1, "agence.NumAgence": 1, "NumCompte": 1, "client.NumClient": 1, "montantPrêt": 1})
[
```

Cette requête recherche tous les documents dans la collection prets où le champ succursale.région est égal à "Nord". Elle projette également les champs NumPrêt, agence.NumAgence, NumCompte, client.NumClient et montantPrêt pour l'affichage.

Résultat :

```

ma_base_de_donnees> db.prets.find({ "succursale.région": "Nord"}, { "NumPrêt": 1, "agence.NumAgence": 1, "NumCompte": 1, "client.NumClient": 1, "montantPrêt": 1 })
[ {
    "_id": ObjectId('66467d76e73f7c6b5846b7a5'),
    "NumPrêt": 15,
    "montantPrêt": 30000,
    "NumCompte": 1180005578,
    "client": { "NumClient": 15 },
    "agence": { "NumAgence": 101 }
}
]

```

- 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.

Pour réaliser cette opération, nous pouvons utiliser l'agrégation MongoDB pour regrouper les prêts par agence, calculer le nombre total de prêts par agence, trier les résultats par ordre décroissant du nombre de prêts, puis insérer ces résultats dans une nouvelle collection.

```

< { ok: 1 }
> db.prets.aggregate([
    { $group: { _id: "$agence.NumAgence", nbPrêts: { $sum: 1 } } },
    { $sort: { nbPrêts: -1 } },
    { $out: "Agence-NbPrêts" }
]);
<
> db["Agence-NbPrêts"].find();
<

```

la commande db["Agence-NbPrêts"].find(); permet d'afficher le contenu de cette nouvelle collection.

```

> db["Agence-NbPrêts"].find();
< [
    {
        "_id": 102,
        "nbPrêts": 5
    },
    {
        "_id": 108,
        "nbPrêts": 4
    },
    {
        "_id": 105,
        "nbPrêts": 3
    },
    {
        "_id": 101,
        "nbPrêts": 1
    }
]

```

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

d'abord réer la collection Prêt-ANSEJ

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

pour faire la recherche et affiche on utilise la requête suivante ou 1 veut dire affiche cette attribue comme : NumPrêt": 1

```
> db["Prêt-ANSEJ"].find(
  { "typePrêt": "ANSEJ" },
  { "NumPrêt": 1, "client.NumClient": 1, "montantPrêt": 1, "dateEffet": 1 }
);
<
ma_base_de_donnees >
```

sinon pour avoir des données dans votre base de données qui correspondent au filtre "typePrêt": "ANSEJ" pour obtenir des résultats.

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

Pour afficher tous les prêts effectués par des clients de type « Particulier » avec les informations sur le client et le prêt, nous pouvons utiliser la requête suivante :

```
> db.prets.find(
  { "client.TypeClient": "Particulier" },
  { "client.NumClient": 1, "client.NomClient": 1, "NumPrêt": 1, "montantPrêt": 1 }
);
< {
  id: ObjectId('66467988e73f7c6b5846b799'),
```

Cette requête recherche tous les documents dans la collection prets où le champ client.TypeClient est égal à "Particulier", et elle projette les champs client.NumClient, client.NomClient, NumPrêt et montantPrêt pour l'affichage.

et voilà le résultat affiché :

```

[{"_id: ObjectId('66467b30e73f7c6b5846b79c'),
  'NumPrêt': 6,
  'montantPrêt': 12000,
  client: {
    NumClient: 6,
    NomClient: 'Hakim'
  }
},
{
  _id: ObjectId('66467b30e73f7c6b5846b79d'),
  'NumPrêt': 7,
  'montantPrêt': 18000,
  client: {
    NumClient: 7,
    NomClient: 'Fatima'
  }
},
{
  _id: ObjectId('66467b30e73f7c6b5846b79e'),
  'NumPrêt': 8,
  'montantPrêt': 25000,
  client: {
    NumClient: 8,
    NomClient: 'Ali'
  }
},
{
  _id: ObjectId('66467b30e73f7c6b5846b79f'),
  'NumPrêt': 9,
  'montantPrêt': 30000,
  client: {
    NumClient: 9,
    NomClient: 'Sara'
  }
},
{
  _id: ObjectId('66467d15e73f7c6b5846b7a0'),
  'NumPrêt': 10,
  'montantPrêt': 20000,
  client: {
    NumClient: 10,
    NomClient: 'Nadia'
  }
},
{
  _id: ObjectId('66467d15e73f7c6b5846b7a1'),
  'NumPrêt': 11,
  'montantPrêt': 25000,
  client: {
    NumClient: 11,
    NomClient: 'Omar'
  }
},
{
  _id: ObjectId('66467988e73f7c6b5846b799'),
  'NumPrêt': 2,
  'montantPrêt': 15000,
  client: {
    NumClient: 2,
    NomClient: 'Karim'
  }
},
{
  _id: ObjectId('66467988e73f7c6b5846b79a'),
  'NumPrêt': 4,
  'montantPrêt': 8000,
  client: {
    NumClient: 4,
    NomClient: 'Salim'
  }
},
{
  _id: ObjectId('66467988e73f7c6b5846b79b'),
  'NumPrêt': 5,
  'montantPrêt': 20000,
  client: {
    NumClient: 5,
    NomClient: 'Lina'
  }
}
]

```

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

```

> db.prets.updateMany(
  {
    "dateEffet": { $lt: "2021-01-01" },
    "montantEchéance": { $exists: true },
    "Solde": { $gt: 0 }
  },
  {
    $inc: { "montantEchéance": 2000 }
  }
);

```

Cette requête met à jour tous les documents dans la collection prets qui répondent aux critères suivants :

La date d'effet est antérieure à janvier 2021.

Le champ montantEchéance existe.

Le solde du prêt est supérieur à 0.

Elle incrémentera alors le champ montantEchéance de 2000 DA pour chacun de ces documents.

Le résultat :

```
    );
    < {
        acknowledged: true,
        insertedId: null,
        matchedCount: 0,
        modifiedCount: 0,
        upsertedCount: 0
    }
ma_base_de_donnees> |
```

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

Création de la fonction map :

La fonction map prend en entrée un document de la collection db.prets.

Elle émet une paire clé-valeur où la clé est l'identifiant de l'agence (agence.NumAgence) et la valeur est toujours 1. Cela permet de compter le nombre de prêts par agence.

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

Création de la fonction reduce

La fonction reduce prend en entrée une clé (identifiant de l'agence) et un tableau de valeurs.

Elle retourne la somme de toutes les valeurs du tableau, c'est-à-dire le nombre total de prêts pour cette agence

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

Appel de la fonction mapReduce :

La fonction mapReduce est appelée sur la collection db.prets avec les fonctions map et reduce spécifiées.

Les résultats sont stockés dans la collection Agence-NbPrêts.

```
        '',
    > db.prets.mapReduce(
        mapFunction,
        reduceFunction,
        { out: "Agence-NbPrêts" }
    );
```

Affichage des résultats :

Les résultats de la collection Agence-NbPrêts sont affichés pour montrer le nombre total de prêts par agence, triés par ordre décroissant.

- **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.**

Notre conception actuelle ne permet pas de répondre directement à cette requête car elle se concentre sur le comptage des prêts par agence et ne prend pas en compte les détails des opérations de crédit sur les comptes des clients. Pour répondre à cette requête, il faudrait avoir une collection distincte pour les opérations de crédit et un moyen de les lier aux clients et à leurs comptes.

Une conception plus complète pourrait inclure une collection d'opérations de crédit avec des informations telles que le montant de l'opération, la date, le type de transaction, ainsi que des références au compte et au client concernés. Ces informations permettraient ensuite de filtrer les opérations de crédit effectuées sur les comptes des clients de type « Entreprise » pendant l'année 2023.

4. Analyse

Donnez votre analyse de ces requêtes par rapport à la conception que vous avez proposée.

Pour éviter la redondance des données dans la modélisation orientée document par exemple on peut normaliser des données : Identifier les données répétitives et créez des collections distinctes pour ces données. Par exemple, au lieu de stocker les détails de l'agence dans chaque prêt, créez une collection distincte pour les agences et référez l'agence appropriée dans chaque prêt.

- Redondance des données: Les données des agences et des succursales sont incluses dans chaque prêt, ce qui entraîne une redondance. Selon la conception proposée, il est préférable de stocker les détails des agences et des succursales dans des collections distinctes et de les référencer dans les prêts.
- Normalisation des données : Pour éviter la redondance, il est important de normaliser les données en créant des collections distinctes pour les agences et les succursales, comme mentionné dans la conception proposée.
- Impact sur les requêtes : En normalisant les données, les requêtes devront être ajustées pour prendre en compte les références aux agences et aux succursales. Par exemple, pour afficher les détails d'une agence pour un prêt donné, il faudrait d'abord récupérer l'ID de l'agence à partir du prêt, puis effectuer une requête pour obtenir les détails de l'agence à partir de la collection des agences.
- Amélioration de la cohérence des données : En normalisant les données, il sera plus facile de maintenir la cohérence des données, car les détails des agences et des succursales ne seront stockés qu'une seule fois dans leurs collections respectives.

Conclusion

Dans ce projet, nous avons examiné deux approches distinctes de gestion de données : le modèle relationnel orienté objet et le modèle NoSQL orienté documents. Nous avons exploré la modélisation, l'implémentation et l'interrogation de bases de données avancées dans chaque cas. Chaque approche présente ses avantages et ses inconvénients. Ce travail nous a permis de mieux comprendre la gestion de données et d'améliorer nos compétences en modélisation et en interrogation de bases de données.