



République Algérienne Démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'informatique
Département d'informatique

Projet BDA

Bases de Données avancées

SQL3-Oracle et NoSQL (MongoDB)

Réalisé par :
OUCHAOU Chafaa
TAOUCI Kenza

Systèmes Informatiques Intelligents

Année scolaire : 2023/2024

Table des matières

TABLE OF CONTENTS	
CHAPITRE 1 Introduction	1
CHAPITRE 2 Partie I : Relationnel-Objet	2
2.1 Modélisation orientée objet	2
2.2 Création des TableSpaces et utilisateur	3
2.2.1 Explication	4
2.2.2 Capture exécution	4
2.3 Langage de définition de données	5
2.3.1 définir les types et les associations	5
2.3.2 Définir les méthodes	7
2.3.3 Définir les tables nécessaires à la base de données	17
2.4 Création des instances dans les tables	19
2.4.1 Succursales	19
2.4.2 Agences	20
2.4.3 Clients	21
2.4.4 Comptes	22
2.4.5 Operations	22
2.4.6 Prêts	24
2.4.7 Capture exécution	25

2.5	Langage d'interrogation de données	25
2.5.1	question 1	25
2.5.2	question 2	26
2.5.3	question 3	27
2.5.4	question 4	28
2.5.5	question 5	29
2.5.6	question 6	29
2.6	Capture d'exécution	31
2.6.1	Requête 1	31
2.6.2	Requête 2	31
2.6.3	Requête 3	31
2.6.4	Requête 4	32
2.6.5	Requête 5	32
2.6.6	Requête 6	32
CHAPITRE 3 Partie II : Non-Sql Mongodb		33
3.1	Modélisation orientée document	33
3.1.1	choix de la modélisation	33
3.1.2	exemple pour illustrer la base de données	33
3.1.3	Justificatif de la modélisation	35
3.1.4	inconvenients de la modélisation	36
3.2	Remplir la base de données	36
3.3	Les requêtes	39
3.3.1	Afficher tous prêts effectués auprès de l'agence de numéro 102	39

3.3.2	Afficher tous prêts effectués auprès des agences rattachées aux succursales de la région « Nord ». Préciser NumPrêt, NumAgence, numCompte, numClient et MontantPrêt. . . .	40
3.3.3	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.	40
3.3.4	Dans une collection Prêt-ANSEJ, récupérer tous les prêts liés à des dossiers ANSEJ. Préciser NumPrêt, numClient, MontantPrêt et dateEffet	41
3.3.5	Afficher toutes les prêts effectués par des clients de type « Particulier ». On affichera le NumClient, NomClient, NumPrêt, montantPrêt.	42
3.3.6	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.	42
3.3.7	Reprendre la 3 ème requête à l'aide du paradigme Map-Reduce	43
3.3.8	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.	43
3.4	Analyse	44

CHAPITRE 1

Introduction

Ce projet en Bases de Données Avancées (BDA) explore les fondements et les applications des systèmes de gestion de bases de données relationnelles-objet (Oracle) et non relationnelles (NoSQL), en mettant en lumière leurs caractéristiques distinctives ainsi que leurs cas d'utilisation. Le SQL-objet représente la dernière évolution des langages de requête structurée, offrant une gamme étendue de fonctionnalités pour la gestion et l'interrogation des données. D'autre part, NoSQL, représenté ici par MongoDB, propose une approche alternative, axée sur la scalabilité, la flexibilité de schéma et la gestion efficace de données non structurées. Dans cette étude, nous explorons les différences entre ces deux paradigmes de gestion de données, en mettant en lumière leurs forces et leurs faiblesses respectives. La première partie de ce projet se concentre sur la modélisation et l'implémentation dans un environnement SQL3 sous Oracle, tandis que la seconde partie explore les défis et les avantages de la modélisation dans un contexte NoSQL avec MongoDB. Ce projet offre ainsi une vision complète des stratégies de gestion de données dans un monde de plus en plus complexe et diversifié.

CHAPITRE 2

Partie I : Relationnel-Objet

2.1 Modélisation orientée objet

Dans cette section, nous allons présenter le diagramme de classe qui modélise notre base de données.

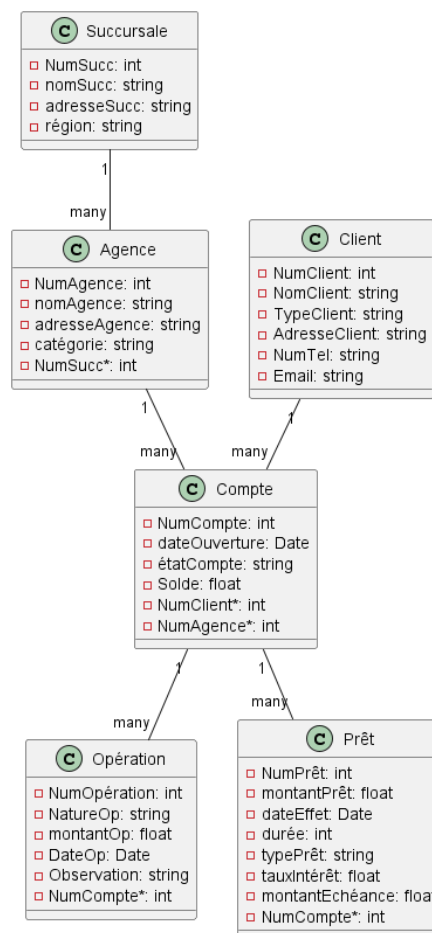


FIGURE 2.1 – Diagramme de classe

2.2 Création des TableSpaces et utilisateur

Dans cette section nous allons présenter les requêtes sql que nous avons utilisé pour la création des TableSpaces et des utilisateurs.

```
1 — Create tablespace SQL3_TBS
2 CREATE TABLESPACE SQL3_TBS
3 DATAFILE 'sql3_tbs.dat'
4 SIZE 100M — Size of the datafile is 100 MB
5 AUTOEXTEND ON — enables automatic extension
6 NEXT 100M — Automatically extend by 100 MB when needed
7 MAXSIZE UNLIMITED;
8
9 — Create temporary tablespace SQL3_TempTBS
10 CREATE TEMPORARY TABLESPACE SQL3_TempTBS
11 TEMPFILE 'sql3 temptbs.tmp'
12 SIZE 50M — Size of the tempfile is 50 MB
13 AUTOEXTEND ON — enables automatic extension
14 NEXT 50M — Automatically extend by 50 MB when needed
15 MAXSIZE UNLIMITED;
16
17 — Create user c##sql3
18 CREATE USER c##sql3 IDENTIFIED BY password123;
19
20 — Set default tablespace and temporary tablespace for user c##
  sql3
21 ALTER USER c##sql3 DEFAULT TABLESPACE SQL3_TBS TEMPORARY
  TABLESPACE SQL3_TempTBS;
22
23 — Grant all privileges to user c##sql3
24 GRANT ALL PRIVILEGES TO c##sql3;
25 —se connecter en tant que c##sql3
26 connect c##sql3/password123;
```

Listing 2.1 – le code SQL pour la création des TableSpaces et des utilisateurs

2.2.1 Explication

1. **c##sql3** : Dans Oracle version 12c ou ultérieur, le concept d'utilisateurs communs et d'utilisateurs locaux a été introduit. Un utilisateur commun est un utilisateur qui a le même nom d'utilisateur et la même authentification à travers plusieurs PDB (bases de données enfichables). Un utilisateur local est un utilisateur qui existe dans une seule PDB. Par défaut, Oracle ne permet pas de créer un utilisateur commun sans préfixe 'C##' ou 'c##'. Ainsi, pour créer un utilisateur commun, il est nécessaire de préfixer le nom d'utilisateur avec 'C##' ou 'c##'. Sinon, nous pouvons nous connecter à une PDB et créer un utilisateur local.

2.2.2 Capture exécution

```
SQL> CREATE TABLESPACE SQL3_TBS
2 DATAFILE 'sql3_tbs.dat'
3 SIZE 100M
4 AUTOEXTEND ON
5 NEXT 100M
6 MAXSIZE UNLIMITED;

TABLESPACE created.

SQL> CREATE TEMPORARY TABLESPACE SQL3_TempTBS
2 TEMPFILE 'sql3_temptbs.tmp'
3 SIZE 50M
4 AUTOEXTEND ON
5 NEXT 50M
6 MAXSIZE UNLIMITED;

TABLESPACE created.
```

FIGURE 2.2 – création des tablespaces

```
SQL> CREATE USER c##sql3 IDENTIFIED BY password123;

USER created.
```

FIGURE 2.3 – création de l'utilisateur

```
SQL> GRANT ALL PRIVILEGES TO c##sql3;

Grant succeeded.
```

FIGURE 2.4 – Donner tous les privilèges à cet utilisateur

2.3 Langage de définition de données

2.3.1 définir les types et les associations

Ici, nous présentons les différents types et associations créés que nous avons utilisés pour la création des tables.

```
1 CREATE OR REPLACE TYPE Succursale_Type ;
2 /
3 CREATE OR REPLACE TYPE Agence_Type AS OBJECT (
4     NumAgence INT ,
5     NomAgence VARCHAR2(100) ,
6     AdresseAgence VARCHAR2(255) ,
7     Categorie VARCHAR2(50) ,
8     NumSucc ref Succursale_Type ,
9     MAP MEMBER FUNCTION get_map RETURN VARCHAR2,
10    member FUNCTION total_amount_of_loans_for_agency (
11        agency_id IN INT,
12        start_date IN DATE,
13        end_date IN DATE
14    ) RETURN NUMBER
15 );
16 /
17
18 CREATE or replace TYPE tset_ref_agence AS TABLE OF REF
19     Agence_Type;
20 /
21 CREATE OR REPLACE TYPE Succursale_Type AS OBJECT (
22     NumSucc INT ,
23     NomSucc VARCHAR2(100) ,
24     AdresseSucc VARCHAR2(255) ,
25     Region VARCHAR2(50) ,
26     agences tset_ref_agence ,
27     MEMBER FUNCTION count_main_agencies RETURN NUMBER
28
```

```

29 );
30 /
31
32 CREATE OR REPLACE TYPE Client_Type AS OBJECT (
33     NumClient INT,
34     NomClient VARCHAR2(100),
35     TypeClient VARCHAR2(50),
36     AdresseClient VARCHAR2(255),
37     NumTel VARCHAR2(20),
38     Email VARCHAR2(100)
39 );
40 /
41
42 CREATE OR REPLACE TYPE Compte_Type;
43 /
44
45 CREATE OR REPLACE TYPE Operation_Type AS OBJECT (
46     NumOperation INT,
47     NatureOp VARCHAR2(20),
48     MontantOp NUMBER(10, 2),
49     DateOp DATE,
50     Observation VARCHAR2(255),
51     NumCompte REF Compte_Type
52 );
53 /
54
55 CREATE OR REPLACE TYPE Pret_Type AS OBJECT (
56     NumPret INT,
57     MontantPret NUMBER(10, 2),
58     DateEffet DATE,
59     Duree INT,
60     TypePret VARCHAR2(50),
61     TauxInteret NUMBER(5, 2),
62     MontantEcheance NUMBER(10, 2),
63     NumCompte REF Compte_Type,

```

```

64     MEMBER FUNCTION agences_Secondary_ANSEJ RETURN NUMBER
65 );
66 /
67
68 Create type tset_ref_pret as table of ref Pret_Type;
69 /
70 Create or REPLACE type tset_ref_operation as table of ref
    Operation_Type;
71 /
72
73 CREATE OR REPLACE TYPE Compte_Type AS OBJECT (
74     NumCompte INT,
75     DateOuverture DATE,
76     EtatCompte VARCHAR2(20) ,
77     Solde NUMBER(10, 2) ,
78     NumClient REF Client_Type ,
79     NumAgence REF Agence_Type ,
80     Operations tset_ref_operation ,
81     Prets tset_ref_pret ,
82     STATIC FUNCTION number_of_loans_for_each_agency RETURN
        NUMBER
83 );
84 /

```

Listing 2.2 – Définition des types et associations SQL

2.3.2 Définir les méthodes

Dans cette partie, nous allons présenter les fonctions permettant de réaliser les actions demandées.

1 :Calculer pour chaque agence le nombre de prêts effectuées

La fonction **number_of_loans_for_each_agency** calcule le nombre total de prêts pour chaque agence et renvoie le nombre total de prêts pour toutes les agences.

Voici une explication détaillée de la fonction :

1. Un curseur **agency_cursor** est déclaré pour sélectionner les agences distinctes de la table **Comptes**.
2. Deux variables **loan_count** et **total_loan_count** sont déclarées pour garder une trace du nombre de prêts pour chaque agence et du nombre total de prêts pour toutes les agences, respectivement.
3. La fonction entre ensuite dans une boucle, itérant sur chaque agence récupérée par le curseur **agency_cursor**.
4. Pour chaque agence, elle exécute une requête **SELECT COUNT(*)** pour compter le nombre de prêts associés à l'agence actuelle. Cela est fait en sélectionnant les prêts de la table **Comptes** où l'agence correspond à l'agence actuelle.
5. Le nombre de prêts pour l'agence actuelle est ensuite imprimé à l'aide de la procédure **DBMS_OUTPUT.PUT_LINE**.
6. Le **loan_count** pour l'agence actuelle est ajouté au **total_loan_count**.
7. Après que la boucle ait itéré sur toutes les agences, la fonction renvoie le **total_loan_count**, qui représente le nombre total de prêts pour toutes les agences.

```
1 STATIC FUNCTION number_of_loans_for_each_agency RETURN NUMBER
  IS
2     CURSOR agency_cursor IS
3         SELECT DISTINCT Deref(NumAgence) AS agency
4         FROM Comptes;
5     loan_count NUMBER := 0;
6     total_loan_count NUMBER := 0;
7 BEGIN
8     FOR agency_rec IN agency_cursor LOOP
9         SELECT COUNT(*) INTO loan_count
10        FROM (
11            SELECT c.Prets
12            FROM Comptes c
13            WHERE Deref(c.NumAgence) = agency_rec.agency
14        ) compte-prets ,
```

```

15         TABLE(compte_prets.Prets) p;
16
17         DBMS.OUTPUT.PUT_LINE('Number of loans for agency '
18                               || agency_rec.agency.NomAgence || ': ' ||
19                               loan_count);
20         total_loan_count := total_loan_count + loan_count;
21     END LOOP;
22
23     RETURN total_loan_count;
24 END number_of_loans_for_each_agency;

```

Listing 2.3 – Fonction pour calculer le nombre de prêts par agence

Exécution de la fonction

```

1 DECLARE
2     total_loans NUMBER;
3 BEGIN
4     total_loans := Compte_Type.number_of_loans_for_each_agency
5         ();
6     DBMS.OUTPUT.PUT_LINE('Total number of loans for all
7         agencies: ' || total_loans);
8 END;
9 /

```

Listing 2.4 – Utilisation de la fonction

Capture d'exécution

```
Number of loans for agency Agence 101: 4
Number of loans for agency Agence 102: 4
Number of loans for agency Agence 103: 4
Number of loans for agency Agence 104: 4
Number of loans for agency Agence 105: 4
Number of loans for agency Agence 106: 4
Number of loans for agency Agence 107: 4
Number of loans for agency Agence 108: 4
Number of loans for agency Agence 109: 4
Number of loans for agency Agence 110: 4
Number of loans for agency Agence 111: 0
Number of loans for agency Agence 112: 0
Number of loans for agency Agence 113: 0
Number of loans for agency Agence 114: 0
Number of loans for agency Agence 115: 0
Number of loans for agency Agence 116: 0
Number of loans for agency Agence 117: 0
Number of loans for agency Agence 118: 0
Number of loans for agency Agence 119: 0
Number of loans for agency Agence 120: 0
Number of loans for agency Agence 121: 0
Number of loans for agency Agence 122: 0
Number of loans for agency Agence 123: 0
Number of loans for agency Agence 124: 0
Number of loans for agency Agence 125: 0
Total number of loans for all agencies: 40

PL/SQL procedure successfully completed.
```

FIGURE 2.5 – Résultat d'exécution fonction 1

2 : Calculer pour chaque succursale, le nombre d'agences principales qui lui sont rattachées.

La fonction **count_main_agencies** compte le nombre d'agences principales dans une collection d'agences(succursale).

Voici une explication détaillée de la fonction :

1. Une variable **main_agency_count** est déclarée pour garder une trace du nombre d'agences principales.
2. Une variable **agence** de type **Agence_Type** est déclarée pour stocker chaque agence lors de l'itération sur la collection d'agences.
3. La fonction entre ensuite dans une boucle, itérant sur chaque agence dans la collection **self.agences**. **self** fait référence à l'instance de l'objet qui appelle cette méthode.

4. Pour chaque itération, la méthode **get_agence** est appelée avec l'agence actuelle comme argument, et le résultat est stocké dans la variable **agence**.
5. Si la catégorie de l'agence est 'Primary', alors **main_agency_count** est incrémenté de 1.
6. Après que la boucle ait itéré sur toutes les agences, la fonction retourne **main_agency_count**, qui représente le nombre d'agences principales dans la collection.

```

1 MEMBER FUNCTION count_main_agencies RETURN NUMBER IS
2     main_agency_count NUMBER := 0;
3     agence Agence_Type;
4 BEGIN
5     FOR i IN 1..self.agences.COUNT LOOP
6         agence := get_agence(self.agences(i));
7         IF agence.Categorie = 'Primary' THEN
8             main_agency_count := main_agency_count + 1;
9         END IF;
10    END LOOP;
11
12    RETURN main_agency_count;
13 END count_main_agencies;
14 END;
```

Listing 2.5 – Fonction pour calculer le nombre d'agences principales par succursale

Execution de la fonction

```

1 SELECT s.count_main_agencies()
2 FROM Succursale s;
```

Listing 2.6 – Utilisation de la fonction

Capture d'exécution

.

S.COUNT_MAIN_AGENCIES()
4
0
4
0
4
0

FIGURE 2.6 – Résultat d’exécution fonction 2

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

cette fonction calcule le montant total des prêts pour une agence spécifique dans une plage de dates donnée.

Voici une explication étape par étape de la fonction :

1. Une variable `total_amount` est déclarée et initialisée à 0. Cette variable sera utilisée pour stocker le montant total des prêts.
2. Un curseur `p_prets` est déclaré. Ce curseur sélectionne le `MontantPret` (montant du prêt) de la table `Prets` où le `NumAgence` (numéro d’agence) est égal à l’`agency_id` entré et la `DateEffet` (date d’effet) est dans la plage de dates entrée (`start_date` et `end_date`).
3. La fonction entre ensuite dans une boucle où elle récupère chaque ligne renvoyée par le curseur `p_prets`. Pour chaque ligne, elle ajoute le `MontantPret` à `total_amount`.
4. Après que toutes les lignes ont été traitées, la fonction renvoie `total_amount`, qui est le montant total des prêts pour l’agence spécifiée dans la plage de dates spécifiée.

```

1  MEMBER FUNCTION total_amount_of_loans_for_agency (
2      agency_id IN INT,
3      start_date IN DATE,
4      end_date IN DATE
5  ) RETURN NUMBER IS
6      total_amount NUMBER := 0;

```



```

7
8     CURSOR p_prets IS
9         SELECT p.MontantPret
10        FROM Prets p
11        where deref(deref(p.NumCompte).NumAgence).NumAgence =
            agency_id
12        AND p.DateEffet >= start_date AND p.DateEffet <=
            end_date;
13
14
15 BEGIN
16     FOR pret_rec IN p_prets LOOP
17         total_amount := total_amount + pret_rec.MontantPret;
18     END LOOP;
19     RETURN total_amount;
20 END total_amount_of_loans_for_agency;
21 /

```

Listing 2.7 – Fonction pour calculer le montant total des prêts pour une agence donnée

Execution de la fonction

```

1 DECLARE
2     agence Agence_Type;
3     total_amount NUMBER;
4 BEGIN
5     SELECT VALUE(a)
6     INTO agence
7     FROM Agences a
8     WHERE a.NumAgence = 101; — Replace 101 with the actual
        agency ID
9
10    total_amount := agence.total_amount_of_loans_for_agency(101,
        TO_DATE('2022-01-01', 'YYYY-MM-DD'), TO_DATE('2022-12-31',

```

```

11      'YYYY-MM-DD' ) );
12  DBMS_OUTPUT.PUT_LINE( 'Total amount of loans: ' ||
13      total_amount );
14  END;
15  /

```

Listing 2.8 – Utilisation de la fonction

Capture d'exécution

```

SQL> DECLARE
2  agence Agence_Type;
3  total_amount NUMBER;
4  BEGIN
5  SELECT VALUE(a)
6  INTO agence
7  FROM Agences a
8  WHERE a.NumAgence = 101; -- Replace 101 with the actual agency ID
9
10  total_amount := agence.total_amount_of_loans_for_agency(101, TO_DATE('2022-01-01', 'YYYY-MM-DD'), TO_DATE('2022-12-31', 'YYYY-MM-DD'));
11  DBMS_OUTPUT.PUT_LINE('Total amount of loans: ' || total_amount);
12  END;
13  /
Total amount of loans: 98238

```

FIGURE 2.7 – Résultat d'exécution fonction 3

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

Cette fonction `agences_Secondary_ANSEJ`, est conçue pour compter le nombre d'agences secondaires qui ont des prêts de type 'ANSEJ'. Voici une explication détaillée :

1. La fonction déclare plusieurs variables : `total_amount` pour stocker le nombre total d'agences, `agence_name` pour stocker le numéro de l'agence, `toto` pour stocker le nom de l'agence et `toto2` pour stocker le nom de la succursale.
2. Un curseur `c_agences_secondary_ansej` est déclaré pour sélectionner le numéro de l'agence des prêts de type 'ANSEJ' où la catégorie de l'agence est 'Secondaire'.
3. La fonction ouvre le curseur et entre dans une boucle. Pour chaque ligne renvoyée par le curseur, elle fait les opérations suivantes :
 - a) Elle récupère le numéro de l'agence dans `agence_name`.

- b) Elle sélectionne le nom de l'agence correspondant à `agence_name` dans la table `Agences` et le stocke dans `toto`.
 - c) Elle sélectionne le nom de la succursale correspondant à `agence_name` dans la table `Agences` et le stocke dans `toto2`.
 - d) Elle affiche le nom de l'agence et le nom de la succursale.
 - e) Elle incrémente `total_amount` de 1.
4. Une fois que toutes les lignes du curseur ont été traitées, la fonction ferme le curseur et renvoie `total_amount`, qui est le nombre total d'agences secondaires ayant des prêts de type 'ANSEJ'.

```

1
2 CREATE OR REPLACE TYPE BODY Pret_Type AS
3 MEMBER FUNCTION agences_Secondary_ANSEJ RETURN NUMBER IS
4     total_amount NUMBER := 0;
5     agence_name VARCHAR2(100);
6     toto VARCHAR2(100); — Assuming NomAgence is of type
7                           VARCHAR2(100)
8     toto2 VARCHAR2(100); — Assuming NomAgence is of type
9                           VARCHAR2(100)
10    CURSOR c_agences_secondary_ansej IS
11        SELECT Deref(Deref(p.NumCompte).NumAgence).NumAgence
12        FROM Prets p
13        WHERE p.TypePret = 'ANSEJ'
14              AND Deref(Deref(p.NumCompte).NumAgence).Categorie = '
15                  Secondary';
16 BEGIN
17     OPEN c_agences_secondary_ansej;
18     LOOP
19         FETCH c_agences_secondary_ansej INTO agence_name;
20         EXIT WHEN c_agences_secondary_ansej%NOTFOUND;
21         — Process the agence_name value here
22         SELECT a.NumAgence INTO toto FROM Agences a WHERE a.
23             NUMAGENCE = TO_NUMBER(agence_name); — Assuming
24             NUMAGENCE is of type NUMBER

```

```

20      SELECT Deref(a.NumSucc).NomSucc INTO toto2 FROM Agences a
      WHERE a.NUMAGENCE = TO_NUMBER(agence_name); —
      Assuming NUMAGENCE is of type NUMBER
21      DBMS_OUTPUT.PUT_LINE('Agency: ' || toto || ' | Succursale
      ' || toto2);
22      total_amount := total_amount + 1;
23  END LOOP;
24  CLOSE c_agences_secondary_ansej;
25  RETURN total_amount;
26  END agences_Secondary_ANSEJ;
27 END;
28 /

```

Listing 2.9 – Procédure pour lister les agences secondaires avec des prêts ANSEJ

Execution de la méthode

```

1
2
3 DECLARE
4     result NUMBER;
5     pret_instance Pret_Type;
6 BEGIN
7     SELECT VALUE(a)
8     INTO pret_instance
9     FROM Prets a
10    WHERE a.NumPret = 1;
11
12     result := pret_instance.agences_Secondary_ANSEJ();
13 END;
14 /

```

Listing 2.10 – Exécution de la procédure

Capture d'exécution

```
SQL> DECLARE
2  result NUMBER;
3  pret_instance Pret_Type;
4  BEGIN
5  SELECT VALUE(a)
6  INTO pret_instance
7  FROM Prets a
8  WHERE a.NumPret = 1;
9
10 result := pret_instance.agences_Secondary_ANS;
11 END;
12 /
```

Agency: Agence 105 | SuccursaleSuccursale 004
Agency: Agence 101 | SuccursaleSuccursale 006
Agency: Agence 105 | SuccursaleSuccursale 004
Agency: Agence 103 | SuccursaleSuccursale 002
Agency: Agence 101 | SuccursaleSuccursale 006
Agency: Agence 109 | SuccursaleSuccursale 002

FIGURE 2.8 – Résultat d'exécution fonction 4

2.3.3 Définir les tables nécessaires à la base de données

Le code SQL qui va suivre définit plusieurs tables dans la base de données. Chaque table correspond à un type d'entité spécifique et est associée à des contraintes d'intégrité pour garantir la cohérence des données.

1. **Clients** : Cette table contient des informations sur les clients. Chaque client est identifié par un numéro unique (NumClient)
2. **Succursales** : Cette table stocke des détails sur les succursales. Chaque succursale est identifiée par un numéro unique (NumSucc) et appartient à une région spécifique.
3. **Agences** : Cette table contient des informations sur les agences. Chaque agence est identifiée par un numéro unique (NumAgence) et appartient à une succursale spécifique.
4. **Comptes** : Cette table stocke des détails sur les comptes. Chaque compte est identifié par un numéro unique (NumCompte) et appartient à un clients et à une agence spécifique.

5. **Operations** : Cette table contient des enregistrements des opérations effectuées sur les comptes. Chaque opération est identifiée par un numéro unique (NumOperation) et appartient à un comptes spécifique.
6. **Prets** : Cette table stocke des détails sur les prêts accordés. Chaque prêt est identifié par un numéro unique (NumPret) et appartient à un comptes spécifique.

```

1
2
3 CREATE TABLE Agences OF Agence_Type(
4     CONSTRAINT pk_agences PRIMARY KEY(NumAgence) ,
5     CONSTRAINT check_categorie CHECK (Categorie IN ( 'Primary' ,
6         'Secondary' ))
7 );
8
9 CREATE TABLE Succursale OF Succursale_Type(
10     CONSTRAINT pk_succursales PRIMARY KEY(NumSucc) ,
11     CONSTRAINT check_region CHECK (Region IN ( 'North' , 'South' ,
12         'East' , 'West' ))
13 ) nested TABLE agences STORE AS table_agences ;
14
15 CREATE TABLE Clients OF Client_Type(
16     CONSTRAINT pk_clients PRIMARY KEY(NumClient) ,
17     CONSTRAINT check_type_client CHECK (TypeClient IN ( '
18         Individual' , 'Business' ))
19 );
20
21 CREATE TABLE Operations OF Operation_Type(
22     CONSTRAINT pk_operations PRIMARY KEY(NumOperation) ,
23     CONSTRAINT check_nature_op CHECK (NatureOp IN ( 'Credit' , '
24         Debit' ))
25 );
26
27 CREATE TABLE Prets OF Pret_Type(
28     CONSTRAINT pk_prets PRIMARY KEY(NumPret) ,

```

```

25     CONSTRAINT check_type_pret CHECK (TypePret IN ( 'Vehicle', '
      Real Estate', 'ANSEJ', 'ANJEM'))
26 );
27
28 CREATE TABLE Comptes OF Compte_Type(
29     CONSTRAINT pk_comptes PRIMARY KEY(NumCompte),
30     CONSTRAINT check_etat_compte CHECK (EtatCompte IN ( 'Active'
      , 'Blocked'))
31 ) nested TABLE Operations STORE AS table_operations ,
32 nested TABLE Prets STORE AS table_prets;

```

Listing 2.11 – Création des tables

2.4 Création des instances dans les tables

Dans cette section, nous allons présenter le code que nous avons utilisé pour peupler notre base de données.

2.4.1 Succursales

Nous avons défini une simple boucle qui itère 6 fois et insère à chaque fois une nouvelle succursale en incrémentant l’ID et en insérant une liste d’agences vide à chaque fois.

```

1
2 BEGIN
3     FOR i IN 1..6 LOOP
4         INSERT INTO Succursale VALUES (
5             i ,
6             'Succursale ' || TO_CHAR(i, 'FM009'),
7             'Address ' || TO_CHAR(i, 'FM009'),
8             CASE MOD(i, 4)
9                 WHEN 0 THEN 'North'
10                WHEN 1 THEN 'South'
11                WHEN 2 THEN 'East'
12                ELSE 'West'

```

```

13         END,
14         tset_ref_agence()
15     );
16 END LOOP;
17 COMMIT;
18 END;
19 /

```

Listing 2.12 – Insertion des données dans la table Succursales

2.4.2 Agences

Nous avons défini une simple boucle qui itère 25 fois et insère à chaque fois une nouvelle agence en incrémentant l’ID et en mettant une référence vers une succursale aléatoire, puis en mettant à jour la liste des agences de cette succursale.

```

1 DECLARE
2     v_agence_ref REF Agence_Type;
3 BEGIN
4     FOR i IN 101..125 LOOP
5         — Insert a new Agence
6         INSERT INTO Agences
7         VALUES (i, 'Agence ' || TO_CHAR(i, 'FM009'), 'Address '
8                 || TO_CHAR(i, 'FM009'),
9                 CASE MOD(i, 2) WHEN 0 THEN 'Primary' ELSE '
10                    Secondary' END,
11                 (SELECT REF(s) FROM Succursale s WHERE NumSucc
12                    = MOD(i, 6) + 1));
13
14         — Get a reference to the new Agence
15         SELECT REF(a) INTO v_agence_ref FROM Agences a WHERE a.
16            NumAgence = i;
17
18         — Update the Succursale table
19         UPDATE Succursale s
20         SET VALUE(s) = Succursale_Type(s.NumSucc, s.NomSucc, s.

```



```

        AdresseSucc , s.Region , s.agences MULTISSET UNION
        tset_ref_agence(v_agence_ref))
17 WHERE s.NumSucc = MOD(i , 6) + 1;
18 END LOOP;
19
20 COMMIT;
21 END;
22 /

```

Listing 2.13 – Insertion des données dans la table Agences

2.4.3 Clients

Nous avons défini une simple boucle qui itère 100 fois et insère à chaque fois un nouveau client en incrémentant l’ID.

```

1 BEGIN
2   FOR i IN 10001..10100 LOOP
3     — Insert a new Client
4     INSERT INTO Clients
5     VALUES (i , 'Client ' || TO_CHAR(i - 10000 , 'FM0099') ,
6             CASE MOD(i , 2) WHEN 0 THEN 'Business' ELSE '
              Individual' END ,
7             'Address ' || TO_CHAR(i - 10000 , 'FM0099') ,
8             '1234567890' , 'client ' || TO_CHAR(i - 10000 , '
              FM0099') || '@example.com');
9   END LOOP;
10
11  COMMIT;
12 END;
13 /

```

Listing 2.14 – Insertion de données clients

2.4.4 Comptes

Nous avons défini une simple boucle qui itère 100 fois et insère à chaque fois un nouveau compte en incrémentant l’ID et en respectant la plage des IDs (numéro de compte). Nous définissons une référence vers un client et vers une agence, et nous déclarons une table de prêts et une table d’opérations qui sont vides.

```
1 BEGIN
2   FOR i IN 0..99 LOOP
3     — Insert a new Account
4     INSERT INTO Comptes
5     VALUES (TO_NUMBER(TO_CHAR(101 + MOD(i, 25), 'FM009') ||
6       TO_CHAR(1000000 + i, 'FM0999999')),
7       TO_DATE('2022-01-01', 'YYYY-MM-DD'),
8       CASE WHEN i < 40 THEN 'Active' ELSE CASE MOD(i,
9         2) WHEN 0 THEN 'Active' ELSE 'Blocked' END
10      END,
11      TRUNC(DBMS_RANDOM.VALUE(50, 10000)),
12      (SELECT REF(c) FROM Clients c WHERE c.NumClient
13        = 10001 + i),
14      (SELECT REF(a) FROM Agences a WHERE a.NumAgence
15        = 101 + MOD(i, 25)),
16      tset_ref_operation(),
17      tset_ref_pret());
18   END LOOP;
19
20   COMMIT;
21 END;
```

Listing 2.15 – Insertion de données de compte

2.4.5 Operations

Nous avons défini une simple boucle qui itère 200 fois et insère à chaque fois une nouvelle opération en incrémentant l’ID, puis nous faisons une référence de

cette opération vers un compte. Après l'insertion, nous mettons à jour la liste des opérations du compte en question.

```
1
2 DECLARE
3     TYPE op_types IS TABLE OF VARCHAR2(20) INDEX BY PLS_INTEGER
4     v_op_types op_types;
5     v_num_compte Comptes.NumCompte%TYPE;
6 BEGIN
7     v_op_types(1) := 'Credit';
8     v_op_types(2) := 'Debit';
9
10    FOR i IN 1..200 LOOP
11        — Select a random account
12        SELECT NumCompte INTO v_num_compte FROM (SELECT
13            NumCompte FROM Comptes ORDER BY DBMS_RANDOM.VALUE)
14        WHERE ROWNUM = 1;
15
16        — Insert a new Operation
17        INSERT INTO Operations
18        VALUES (i,
19            v_op_types(TRUNC(DBMS_RANDOM.VALUE(1, 2))),
20            TRUNC(DBMS_RANDOM.VALUE(50, 10000)),
21            TO_DATE('2022-01-03', 'YYYY-MM-DD'),
22            'Observation ' || i,
23            (SELECT REF(c) FROM Comptes c WHERE NumCompte =
24                v_num_compte));
25
26        — Update the Comptes table
27        UPDATE Comptes c
28        SET c.Operations = tset_ref_operation((SELECT REF(o)
29            FROM Operations o WHERE NumOperation = i))
30        WHERE NumCompte = v_num_compte;
31    END LOOP;
32
33    COMMIT;
```

```

30 END;
31 /

```

Listing 2.16 – Insertion de données d’opération

2.4.6 Prêts

Nous avons défini une simple boucle qui itère 200 fois et insère à chaque fois un nouveau prêt en incrémentant l’ID, puis nous faisons une référence de ce prêt vers un compte. Après l’insertion, nous mettons à jour la liste des prêts du compte en question.

```

1
2 DECLARE
3     TYPE pret_types IS TABLE OF VARCHAR2(20) INDEX BY
4         PLS_INTEGER;
5     v_pret_types pret_types;
6     v_num_compte Comptes.NumCompte%TYPE;
7 BEGIN
8     v_pret_types(1) := 'Vehicle';
9     v_pret_types(2) := 'Real Estate';
10    v_pret_types(3) := 'ANSEJ';
11    v_pret_types(4) := 'ANJEM';
12
13    FOR i IN 1..200 LOOP
14        — Select a random account from the first 40 accounts
15        SELECT NumCompte INTO v_num_compte FROM (SELECT
16            NumCompte FROM Comptes WHERE ROWNUM <= 40 ORDER BY
17            DBMS_RANDOM.VALUE) WHERE ROWNUM = 1;
18
19        — Insert a new Pret
20        INSERT INTO Prets
21        VALUES (i,
22            TRUNC(DBMS_RANDOM.VALUE(50, 10000)),
23            TO_DATE('2022-01-03', 'YYYY-MM-DD'),
24            TRUNC(DBMS_RANDOM.VALUE(1, 12)),

```

```

22         v_pret_types (TRUNC(DBMSRANDOM.VALUE(1, 4)) ,
23         TRUNC(DBMSRANDOM.VALUE(3, 99)) ,
24         TRUNC(DBMSRANDOM.VALUE(100, 500)) ,
25         (SELECT REF(c) FROM Comptes c WHERE NumCompte =
           v_num_compte)) ;
26
27     — Update the Comptes table
28     UPDATE Comptes c
29     SET c.Prets = tset_ref_pret ((SELECT REF(p) FROM Prets p
           WHERE NumPret = i))
30     WHERE NumCompte = v_num_compte;
31 END LOOP;
32
33 COMMIT;
34 END;
35 /

```

Listing 2.17 – Génération de prêts aléatoires

2.4.7 Capture exécution

Toutes les fonctions et commandes de création des tables ont été exécutées de manière appropriée. Nous avons décidé de ne pas inclure de captures d’écran afin de ne pas alourdir le rapport.

2.5 Langage d’interrogation de données

2.5.1 question 1

cette commande est une requête SQL qui sélectionne les comptes d’affaires (Business) d’une agence spécifique. Voici une explication détaillée :

1. **SELECT c.NumCompte, c.DateOuverture, c.EtatCompte, c.Solde :**

Cette partie de la requête sélectionne les colonnes NumCompte, DateOuverture, EtatCompte et Solde de la table Comptes.

2. **FROM Comptes c** : Cette partie spécifie que les données sont extraites de la table Comptes, et c est un alias pour Comptes.
3. **WHERE Deref(c.NumClient).TypeClient = 'Business'** : Cette condition filtre les enregistrements où le TypeClient est 'Business'. Deref(c.NumClient) déréférence le pointeur NumClient pour accéder à l'objet client associé.
4. **AND Deref(c.NumAgence).NumAgence = :given_agency_id** : cette condition est pour choisir l'agence pour laquelle on affiche les comptes.

```

1 SELECT c.NumCompte, c.DateOuverture, c.EtatCompte, c.Solde
2 FROM Comptes c
3 WHERE Deref(c.NumClient).TypeClient = 'Business'
4 AND Deref(c.NumAgence).NumAgence = :given_agency_id;

```

Listing 2.18 – Query 1

2.5.2 question 2

cette commande est une requête SQL qui sélectionne les prêts des agences d'une succursale donnée. Voici une explication détaillée :

1. **SELECT Deref(p.COLUMN_VALUE).NumPret, Deref(c.NumAgence).NumAgence AS NumAgence, c.NumCompte, Deref(p.COLUMN_VALUE).MontantPret** : Cette partie de la requête sélectionne le numéro de prêt, le numéro d'agence, le numéro de compte et le montant du prêt. Deref(p.COLUMN_VALUE) déréférence la valeur de la colonne pour accéder à l'objet prêt associé.
2. **FROM Comptes c, TABLE(c.Prets) p** : Cette partie spécifie que les données sont extraites de la table Comptes et de la table imbriquée Prets.
3. **WHERE Deref(c.NumAgence).NumSucc.NumSucc = 1** : Cette condition filtre les enregistrements où le numéro de succursale (NumSucc) est 1. Deref(c.NumAgence) déréférence le pointeur NumAgence pour accéder à l'objet agence associé.

```

1
2

```

```

3 SELECT Deref(p.COLUMN_VALUE).NumPret, Deref(c.NumAgence).
   NumAgence AS NumAgence, c.NumCompte, Deref(p.COLUMN_VALUE).
   MontantPret
4 FROM Comptes c, TABLE(c.Prets) p
5 WHERE Deref(c.NumAgence).NumSucc.NumSucc = 1;

```

Listing 2.19 – Query 2

2.5.3 question 3

Cette commande est une requête SQL qui sélectionne les numéros de compte (NumCompte) qui n'ont pas eu d'opérations de débit entre le 1er janvier 2000 et le 31 décembre 2022. Voici une explication détaillée :

1. **SELECT c.NumCompte** : Cette partie de la requête sélectionne la colonne NumCompte de la table Comptes.
2. **FROM Comptes c** : Cette partie spécifie que les données sont extraites de la table Comptes, et c est un alias pour Comptes.
3. **WHERE NOT EXISTS** : Cette condition vérifie l'absence d'un enregistrement correspondant dans le sous-requête.
4. **SELECT 1 FROM TABLE(c.Operations) o WHERE Deref(o.COLUMN_VALUE) = 'Debit' AND Deref(o.COLUMN_VALUE).DateOperation BETWEEN TO_DATE('2000-01-01', 'YYYY-MM-DD') AND TO_DATE('2022-12-31', 'YYYY-MM-DD')** : Cette sous-requête sélectionne les opérations de débit qui ont eu lieu entre le 1^{er} janvier 2000 et le 31 décembre 2022. Deref(o.COLUMN_VALUE) déréférence la valeur de la colonne pour accéder à l'objet opération associé.

```

1 SELECT c.NumCompte
2 FROM Comptes c
3 WHERE NOT EXISTS (
4     SELECT 1
5     FROM TABLE(c.Operations) o
6     WHERE o.COLUMN_VALUE.NatureOp = 'Debit'

```

```

7      AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE( '2000-01-01' , '
      YYYY-MM-DD' ) AND TO_DATE( '2022-12-31' , 'YYYY-MM-DD' )
8  ) ;

```

Listing 2.20 – Query 3

2.5.4 question 4

Cette commande est une requête SQL qui calcule la somme des montants de toutes les opérations de crédit pour le compte numéro 1180005564 qui ont eu lieu entre le 1er janvier 2023 et le 31 décembre 2023. Voici une explication détaillée :

1. **SELECT SUM(o.COLUMN_VALUE.MontantOp)** : Cette partie de la requête calcule la somme des montants des opérations. `o.COLUMN_VALUE.MontantOp` accède à la valeur du montant de l'opération.
2. **FROM Comptes c, TABLE(c.Operations) o** : Cette partie spécifie que les données sont extraites de la table Comptes et de la table imbriquée Operations.
3. **WHERE c.NumCompte = 1180005564 AND o.COLUMN_VALUE.NatureOp = 'Credit' AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND TO_DATE('2023-12-31', 'YYYY-MM-DD')** : Ces conditions filtrent les enregistrements où le numéro de compte est 1180005564, le type d'opération est 'Credit', et la date de l'opération est entre le 1^{er} janvier 2023 et le 31 décembre 2023.

```

1  SELECT SUM(o.COLUMN_VALUE.MontantOp)
2  FROM Comptes c, TABLE(c.Operations) o
3  WHERE c.NumCompte = 1180005564
4      AND o.COLUMN_VALUE.NatureOp = 'Credit'
5      AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE( '2023-01-01' , 'YYYY
      -MM-DD' ) AND TO_DATE( '2023-12-31' , 'YYYY-MM-DD' ) ;

```

Listing 2.21 – Query 4

2.5.5 question 5

Cette commande est une requête SQL qui sélectionne les prêts dont la date d'effet plus la durée est supérieure à la date actuelle. Voici une explication détaillée :

1. **SELECT p.COLUMN_VALUE.NumPret, Deref(c.NumAgence).NumAgence AS NumAgence, c.NumCompte, Deref(c.NumClient).NumClient AS NumClient, p.COLUMN_VALUE.MontantPret** : Cette partie de la requête sélectionne le numéro de prêt, le numéro d'agence, le numéro de compte, le numéro de client et le montant du prêt.
2. **FROM Comptes c, TABLE(c.Prets) p** : Cette partie spécifie que les données sont extraites de la table Comptes et de la table imbriquée Prets.
3. **WHERE ADD_MONTHS(p.COLUMN_VALUE.DateEffet, p.COLUMN_VALUE.Duree) (sup) SYSDATE** : Cette condition filtre les enregistrements où la date d'effet plus la durée du prêt est supérieure à la date actuelle. **ADD_MONTHS** est une fonction qui ajoute un certain nombre de mois à une date.

```
1
2 SELECT p.COLUMN_VALUE.NumPret ,
3        Deref( c . NumAgence ) . NumAgence AS NumAgence ,
4        c . NumCompte ,
5        Deref( c . NumClient ) . NumClient AS NumClient ,
6        p.COLUMN_VALUE. MontantPret
7 FROM Comptes c , TABLE( c . Prets ) p
8 WHERE ADD_MONTHS( p . COLUMN_VALUE. DateEffet , p . COLUMN_VALUE. Duree
   ) > SYSDATE;
```

Listing 2.22 – Query 5

2.5.6 question 6

Cette commande est une requête SQL qui sélectionne le numéro de compte (NumCompte) ayant le plus grand nombre d'opérations effectuées entre le 1er janvier 2023 et le 31 décembre 2023. Voici une explication détaillée :

1. **SELECT c.NumCompte, COUNT(*) AS TotalOperations** : Cette partie de la requête sélectionne la colonne NumCompte et compte le nombre total d'opérations pour chaque NumCompte.
2. **FROM Comptes c, TABLE(c.Operations) o** : Cette partie spécifie que les données sont extraites de la table Comptes et de la table imbriquée Operations.
3. **WHERE o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND TO_DATE('2023-12-31', 'YYYY-MM-DD')** : Cette condition filtre les enregistrements où la date de l'opération est entre le 1^{er} janvier 2023 et le 31 décembre 2023.
4. **GROUP BY c.NumCompte** : Cette partie regroupe les résultats par NumCompte, ce qui permet de compter le nombre total d'opérations pour chaque NumCompte.
5. **ORDER BY TotalOperations DESC** : Cette partie ordonne les résultats en fonction du nombre total d'opérations, en ordre décroissant.
6. **FETCH FIRST 1 ROW ONLY** : Cette partie limite les résultats à la première ligne seulement, ce qui signifie que seul le NumCompte avec le plus grand nombre d'opérations sera retourné.

```
1
2 SELECT c.NumCompte, COUNT(*) AS TotalOperations
3 FROM Comptes c, TABLE(c.Operations) o
4 WHERE o.COLUMN_VALUE.DateOp BETWEEN TO_DATE( '2023-01-01' , 'YYYY
   -MM-DD' ) AND TO_DATE( '2023-12-31' , 'YYYY-MM-DD' )
5 GROUP BY c.NumCompte
6 ORDER BY TotalOperations DESC
7 FETCH FIRST 1 ROW ONLY;
```

Listing 2.23 – Query 6

2.6 Capture d'exécution

2.6.1 Requête 1

```
SQL> SELECT c.NumCompte, c.DateOuverture, c.EtatCompte, c.Solde
2 FROM Comptes c
3 WHERE Deref(c.NumClient).TypeClient = 'Business'
4 AND Deref(c.NumAgence).NumAgence = 'given_agency_id';
```

NUMCOMPTE	DATEOUVERTURE	ETATCOMPTE	SOLDE
1011000025	01-JAN-22	Active	3914
1011000075	01-JAN-22	Blocked	2923

Page 1 of 1 |< >| (1-2 of 2 rows)

FIGURE 2.9 – Résultat d'exécution de la Requête 1

2.6.2 Requête 2

```
SQL> SELECT Deref(p.COLUMN_VALUE).NumPret, Deref(c.NumAgence).NumAgence AS NumAgence, c.NumCompte, Deref(p.COLUMN_VALUE).MontantPret
2 FROM Comptes c, TABLE(c.Prets) p
3 WHERE Deref(c.NumAgence).NumSucc.NumSucc = 1;
```

DEREF(P.COLUMN_VALUE).NUMPRET	NUMAGENCE	NUMCOMPTE	DEREF(P.COLUMN_VALUE).MONTANTPRET
184	102	1021000001	6859
200	108	1081000007	9749
181	102	1021000026	6647
188	108	1081000032	774
161	102	1021000051	9377
134	108	1081000057	4195
172	102	1021000076	1724
149	108	1081000082	5563

Page 1 of 1 |< >| (1-8 of 8 rows)

FIGURE 2.10 – Résultat d'exécution de la Requête 2

2.6.3 Requête 3

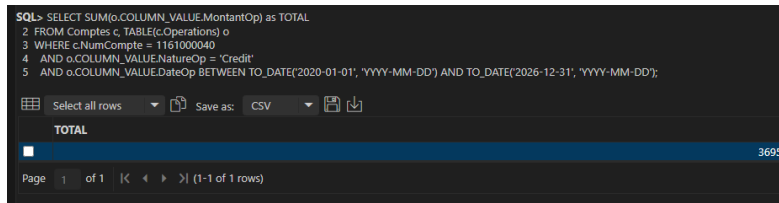
```
SQL> SELECT c.NumCompte
2 FROM Comptes c
3 WHERE NOT EXISTS (
4 SELECT 1
5 FROM TABLE(c.Operations) o
6 WHERE o.COLUMN_VALUE.NatureOp = 'Debit' -- Assuming NatureOp is the column for transaction type
7 AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2000-01-01', 'YYYY-MM-DD') AND TO_DATE('2022-12-31', 'YYYY-MM-DD')
8 );
```

NUMCOMPTE
1071000031
1121000036
1011000050
1151000014
1051000004
1191000034
1161000065
1161000040
1231000022
1061000030

Page 1 of 10 |< >| (1-10 of 100 rows)

FIGURE 2.11 – Résultat d'exécution de la Requête 3

2.6.4 Requête 4



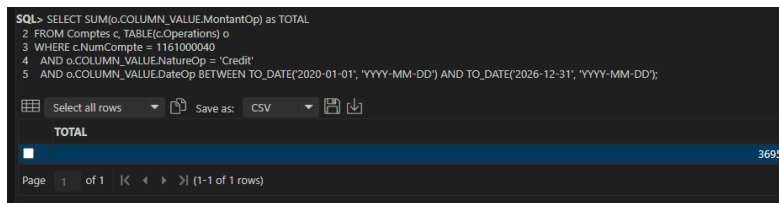
```
SQL> SELECT SUM(o.COLUMN_VALUE.MontantOp) as TOTAL
2 FROM Comptes c, TABLE(c.Operations) o
3 WHERE c.NumCompte = 1161000040
4 AND o.COLUMN_VALUE.NatureOp = 'Credit'
5 AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2020-01-01', 'YYYY-MM-DD') AND TO_DATE('2026-12-31', 'YYYY-MM-DD');
```

TOTAL
3695

Page 1 of 1 |< < > >| (1-1 of 1 rows)

FIGURE 2.12 – Résultat d’exécution de la Requête 4

2.6.5 Requête 5



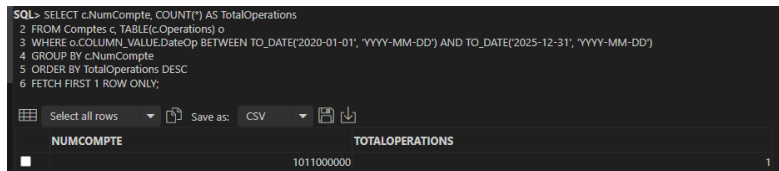
```
SQL> SELECT SUM(o.COLUMN_VALUE.MontantOp) as TOTAL
2 FROM Comptes c, TABLE(c.Operations) o
3 WHERE c.NumCompte = 1161000040
4 AND o.COLUMN_VALUE.NatureOp = 'Credit'
5 AND o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2020-01-01', 'YYYY-MM-DD') AND TO_DATE('2026-12-31', 'YYYY-MM-DD');
```

TOTAL
3695

Page 1 of 1 |< < > >| (1-1 of 1 rows)

FIGURE 2.13 – Résultat d’exécution de la Requête 5

2.6.6 Requête 6



```
SQL> SELECT c.NumCompte, COUNT(*) AS TotalOperations
2 FROM Comptes c, TABLE(c.Operations) o
3 WHERE o.COLUMN_VALUE.DateOp BETWEEN TO_DATE('2020-01-01', 'YYYY-MM-DD') AND TO_DATE('2025-12-31', 'YYYY-MM-DD')
4 GROUP BY c.NumCompte
5 ORDER BY TotalOperations DESC
6 FETCH FIRST 1 ROW ONLY;
```

NUMCOMPTE	TOTALOPERATIONS
1011000000	1

FIGURE 2.14 – Résultat d’exécution de la Requête 6

CHAPITRE 3

Partie II : Non-Sql Mongodb

3.1 Modélisation orientée document

3.1.1 choix de la modélisation

Vu que la plupart des requêtes vont être sur les prêts, nous avons décidé de mettre dans une collection l'enregistrement prêt en premier, suivi du Compte, puis du Client, de l'Agence et de la Succursale à l'intérieur de l'Agence. Pour les opérations, nous avons décidé de les mettre à part dans une autre collection qui respecte la même structure, et cela est dû au fait qu'il n'y a pas de relation directe entre prêts et opérations, ce qui veut dire qu'il y aura probablement pas trop de requêtes qui manipuleront les deux en même temps.

3.1.2 exemple pour illustrer la base de données

Prêt

```
{
  "NumPrêt": 1,
  "montantPrêt": 10000.00,
  "dateEffet": ISODate("2024-04-29"),
  "durée": 24,
  "typePrêt": "Véhicule",
  "tauxIntérêt": 0.05,
  "montantEchéance": 500.00,
  "Compte": {
    "NumCompte": 1,
    "dateOuverture": ISODate("2024-01-15"),
    "étatCompte": "Actif",
    "Solde": 5000.00,
```

```

"Client": {
  "NumClient": 1,
  "NomClient": "Nom du client",
  "TypeClient": "Particulier",
  "AdresseClient": "Adresse du client",
  "NumTel": "Numéro de téléphone",
  "Email": "Adresse email"
},
"Agence": {
  "NumAgence": 102,
  "nomAgence": "Nom de l'agence",
  "adresseAgence": "Adresse de l'agence",
  "catégorie": "Principale",
  "Succursale": {
    "NumSucc": 1,
    "nomSucc": "Nom de la succursale",
    "adresseSucc": "Adresse de la succursale",
    "région": "Nord"
  }
}
}
}
}

```

Operation

```

{
  "NumOpération": 1,
  "NatureOp": "Crédit",
  "montantOp": 500.25,
  "DateOp": ISODate("2024-04-29"),
  "Observation": "Observation sur l'opération",
  "Compte": {
    "NumCompte": 1,

```

```

    "dateOuverture": ISODate("2024-01-15"),
    "étatCompte": "Actif",
    "Solde": 5000.00,
    "Client": {
      "NumClient": 1,
      "NomClient": "Nom du client",
      "TypeClient": "Particulier",
      "AdresseClient": "Adresse du client",
      "NumTel": "Numéro de téléphone",
      "Email": "Adresse email"
    },
    "Agence": {
      "NumAgence": 102,
      "nomAgence": "Nom de l'agence",
      "adresseAgence": "Adresse de l'agence",
      "catégorie": "Principale",
      "Succursale": {
        "NumSucc": 1,
        "nomSucc": "Nom de la succursale",
        "adresseSucc": "Adresse de la succursale",
        "région": "Nord"
      }
    }
  }
}

```

3.1.3 Justificatif de la modélisation

En incorporant les données des collections liées à l'intérieur des documents des collections principales (prêt, opération), nous favorisons une meilleure efficacité des requêtes. Cette approche réduit la nécessité d'effectuer des opérations de jointure coûteuses, car les informations pertinentes sont déjà incluses dans chaque document. Dans notre cas, en intégrant les informations sur les comptes, les agences, les clients et les succursales à l'intérieur des documents des opérations et des prêts,

nous simplifions les requêtes et minimisons le temps de traitement. De plus, cette conception permet une modélisation des données plus naturelle et intuitive, correspondant davantage à la manière dont les données sont manipulées dans le domaine de la banque.

3.1.4 inconvénients de la modélisation

Redondance des données : En incorporant les données des collections liées à l'intérieur des documents des collections principales, il peut y avoir de la redondance des données. Cela peut entraîner une utilisation plus importante de l'espace de stockage, surtout si les données dupliquées sont volumineuses.

Consistance des données : La mise à jour des informations dupliquées peut poser des défis en termes de maintien de la cohérence des données. Si une donnée est modifiée dans un document, elle doit également être mise à jour dans tous les autres documents qui contiennent cette information. Cela nécessite une gestion rigoureuse pour éviter les incohérences.

Performance des requêtes de lecture : Bien que l'incorporation des données liées puisse améliorer les performances des requêtes en évitant les opérations de jointure coûteuses, cela peut également entraîner des documents plus volumineux. Les requêtes de lecture sur des documents volumineux peuvent être plus lentes car avec cette modélisation on se retrouve par fois obligé de lire des information dont on a pas forcément besoin

Migration des données : Si les exigences de modélisation des données changent avec le temps, la migration des données dans une structure différente peut être complexe. Les données dupliquées doivent être mises à jour ou nettoyées pour refléter la nouvelle structure de la base de données, ce qui peut être fastidieux et potentiellement source d'erreurs.

3.2 Remplir la base de données

Pour remplir la base de données, nous avons utilisé le script suivant :

```
use ('ProjetBDA')  
function genererPret() {
```



```

return {
  "NumPrêt": Math.floor(Math.random() * 200) + 1,
  "montantPrêt": parseFloat((Math.random() * (100000 - 1000) + 1000).toFixed(2)),
  "dateEffet": new Date(),
  "durée": Math.floor(Math.random() * (60 - 12) + 12),
  "typePrêt": "Véhicule",
  "tauxIntérêt": parseFloat((Math.random() * 0.1).toFixed(2)),
  "montantEchéance": parseFloat((Math.random() * (5000 - 50) + 50).toFixed(2)),
  "Compte": {
    "NumCompte": Math.floor(Math.random() * 200) + 1,
    "dateOuverture": new Date(),
    "étatCompte": "Actif",
    "Solde": parseFloat((Math.random() * (100000 - 1000) + 1000).toFixed(2)),
    "Client": {
      "NumClient": Math.floor(Math.random() * 200) + 1,
      "NomClient": "Client " + Math.floor(Math.random() * 10) + 1,
      "TypeClient": "Particulier",
      "AdresseClient": "Adresse " + Math.floor(Math.random() * 100) + 1,
      "NumTel": "Numéro " + Math.floor(Math.random() * 1000000) + 1,
      "Email": "Email" + Math.floor(Math.random() * 100) + 1 + "@example",
    },
    "Agence": {
      "NumAgence": Math.floor(Math.random() * 100) + 101,
      "nomAgence": "Agence " + Math.floor(Math.random() * 10) + 1,
      "adresseAgence": "Adresse " + Math.floor(Math.random() * 100) + 1,
      "catégorie": "Principale",
      "Succursale": {
        "NumSucc": Math.floor(Math.random() * 10) + 1,
        "nomSucc": "Succursale " + Math.floor(Math.random() * 5) + 1,
        "adresseSucc": "Adresse " + Math.floor(Math.random() * 20) + 1,
        "région": "Nord"
      }
    }
  }
}

```

```

    }
};
}

```

// Fonction pour générer des données aléatoires pour les opérations

```

function genererOperation() {
    return {
        "NumOpération": Math.floor(Math.random() * 200) + 1,
        "NatureOp": "Crédit",
        "montantOp": parseFloat((Math.random() * (1000 - 10) + 10).toFixed(2)),
        "DateOp": new Date(),
        "Observation": "Observation sur l'opération",
        "Compte": {
            "NumCompte": Math.floor(Math.random() * 200) + 1,
            "dateOuverture": new Date(),
            "étatCompte": "Actif",
            "Solde": parseFloat((Math.random() * (100000 - 1000) + 1000).toFixed(2)),
            "Client": {
                "NumClient": Math.floor(Math.random() * 200) + 1,
                "NomClient": "Client " + Math.floor(Math.random() * 10) + 1,
                "TypeClient": "Particulier",
                "AdresseClient": "Adresse " + Math.floor(Math.random() * 100) + 1,
                "NumTel": "Numéro " + Math.floor(Math.random() * 1000000) + 1,
                "Email": "Email" + Math.floor(Math.random() * 100) + 1 + "@example"
            },
            "Agence": {
                "NumAgence": Math.floor(Math.random() * 100) + 101,
                "nomAgence": "Agence " + Math.floor(Math.random() * 10) + 1,
                "adresseAgence": "Adresse " + Math.floor(Math.random() * 100) + 1,
                "catégorie": "Principale",
                "Succursale": {

```

```

        "NumSucc": Math.floor(Math.random() * 10) + 1,
        "nomSucc": "Succursale " + Math.floor(Math.random() * 5) + 1,
        "adresseSucc": "Adresse " + Math.floor(Math.random() * 20) + 1,
        "région": "Nord"
    }
}
};
}

// Insertion des prêts dans la collection "Prêts"
for (let i = 0; i < 200; i++) {
    let pret = genererPret();
    db.Prets.insertOne(pret);
}

// Insertion des opérations dans la collection "Opérations"
for (let i = 0; i < 200; i++) {
    let operation = genererOperation();
    db.Opérations.insertOne(operation);
}

```

3.3 Les requêtes

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

```

use ('ProjetBDA')
db.Prets.find({"Compte.Agence.NumAgence": 102})

```

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

```
use ('ProjetBDA')
db.Prets.aggregate([
  {
    $match: {
      "Compte.Agence.Succursale.région": "Nord"
    }
  },
  {
    $project: {
      "NumPrêt": 1,
      "Compte.Agence.NumAgence": 1,
      "Compte.NumCompte": 1,
      "Compte.Client.NumClient": 1,
      "montantPrêt": 1,
      "Compte.Agence.Succursale.région": 1
    }
  }
])
```

3.3.3 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.

```
use ('ProjetBDA')
db.Prets.aggregate([
  {
    $group: {
```

```

        _id: "$Compte.Agence.NumAgence",
        totalPrets: { $sum: 1 }
    }
},
{
    $sort: {
        totalPrets: -1
    }
},
{
    $out: "Agence-NbPrets"
}
])

// To display the content of the new collection
db['Agence-NbPrets'].find()

```

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

```

use ('ProjetBDA')
db.Prets.aggregate([
    {
        $match: {
            "typePrêt": "ANSEJ"
        }
    },
    {
        $project: {
            "NumPrêt": 1,
            "Compte.Client.NumClient": 1,
            "montantPrêt": 1,

```

```

        "dateEffet": 1
    }
},
{
    $out: "Prêt-ANSEJ"
}
])

```

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

```

use ('ProjetBDA')

db.Prets.find(
    { "Compte.Client.TypeClient": "Particulier" },
    { "Compte.Client.NumClient": 1, "Compte.NomClient": 1, "NumPrêt": 1, "montantPrêt": 1 }
).toArray();

```

3.3.6 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.

```

use ('ProjetBDA')
db.Prets.updateMany(
    {
        "dateEffet": { $lt: new Date("2025-01-01") },
        "étatPrêt": { $ne: "Soldé" }
    },
    {
        $inc: { "montantEchéance": 2000 }
    }
)

```

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

```
use ('ProjetBDA')
db.Prets.mapReduce(
  function() {
    emit(this.Compte.Agence.NumAgence, 1);
  },
  function(key, values) {
    return Array.sum(values);
  },
  {
    out: { replace: "Agence-NbPrêts2" },
    sort: { value: -1 }
  }
)
```

3.3.8 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 on peut répondre a la requête car on a bien définie une collection pour les opérations

```
use ('ProjetBDA')
db.Opérations.find({
  "NatureOp": "Crédit",
  "Compte.Client.TypeClient": "Entreprise",
  "DateOp": {
    $gte: ISODate("2023-01-01"),
    $lt: ISODate("2026-01-01")
  }
})
```

3.4 Analyse

La modélisation proposée, où les données des collections liées sont incorporées à l'intérieur des documents des collections principales, s'avère être bien adaptée aux types de requêtes spécifiés. Les requêtes comme "Afficher tous les prêts effectués auprès de l'agence de numéro 102" et "Afficher tous les prêts effectués auprès des agences rattachées aux succursales de la région 'Nord'" sont traitées efficacement grâce à cette conception. En évitant les opérations de jointure coûteuses, la modélisation permet des performances optimales lors de l'exécution de ces requêtes. De plus, en incluant toutes les informations pertinentes à l'intérieur des documents, la structure de données simplifie la logique des requêtes et garantit une récupération rapide et précise des résultats, offrant ainsi une solution robuste et efficace pour les besoins spécifiques de l'application bancaire.