

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'informatique

Département informatique

Projet BDA  
SQL3-Oracle et NoSQL (MongoDB)

**Présenté par:**

- Hadjimi Lilia 202031075620
- Ferchichi Manel 202031036637

# Introduction

Dans ce projet, nous explorons deux approches distinctes pour la modélisation et la gestion de bases de données dans le contexte d'une application de gestion bancaire. Dans la première partie, nous adoptons **une méthodologie orientée objet** en utilisant des diagrammes de classe UML pour conceptualiser la structure des données. **Nous mettons ensuite en œuvre cette conception à l'aide de SQL3**, une extension SQL dédiée aux bases de données orientées objet. SQL3 permet la représentation efficace et la manipulation d'entités complexes telles que les succursales, les agences, les clients, les comptes, les opérations et les prêts. **Dans la deuxième partie, nous optons pour une approche NoSQL en utilisant MongoDB**, une base de données orientée document. MongoDB stocke les données sous forme de documents JSON souples au sein de collections, ce qui offre une flexibilité accrue dans la modélisation des données et permet une évolutivité horizontale aisée. MongoDB est particulièrement adapté aux applications nécessitant une manipulation agile et rapide des données non structurées ou semi-structurées.

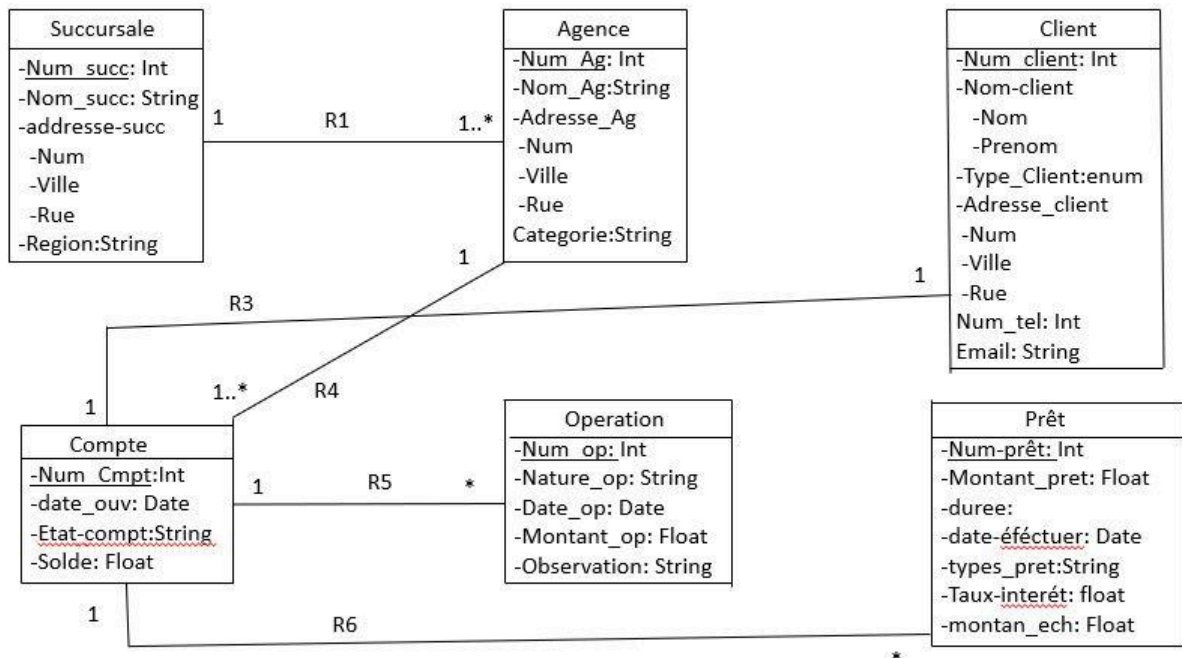
# Partie I : SQL3 :

## A- Modélisation orientée objet:

### 1. Diagramme de classe:

#### Définition:

Un diagramme de classe est une représentation statique qui décrit la structure d'un système en identifiant ses classes, leurs attributs et leurs relations. Il sert de modèle pour concevoir et comprendre la structure d'un système logiciel



#### Description:

Nom de la classe	Attributs
Sucursale	Num_Succ, Nom_suc, Adresse de type TAdresse,Region
Agence	Num_ag, Nom_ag , adresse de type TAdresse, categorie
Client	Num_client,Nom de type Tnom, Adresse de type TAdresse, Num_tel, Email

Compte	Num_Cmpt,date_ouv ,État-compt -Solde: Float
Operation	-Num_op,Nature_op,Date_op,Montant_op Observation
Pret	Num-prêt,Montant_pret,durée,date-éfectuer, types_pret,Taux-interêt,montan_ech

*Les relations entre classes:*

Nom de la relation	Description
R1	entre agence et succursale
R3	entre client et compte
R4	entre compte et agence
R5	entre compte et operation
R6	entre prêt et compte

## B- Création des TableSpaces et utilisateur:

Création de la tablespace space TBS\_SL3 :

```
SQL> CREATE TABLESPACE SQL3_TBS  
2 DATAFILE 'C:\TBS_SQL3.dat' SIZE 100M  
3 AUTOEXTEND ON ONLINE;  
  
Tablespace cr   .
```

Cr  ation de la tablespace temporaire SQL3\_TempTBS :

```
SQL> CREATE TEMPORARY TABLESPACE SQL3_TempTBS  
2 TEMPFILE 'C:\TempTBS_SQL3.dat' SIZE 100M  
3 AUTOEXTEND ON;  
  
Tablespace cr   .
```

Cr  ation de l'utilisateur SQL3 et lui attribuer les deux tables spaces :

```
SQL> CREATE USER C##user_SQL3 IDENTIFIED BY project DEFAULT TABLESPACE SQL3_TBS TEMPORARY TABLESPACE SQL3_TempTBS;  
  
Utilisateur cr   .
```

Attribution des privil  ges    l'utilisateur C ##user\_SQL3 :

```
SQL> GRANT ALL PRIVILEGES TO C##user_SQL3;  
  
Autorisation de privil  ges (GRANT) accept  e.
```

## C- Langage de définition de données:

### Définition des types abstraits:

Nom du type abstrait	Description du type
Tadresse	a comme attributs <i>num</i> de type entier qui représente le numéro de la rue, <i>Ville</i> de type String représente un nom de ville, <i>Rue</i> de type String représente le nom de rue
Tnom	a comme attributs <i>Nom</i> de type String qui représente le nom de famille, <i>Prenom</i> qui représente le nom propre
Tsuccursale	a comme attributs Num_succ de type entier représente le numéro de succursale, Nom_succ de type String, adresse de type Adresse, Région de type String
Tagence	a comme attributs Num_Ag de type entier represente le numéro de l'agence, Nom_Ag de type String, adresse de type Adresse, Catégorie de type String
Tclient	a comme attributs Num_client de type entier, Nom_client de type Tnom, adresse_client de type Adresse, Num_tel de type entier, Email de type String
Tcompte	a comme attributs Num_cmpt de type entier, date_ouv de type Date, Etat_compt de type String, Solde de type float
Toperation	a comme attribut Num_op de type entier, Nature_op de type String, Date_op de type Date, Montant_op de type Float, Observation de type String
Tpret	a comme attribut Num_pret de type entier, Montant_pret de type float, durée de type String, date-effectuer de type Date, type_pret de type String, Taux_interet de type float, montant-ech de type float

### Code de la création des types:

### type Tnom:

```
SQL> create type Tnom as object(  
2     nomf varchar(15),  
3     prenom varchar(20)  
4 );  
5 /
```

### type Tadresse

```
SQL> create type Tadresse as object(  
2  
3     Num number,  
4     ville varchar(20),  
5     Rue varchar(20)  
6  
7 );  
8 /  
  
Type créé.
```

### Type TPret:

```
SQL> create type Tpret as object (  
2     Num_pret number,  
3     Duree varchar(25),  
4     Date_effectuer DATE,  
5     Montant_pret double precision,  
6     Type_pret varchar(20),  
7     Taux_interet double precision,  
8     Num_cmpt ref Tcompte ,  
9     Montant_ech double precision  
10  
11 );  
12 /  
  
Type créé.
```

### Type Toperation:

```
SQL> create type Toperation as object (  
2     Num_op number,  
3     Nature_op varchar(10),  
4     Date_op DATE,  
5     Montant_op double precision,  
6     Num_cmpt ref Tcompte,  
7     Observation varchar(20)  
8 );  
9 /  
  
Type cr   .
```

### La table de r  f  rences des op  rations:

```
SQL> create type T_set_ref_operations as table of ref Toperation;  
2 /  
  
Type cr   .
```

### Modification du type Tcompte:

```
SQL> ALTER TYPE Tcompte ADD ATTRIBUTE operation_compte T_set_ref_operations cascade;  
  
Type modifi  .
```

### Type Tagence:

```
SQL> create type Tagence as object(  
2     Num_Ag number,  
3     Nom_Ag varchar(20),  
4     adresse_Ag Tadresse,  
5     Num_succ ref Tsuccursale,  
6     Categorie varchar(20)  
7 );  
8 /  
  
Type cr   .
```



### Type Tsuccursale:

```
SQL> create type Tsuccursale as object(  
2     Num_succ number,  
3     Nom_succ varchar(20),  
4     adresse_succ Tadresse,  
5     Region varchar(20)  
6  
7 );  
8 /  
  
Type créé.
```

### La table de références des agences :

```
SQL> create type t_set_ref_agences as table of ref Tagence;  
2 /  
  
Type créé.  
  
SQL> alter type Tsuccursale add attribute agence_succursale t_set_ref_agences cascade;  
  
Type modifié.
```

### Type Tcompte:

```
SQL> create or replace type Tcompte as object (  
2     Num_cmpt number,  
3     date_ouv DATE,  
4     Etat_compt varchar(10),  
5     Num_client ref Tclient,  
6     Num_Ag ref Tagence,  
7     Solde double precision  
8 );  
9 /  
  
Type créé.
```

### La table de références des comptes:

```
SQL> create type t_set_ref_compte as table of ref Tcompte;  
2 /  
  
Type créé.  
  
SQL> Alter type Tagence add attribute agence_compte t_set_ref_compte cascade;  
  
Type modifié.
```

### Type Tclient:

```

SQL> create type Tclient as object (
2     Num_client number,
3     Nom_client varchar(20),
4     Type_client varchar(13),
5     adresse_client Tadresse,
6     Num_tel number,
7     Email varchar(20),
8     Num_Ag ref Tagence,
9     Num_cmpt ref Tcompte
10
11 );
12 /

```

Type créé.

## Création des méthodes:

1. Calculer pour chaque agence, le nombre de prêts effectués:

```

alter type Tagence add member function Nombre_Prets_Par_Agence return number cascade;
-- la methode 1
CREATE OR REPLACE TYPE BODY Tagence AS
    MEMBER FUNCTION Nombre_Prets_Par_Agence RETURN NUMBER
    IS
        total_prets NUMBER := 0;
    BEGIN
        -- Boucle sur chaque agence
        FOR agence_rec IN (SELECT Num_Ag FROM Agence) LOOP
            -- Recherche du nombre de prêts pour l'agence actuelle
            SELECT COUNT(*)
            INTO total_prets
            FROM Pret p
            JOIN Compte c ON p.Num_cmpt = REF(c)
            WHERE c.Num_Ag = agence_rec.Num_Ag;

            -- Ajout du nombre de prêts pour l'agence actuelle au total
            total_prets := total_prets + nombre_prets;
        END LOOP;

        -- Retourner le total des prêts pour toutes les agences
        RETURN total_prets;
    END;
END;

```

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

```
--methode 2:
alter type Tsuccursale add member function Nombre_Prets_Par_Agence return number cascade;

CREATE OR REPLACE TYPE BODY Tsuccursale AS
  MEMBER FUNCTION Nombre_Agences_Principales RETURN NUMBER
  IS
    total_agences INTEGER := 0;
  BEGIN
    -- Compte le nombre d'agences principales pour la succursale actuelle
    SELECT COUNT(*)
    INTO total_agences
    FROM Agence
    WHERE NumSucc = REF(self).Num_succ
    AND Categorie = 'Principale';

    -- Retourne le nombre total d'agences principales pour la succursale
    RETURN total_agences;
  END;
END;
/
```

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:

```
alter type Tagence add member function Montant_Global_Prets_Par_Agence return number cascade;
CREATE OR REPLACE TYPE BODY Tagence AS
  MEMBER FUNCTION Montant_Global_Prets_Par_Agence RETURN NUMBER
  IS
    montant_global NUMBER := 0;
  BEGIN
    -- Calcul du montant global des prêts pour une agence donnée
    SELECT SUM(p.Montant_pret)
    INTO montant_global
    FROM Pret p
    JOIN Compte c ON p.Num_cmpt = REF(c)
    JOIN Agence a ON c.Num_Ag = REF(a)
    WHERE a.Num_Ag = self.Num_Ag
    AND p.Date_effectuer >= TO_DATE('2020-01-01', 'YYYY-MM-DD')
    AND p.Date_effectuer < TO_DATE('2024-01-01', 'YYYY-MM-DD');

    -- Retour du montant global des prêts
    RETURN montant_global;
  END;
END;
/
```

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

```

alter type Tagence add member function Agences_Secondaires_Avec_Pret_ANSEJ return number cascade;

CREATE OR REPLACE TYPE BODY Tagence AS
  MEMBER FUNCTION Agences_Secondaires_Avec_Pret_ANSEJ RETURN NUMBER
  IS
    total_agences INTEGER := 0;
  BEGIN
    -- Compter le nombre d'agences secondaires avec au moins un prêt ANSEJ
    SELECT COUNT(*)
    INTO total_agences
    FROM Agence a, Compte c, Pret p, Succursale s
    WHERE a.Num_Ag = c.Num_Ag
    AND c.Num_cmpt = p.Num_cmpt
    AND a.Num_succ = s.Num_succ
    AND a.Categorie = 'Secondaire'
    AND p.Type_pret = 'ANSEJ';

    -- Retourner le nombre d'agences secondaires avec au moins un prêt ANSEJ
    RETURN total_agences;
  END;
END;
/

```

## Création des tables:

### La table succursale avec la tables de références sur les succursales:

```

SQL> create table Succursale of Tsuccursale (primary key (Num_succ),
2  CONSTRAINT check_region CHECK (Region IN ('Nord', 'Sud', 'Est', 'Ouest')))
3  nested table agence_succursale store as table_ref_agances;

Table créée.

```

### La table agences avec la tables de références:

```

SQL> create table Agence of Tagence( primary key (Num_Ag),
2  FOREIGN key(Num_succ) references Succursale,
3  CONSTRAINT check_categorie CHECK (Categorie IN ('Principale', 'Secondaire')))
4  nested table agence_compte store as table_ref_comptes;

Table créée.

```

### Création de la table Client:

```
SQL> create table Client of Tclient(primary key (Num_client),CONSTRAINT check_type_client CHECK (Type_client IN ('Particulier', 'Entreprise')));
Table créée.
```

## Création de la table Compte:

```
SQL> CREATE TABLE Compte OF Tcompte (
2     PRIMARY KEY(Num_cmpt),
3     FOREIGN KEY(Num_client) REFERENCES Client,
4     FOREIGN KEY(Num_Ag) REFERENCES Agence,
5     CONSTRAINT check_EtatCMPT CHECK (Etat_cmpt IN ('Actif', 'Bloquer')),
6     CONSTRAINT check_Solde CHECK (Solde >= 0)
7 )
8 nested table operation_compte store as table_ref_operations,
9 nested table pret_compte store as table_ref_prets;
Table créée.

SQL> CREATE OR REPLACE TRIGGER check_date_ouverture
2 BEFORE INSERT OR UPDATE ON Compte
3 FOR EACH ROW
4 BEGIN
5     IF :NEW.date_ouv < SYSDATE THEN
6         RAISE_APPLICATION_ERROR(-20001, 'La date d'ouverture doit être postérieure ou égale à la date actuelle.');
```

7 END IF;

8 END;

9 /

Déclencheur créé.

## Création de la table operation:

```
SQL> create table Operation of Toperation(primary key (Num_op),
2 FOREIGN key(Num_cmpt) references Compte, CONSTRAINT check_NatureOp CHECK (Nature_op IN ('Credit', 'Debit')));
Table créée.

SQL>
SQL>
SQL>
SQL> CREATE OR REPLACE TRIGGER check_date_operation
2 BEFORE INSERT ON Operation
3 FOR EACH ROW
4 BEGIN
5     IF :NEW.Date_op < SYSDATE THEN
6         RAISE_APPLICATION_ERROR(-20001, 'La date de l'opération doit être postérieure ou égale à la date actuelle.');
```

7 END IF;

8 END;

9 /

Déclencheur créé.

## Création de la table prêt:



```

SQL> create table Pret of Tpret(primary key(Num_pret),
2   FOREIGN key(Num_cmpt) references Compte,
3   CONSTRAINT check_type_pret CHECK (Type_pret IN ('Vehicule', 'Immobilier','ANSEJ','ANJEM'))
4   );

Table cr   e.

SQL>
SQL> CREATE OR REPLACE TRIGGER check_date_effect
2   BEFORE INSERT ON Pret
3   FOR EACH ROW
4   BEGIN
5       IF :NEW.Date_effetuer < SYSDATE THEN
6           RAISE_APPLICATION_ERROR(-20001, 'La date de l'effet du pr  t doit   tre post  rieure ou   gale    la date ac
tuelle.');
```

```

7       END IF;
8   END;
9   /

D  clencheur cr    .
```

### Les insertions dans les tables:

**Dans la table succursale on a ins  r   6 agences donnant un exemple d'une seule insertion les cinqs autre se font de la m  me mani  re:**

```

SQL> INSERT INTO Succursale values(
2   Tsuccursale(001,'SuccursaleNord',Tadresse(001,'Alger','1600 didouch'),'Nord',t_set_ref_agences())
3   );

1 ligne cr   e.

SQL> commit ;
```

**Dans la table agence on a inserer 25 agences donnant un exemple d'une seule insertion les 24 autre se font de la m  me mani  re:**

```

SQL> INSERT INTO Agence VALUES (Tagence(001,
2   'BNA',Tadresse(001,'Alger','Didouch Mourad'),
3   (select ref(s) from succursale s where s.Num_succ=001),'Principale',t_set_ref_compte()));

1 ligne cr   e.

SQL> commit;
```

**Dans la table Client et dans la table Compte simultan  ment:**

```

INSERT into client values(tclient(15202,'Ziad',
'Particulier',tadresse('05','alger','Hussin dey'),0559561388,'ziado@gmail.com',NULL));

INSERT into compte values(tcompte(1010000002,to_date('18/05/2024'),'Actif',
(select ref(c) from client c where c.Num_client=15202),
(select ref(a) from agence a where a.Num_Ag=004),1530,T_set_ref_operations(), t_set_ref_pret()));

update client c set c. Num_cmpt=(select ref(com) from compte com where com.Num_cmpt=1010000002)
where c.Num_client=15202;

```

### A l'exécution:

```

SQL> INSERT into client values(tclient(15202,'Ziad',
  2 'Particulier',tadresse('05','alger','Hussin dey'),0559561388,'ziado@gmail.com',NULL));
1 ligne cr    e.

SQL>
SQL> INSERT into compte values(tcompte(1010000002,to_date('18/05/2024'),'Actif',
  2 (select ref(c) from client c where c.Num_client=15202),
  3 (select ref(a) from agence a where a.Num_Ag=004),1530,T_set_ref_operations(), t_set_ref_pret()));
1 ligne cr    e.

SQL>
SQL> update client c set c. Num_cmpt=(select ref(com) from compte com where com.Num_cmpt=1010000002)
  2 where c.Num_client=15202;
1 ligne mise    jour.

```

### Dans la table operation:

```

INSERT INTO operation VALUES (toperation(1,'Credit', TO_DATE('20/05/2024'),200, (SELECT REF(com) FROM
compte com WHERE com.Num_cmpt = 1010000001),'valide'));
insert into table (select c.operation_compte
from compte c
where c.Num_cmpt=1010000001)
values ((select ref(op) from operation op where op.Num_op=1));

```

### A l'ex  cution:

```

SQL> insert into table (select c.operation_compte
  2 from compte c
  3 where c.Num_cmpt=1010000001)
  4 values ((select ref(op) from operation op where op.Num_op=1));
1 ligne cr    e.

SQL>
SQL> INSERT INTO operation VALUES (toperation(2,'Credit', TO_DATE('21/05/2024'),400, (SELECT REF(com) FROM compte com W
HERE com.Num_cmpt = 1010000001),'valide'));
1 ligne cr    e.

```

### Dans la table pr  t:

```

INSERT into pret values(tpret(1,'1min',
to_date('19/05/2024'),100,'Vehicule',10,(select ref(com) from compte com where com.Num_cmpt=1010000001),150));

INSERT INTO pret VALUES (tpret(2,'2min', TO_DATE('20/05/2024'), 200,'Immobilier',
12, (SELECT REF(com) FROM compte com WHERE com.Num_cmpt= 1010000001),250));

insert into table (select c.pret_compte
                    from compte c
                    where c.Num_cmpt= 1010000001)

values ((select ref(p) from pret p where p.Num_pret=1));
insert into table (select c.pret_compte
                    from compte c
                    where c.Num_cmpt=1010000001)

values ((select ref(p) from pret p where p.Num_pret=2));

```

## A l'exécution:

```

SQL> INSERT into pret values(tpret(1,'1min',
  2  to_date('19/05/2024'),100,'Vehicule',10,(select ref(com) from compte com where com.Num_cmpt=1010000001),150));
1 ligne créée.

SQL>
SQL> INSERT INTO pret VALUES (tpret(2,'2min', TO_DATE('20/05/2024'), 200,'Immobilier',
  2  12, (SELECT REF(com) FROM compte com WHERE com.Num_cmpt= 1010000001),250));
1 ligne créée.

```



## D- Langage d'interrogation de données:

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

```
SQL> SELECT c.Num_cmpt
  2 FROM Compte c, Agence a, Client cl
  3 WHERE c.Num_Ag = REF(a)
  4 AND a.Num_Ag = 008
  5 AND c.Num_client = REF(cl)
  6 AND cl.Type_client = 'Entreprise';

NUM_CMPT
-----
1010000005
1010000006
```

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

```
SQL> SELECT p.Num_pret, p.Num_cmpt.Num_Ag.Num_Ag, p.Num_cmpt.Num_cmpt, p.Montant_pret
  2 FROM Pret p, Compte c, Agence a, Succursale s
  3 WHERE p.Num_cmpt = REF(c)
  4 AND c.Num_Ag = REF(a)
  5 AND a.Num_succ = REF(s)
  6 AND s.Num_succ = 005;

NUM_PRET NUM_CMPT.NUM_AG.NUM_AG NUM_CMPT.NUM_CMPT MONTANT_PRET
-----
6          11          1010000004          199
10         11          1010000009          199
14         11          1010000012          167
16         11          1010000012          1698
26         15          1010000022          7698
33         21          1010000029          4306

6 lignes sélectionnées.
```

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

```
SQL> SELECT c.Num_cmpt
 2  FROM Compte c
 3  WHERE NOT EXISTS (
 4      SELECT 1
 5      FROM Operation o
 6      WHERE o.Num_cmpt = REF(c)
 7      AND o.Nature_op = 'Debit'
 8      AND o.Date_op BETWEEN TO_DATE('2000-01-01', 'YYYY-MM-DD') AND TO_DATE('2022-12-31', 'YYYY-MM-DD')
 9  );

NUM_CMPT
-----
1010000007
1010000003
1010000004
1010000006
1010000024
1010000002
1010000026
1010000016
1010000025
1010000010
1010000009
```

4. Quel est le montant total des crédits effectués sur un compte (numCompte = 1180005564) en 2024:

```
SQL> SELECT SUM(o.Montant_op)
 2  FROM Operation o, Compte c
 3  WHERE o.Num_cmpt = REF(c)
 4  AND c.Num_cmpt =1010000022
 5  AND o.Nature_op = 'Credit'
 6  AND EXTRACT(YEAR FROM o.Date_op) = 2024;

SUM(O.MONTANT_OP)
-----
129
```

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

```
SQL> SELECT p.Num_pret, p.Num_cmpt.Num_Ag.Num_Ag, p.Num_cmpt.Num_cmpt, p.Num_cmpt.Num_client.Num_client, p.Montant_pret
 2 FROM Pret p
 3 WHERE p.Montant_pret > 0;

NUM_PRET NUM_CMPT.NUM_AG.NUM_AG NUM_CMPT.NUM_CMPT
-----
NUM_CMPT.NUM_CLIENT.NUM_CLIENT MONTANT_PRET
-----
      1              15201      4      1010000001
                                100
      2              15201      4      1010000001
                                200
      4              15202      4      1010000002
                                200

NUM_PRET NUM_CMPT.NUM_AG.NUM_AG NUM_CMPT.NUM_CMPT
```

24 lignes sélectionnées.

# Partie II : NoSQL – Modèle orienté « documents »

## A- 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.

Concernant la modélisation Orientée document de la BD décrite dans la partie 1 : On propose une modélisation qui comporte la conception Imbriqué pour les documents ( Compte ) et Hybrides pour le reste de la base de données .

Documents Compte va comporter les collections de Prêt et opérations ainsi que les informations de Client qui a ce compte , donc pour les tables Compte ,Prêt ,Opérations,et Client ca sera Imbriqué .

Pour les tables Succursale Agence on a opté pour la conception par référence .

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

Pour l' illustration , voici un exemple de documents Comptes , et voici un exemple de documents Succursale:

### Remarques :

- Pour la collection Compte , ça comporte ,le client qui est propriétaire du compte , qui est de type Object (donc on site tous ses informations comme l'exemple le montre ci dessous )
- Toujours pour la collection Compte, ça comporte deux ensembles d'objets (des Prêts et des opérations ) donc il y aura un liste des informations de chaque Prêt ou opération.
- Pour la collection Agence , elle va contenir le numéro de Succursale , alors que pour la collection Succursale elle contient un tableau de numéro d' agences .

### Exemple D'un Compte :

```
{
  "NumCompte": 1001,
  "dateOuverture": ISODate("2024-05-03T00:00:00Z"),
  "etatCompte": "Actif",
  "Solde": 5000.00,
  "NumAgence": 1,
  "Client": {
    "NumClient": 123456789,
    "NomClient": "John Doe",
    "TypeClient": "Particulier",
    "adresseClient": "456 Rue Secondaire, Ville",
    "numtel": 1234567890,
    "email": "john.doe@example.com",
  },
  "Operations": [
    {
      "NumOperation": 1,
      "natureOp": "Credit",
      "montantOp": 1000.00,
      "dateOp": ISODate("2024-05-01T08:30:00Z"),
      "observation": "Dépôt en espèces",
      "Compte": 1001
    }
  ],
  "Prets": [
    {
      "NumPret": 101,
      "montantOp": 20000.00,
      "dateEffet": ISODate("2023-10-15T00:00:00Z"),
      "duree": "36 mois",
      "typePret": "Vehicule",
      "tauxInteret": 0.05,
      "montantEcheance": 600.00,
      "Compte": 1001
    }
  ]
}
```

### Exemple D'un Succursale :

```
{  
  "NumSucc": 1,  
  "NomSucc": "Succursale Nord",  
  "adresseSucc": "789 Avenue Principale, Ville",  
  "region": "Nord",  
  "Agences": [1, 2]  
}
```

### - Justifiez votre choix de conception

#### Justifications :

- Comme on sait que la plupart des requêtes (traitement) seront faites sur les Prêt donc l'utilisation de l'imbrication va réduire le taux des jointure lors des traitement de ces données . Ainsi on simplifie la gestion et la récupération des associés au comptes .
- maintenir une base de données cohérente et atomique
- en référençant toutes les agences dans la Succursale parente ca aide à récupérer toutes les agences qui appartiennent à une même Succursale .

### - Quelles sont les inconvénients de votre conception

#### Inconvénients :

- Tomber dans le cas de redondances de données est possible.
- Avec une structure fortement imbriquée, la gestion des données peut devenir plus complexe.
- MongoDB a des limites sur la taille maximale des documents donc dans le cas où la base de données sera trop volumineuse ca pose un problème.

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

Vous trouverez dans le script mongoDB les requêtes de créations des collections ainsi que l'insertion des données.

### Vérification des insertions et créations :

#### 1- vérification de la création des collections :

```
Banque> show collections
Agence
Compte
Succursale
Banque>
```

#### 2- vérification des insertions dans la collection Succursale :

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6635738006208a4458117b7b'),
    '1': ObjectId('6635738006208a4458117b7c'),
    '2': ObjectId('6635738006208a4458117b7d'),
    '3': ObjectId('6635738006208a4458117b7e'),
    '4': ObjectId('6635738006208a4458117b7f'),
    '5': ObjectId('6635738006208a4458117b80')
  }
}
```

#### 3- vérification des insertions dans la collection Compte :

The screenshot shows the MongoDB Compass interface. At the top, there are tabs for 'Banque', 'Compte', and 'question3again'. Below the tabs, the path 'localhost:27017 > Banque > Compte' is displayed. The 'Documents' tab is selected, showing a list of documents. The first document is expanded, showing the following fields and values:

- `_id`: ObjectId('6635a29e85c517ae99117cd4')
- `NumCompte`: 6
- `dateOuverture`: 2019-05-15T00:00:00.000+00:00
- `etatCompte`: "Actif"
- `Solde`: 0
- `NumAgence`: 101
- `Client`: Object
- `Operations`: Array (2)
- `Prets`: Array (2)

#### 4- vérification des insertions dans la collection Agence:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66357c4cf5466eb614117b7b'),
    '1': ObjectId('66357c4cf5466eb614117b7c'),
    '2': ObjectId('66357c4cf5466eb614117b7d'),
    '3': ObjectId('66357c4cf5466eb614117b7e'),
    '4': ObjectId('66357c4cf5466eb614117b7f'),
    '5': ObjectId('66357c4cf5466eb614117b80'),
    '6': ObjectId('66357c4cf5466eb614117b81'),
    '7': ObjectId('66357c4cf5466eb614117b82'),
    '8': ObjectId('66357c4cf5466eb614117b83'),
    '9': ObjectId('66357c4cf5466eb614117b84'),
    '10': ObjectId('66357c4cf5466eb614117b85'),
    '11': ObjectId('66357c4cf5466eb614117b86'),
    '12': ObjectId('66357c4cf5466eb614117b87'),
    '13': ObjectId('66357c4cf5466eb614117b88'),
    '14': ObjectId('66357c4cf5466eb614117b89'),
    '15': ObjectId('66357c4cf5466eb614117b8a'),
    '16': ObjectId('66357c4cf5466eb614117b8b'),
    '17': ObjectId('66357c4cf5466eb614117b8c'),
    '18': ObjectId('66357c4cf5466eb614117b8d'),
    '19': ObjectId('66357c4cf5466eb614117b8e'),
    '20': ObjectId('66357c4cf5466eb614117b8f'),
    '21': ObjectId('66357c4cf5466eb614117b90'),
    '22': ObjectId('66357c4cf5466eb614117b91'),
    '23': ObjectId('66357c4cf5466eb614117b92'),
    '24': ObjectId('66357c4cf5466eb614117b93'),
    '25': ObjectId('66357c4cf5466eb614117b94'),
    '26': ObjectId('66357c4cf5466eb614117b95'),
    '27': ObjectId('66357c4cf5466eb614117b96'),
    '28': ObjectId('66357c4cf5466eb614117b97'),
    '29': ObjectId('66357c4cf5466eb614117b98')
  }
}
```

Affichage du nombre total des documents dans chaque collection :

Requête :

```
var totalDocumentsCount = db.Compte.countDocuments();

var distinctNumAgenceCount = db.Agence.distinct("NumAgence").length;

var distinctNumSuccursaleCount = db.Succursale.distinct("NumSucc").length;

print("Total documents in Compte collection: " + totalDocumentsCount);
print("Distinct NumAgence count: " + distinctNumAgenceCount);
print("Distinct NumSuccursale count: " + distinctNumSuccursaleCount);
```



### Capture d'écran de résultat sur le mongo shell :

```
Banque> print("Total documents in Compte collection: " + totalDocumentsCount);  
Total documents in Compte collection: 100  
  
Banque> print("Distinct NumAgence count: " + distinctNumAgenceCount);  
Distinct NumAgence count: 30  
  
Banque> print("Distinct NumSuccursale count: " + distinctNumSuccursaleCount);  
Distinct NumSuccursale count: 6
```

## C- Répondre aux requêtes suivantes :

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

Requête :

```
db.Compte.aggregate([
  { $match: { NumAgence: 102 } },
  { $unwind: "$Prets" },
  { $project: { _id: 0, NumPret: "$Prets.NumPret", montantOp:
"$Prets.montantOp", dateEffet: "$Prets.dateEffet", duree: "$Prets.duree",
typePret: "$Prets.typePret", tauxInteret: "$Prets.tauxInteret", montantEcheance:
"$Prets.montantEcheance" } }
]).forEach(printjson);
```

Capture d'écran de résultat sur le mongo shell :

```
{
  NumPret: 2002,
  montantOp: 25000.1,
  dateEffet: ISODate('2023-06-15T00:00:00.000Z'),
  duree: '48 months',
  typePret: 'Vehicule',
  tauxInteret: 6.1,
  montantEcheance: 1000.1
}
{
  NumPret: 2334,
  montantOp: 10000.5,
  dateEffet: ISODate('2023-01-10T00:00:00.000Z'),
  duree: '24 mois',
  typePret: 'ANSEJ',
  tauxInteret: 5.5,
  montantEcheance: 500.5
}
{
  NumPret: 1032,
  montantOp: 8000.5,
  dateEffet: ISODate('2022-08-15T00:00:00.000Z'),
  duree: '12 mois',
  typePret: 'ANSEJ',
  tauxInteret: 6.8,
  montantEcheance: 600.5
}
{
  NumPret: 53,
  montantOp: 53.95,
  dateEffet: ISODate('2024-11-30T05:28:52.457Z'),
  duree: '2 years',
  typePret: 'ANJEM',
  tauxInteret: 4.57,
  montantEcheance: 517.22
}
```

### Explication de la Requête (Ou vérification si besoin ) :

1. Filtrage des comptes: On sélectionne les comptes qui appartiennent à l'agence numéro 102.
2. Décomposition des prêts: Les prêts de chaque compte sont décomposés en documents individuels.
3. Projection des champs: On choisit spécifiquement certains champs des prêts pour l'affichage.
4. Projection des résultats: Les résultats sont affichés sous forme JSON.

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

### Requête :

```
var succursales = db.Succursale.find({ region: "Nord" }, { Agences: 1 });

var agencyNumbers = [];
succursales.forEach(function(succursale) {
    agencyNumbers = agencyNumbers.concat(succursale.Agences);
});

db.Compte.aggregate([
    { $match: { NumAgence: { $in: agencyNumbers } } },
    { $unwind: "$Prets" },
    { $project: { _id: 0, NumPret: "$Prets.NumPret", NumAgence: 1, NumCompte: 1,
    "Client.NumClient": 1, montantOp: "$Prets.montantOp" } }
]).forEach(printjson);
```

### Explication de la Requête (Ou vérification si besoin ) :

1. Récupération des numéros d'agences: On récupère les numéros des agences appartenant à la région "Nord" depuis la collection Succursale et les stocke dans un tableau appelé `agencyNumbers`.
2. Filtrage des comptes : On utilise l'opérateur `\$match` pour sélectionner les comptes ayant pour numéro d'agence l'un des numéros contenus dans le tableau `agencyNumbers`.
3. Décomposition des prêts : Les prêts de chaque compte sont décomposés en documents individuels.
4. Projection des champs : On sélectionne certains champs spécifiques des prêts et des clients associés pour l'affichage.
5. Impression des résultats : Les résultats sont imprimés sous forme JSON.

Capture d'écran de résultat sur le mongo shell :

```
{
  NumCompte: 6,
  NumAgence: 101,
  Client: {
    NumClient: 123
  },
  NumPret: 10011,
  montantOp: 10000.5
}
{
  NumCompte: 6,
  NumAgence: 101,
  Client: {
    NumClient: 123
  },
  NumPret: 10032,
  montantOp: 8000.5
}
{
  NumCompte: 2,
  NumAgence: 101,
  Client: {
    NumClient: 124
  },
  NumPret: 10039,
  montantOp: 12000.5
}
{
  NumCompte: 3,
  NumAgence: 101,
  Client: {
    NumClient: 124
  },
  NumPret: 1009,
  montantOp: 10000.5
}
```

**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.**

Requête :

```
db.Compte.aggregate([
  {
    $unwind: "$Prets" },
  {
    $group: {
      _id: "$NumAgence",
      totalPrets: { $sum: 1 } }
  },
  {
    $sort: { totalPrets: -1 }
  }
])
```

```

    },
    {
      $project: {
        _id: 0,
        NumAgence: "$_id",
        totalPrets: 1
      }
    },
    {
      $out: "Agence-NbPrêts" }
  });

```

Capture d'écran de résultat sur le mongo shell :

```

Banque> db.getCollection("Agence-NbPrêts").find().pretty()
[
  {
    _id: ObjectId('6635acc9b3a370b13dbe1dffa'),
    totalPrets: 9,
    NumAgence: 113
  },
  {
    _id: ObjectId('6635acc9b3a370b13dbe1e00'),
    totalPrets: 8,
    NumAgence: 105
  },
  {
    _id: ObjectId('6635acc9b3a370b13dbe1e01'),
    totalPrets: 8,
    NumAgence: 101
  },
  {
    _id: ObjectId('6635acc9b3a370b13dbe1e02'),
    totalPrets: 7,
    NumAgence: 120
  },
]

```

Explication de la Requête (Ou vérification si besoin) :

1. Décomposition des prêts : Chaque prêt est séparé en documents individuels.
2. Regroupement par agence : Les prêts sont regroupés selon le numéro d'agence.
3. Comptage des prêts par agence : On compte le nombre total de prêts pour chaque agence.
4. Tri par nombre de prêts : Les résultats sont triés par ordre décroissant du nombre total de prêts.
5. Projection et sauvegarde : On sélectionne et sauvegarde le numéro d'agence et le total des prêts dans une nouvelle collection.

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

**Requête :**

```
var pipeline = [
  { $unwind: "$Prets" },
  { $match: { "Prets.typePret": "ANSEJ" } },
  {
    $project: {
      _id: 0,
      NumPret: "$Prets.NumPret",
      numClient: "$Client.NumClient",
      MontantPrêt: "$Prets.montantOp",
      dateEffet: "$Prets.dateEffet"
    }
  }
];
var result = db.Compte.aggregate(pipeline).toArray();

db.createCollection("Prêt-ANSEJ");
db["Prêt-ANSEJ"].insertMany(result);
db["Prêt-ANSEJ"].find().forEach(printjson);
```

**Capture d'écran de résultat sur le mongo shell :**

```
Banque> db["Prêt-ANSEJ"].find().forEach(printjson);
{
  _id: ObjectId('6635ad3085c517ae99117ee2'),
  NumPret: 1011,
  numClient: 123,
  'MontantPrêt': 10000.5,
  dateEffet: ISODate('2023-01-10T00:00:00.000Z')
}
{
  _id: ObjectId('6635ad3085c517ae99117ee3'),
  NumPret: 1032,
  numClient: 123,
  'MontantPrêt': 8000.5,
  dateEffet: ISODate('2022-08-15T00:00:00.000Z')
}
{
  _id: ObjectId('6635ad3085c517ae99117ee4'),
  NumPret: 2334,
  numClient: 133,
  'MontantPrêt': 10000.5,
  dateEffet: ISODate('2023-01-10T00:00:00.000Z')
}
{
  _id: ObjectId('6635ad3085c517ae99117ee5'),
  NumPret: 1032,
  numClient: 133,
  'MontantPrêt': 8000.5,
  dateEffet: ISODate('2022-08-15T00:00:00.000Z')
}
{
  _id: ObjectId('6635ad3085c517ae99117ee6'),
  NumPret: 1,
  numClient: 1047,
  'MontantPrêt': 364.32,
  dateEffet: ISODate('2024-09-13T17:50:58.559Z')
}
```

### Explication de la Requête (Ou vérification si besoin) :

Voici une vérification:

```
Banque> db["Prêt-ANSE"].insertMany(result);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6635ad3085c517ae99117ee2'),
    '1': ObjectId('6635ad3085c517ae99117ee3'),
    '2': ObjectId('6635ad3085c517ae99117ee4'),
    '3': ObjectId('6635ad3085c517ae99117ee5'),
    '4': ObjectId('6635ad3085c517ae99117ee6'),
    '5': ObjectId('6635ad3085c517ae99117ee7'),
    '6': ObjectId('6635ad3085c517ae99117ee8'),
    '7': ObjectId('6635ad3085c517ae99117ee9'),
    '8': ObjectId('6635ad3085c517ae99117eea'),
    '9': ObjectId('6635ad3085c517ae99117eeb'),
    '10': ObjectId('6635ad3085c517ae99117eec'),
    '11': ObjectId('6635ad3085c517ae99117eed'),
    '12': ObjectId('6635ad3085c517ae99117eee'),
    '13': ObjectId('6635ad3085c517ae99117eef'),
    '14': ObjectId('6635ad3085c517ae99117ef0'),
    '15': ObjectId('6635ad3085c517ae99117ef1'),
  }
}
```

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

### Requête :

```
var pipeline = [
  { $unwind: "$Prets" },

  { $match: { "Client.TypeClient": "Particulier" } },
  {
    $project: {
      _id: 0,
      NumClient: "$Client.NumClient",
      NomClient: "$Client.NomClient",
      NumPret: "$Prets.NumPret",
      montantPrêt: "$Prets.montantOp"
    }
  }
];

var result = db.Compte.aggregate(pipeline).toArray();
printjson(result);
```

### Capture d'écran de résultat sur le mongo shell :

```
Banque> printjson(result);
[
  {
    NumClient: 123,
    NomClient: 'John Doe',
    NumPret: 10011,
    'montantPrêt': 10000.5
  },
  {
    NumClient: 123,
    NomClient: 'John Doe',
    NumPret: 10032,
    'montantPrêt': 8000.5
  },
  {
    NumClient: 124,
    NomClient: 'Jane Smith',
    NumPret: 10039,
    'montantPrêt': 12000.5
  },
  {
    NumClient: 124,
    NomClient: 'Jane Smith',
    NumPret: 1009,
    'montantPrêt': 10000.5
  },
  {
    NumClient: 124,
    NomClient: 'Jane Smith',
    NumPret: 10098,
    'montantPrêt': 8000.5
  },
  {
    NumClient: 125,
    NomClient: 'Alice Johnson',
    NumPret: 1004,
    'montantPrêt': 15000.5
  },
],
```

### Explication de la Requête (Ou vérification si besoin) :

1. Décomposition des prêts : Chaque prêt est séparé en documents individuels à l'aide de '\$unwind'.
2. Filtrage des clients particuliers : On ne garde que les prêts des clients de type "Particulier" en utilisant '\$match'.
3. Projection des champs : On sélectionne spécifiquement les champs NumClient, NomClient, NumPret et montantPrêt pour l'affichage à l'aide de '\$project'.
4. Exécution de l'agrégation : Le pipeline est exécuté sur la collection Compte, et le résultat est stocké dans un tableau à l'aide de 'toArray()'.
5. Impression des résultats : Les résultats sont imprimés sous forme JSON.



**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.**

**Requête :**

```
db.Compte.updateMany(  
  {  
    Solde: 0,  
    "Prets.dateEffet": { $lt: ISODate("2021-01-01") }  
  },  
  {  
    $inc: { "Prets.$.montantEcheance": 2000 }  
  }  
)
```

**Capture d'écran de résultat sur le mongo shell :**

```
Banque> db.Compte.updateMany(  
...   {  
...     Solde: 0,  
...     "Prets.dateEffet": { $lt: ISODate("2021-01-01") }  
...   },  
...   {  
...     $inc: { "Prets.$.montantEcheance": 2000 }  
...   }  
... )  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 2,  
  modifiedCount: 2,  
  upsertedCount: 0  
}
```

**Explication de la Requête (Ou vérification si besoin) :**

**Avant la modification :**

```
[  
  { NumPret: 10011, montantEcheance: 500.5 },  
  { NumPret: 10032, montantEcheance: 600.5 },  
  { NumPret: 10009, montantEcheance: 500.5 },  
  { NumPret: 10098, montantEcheance: 600.5 }  
]
```

**Requête de vérification :**

```
db.Compte.aggregate([
  {
    $match: {
      "Prets.dateEffet": { $lt: ISODate("2021-01-01") },
      "Solde": 0 // Filter for Prets where solde is 0.0
    }
  },
  {
    $unwind: "$Prets" // Unwind the Prets array to deconstruct the documents
  },
  {
    $project: {
      _id: 0,
      NumPret: "$Prets.NumPret",
      montantEcheance: "$Prets.montantEcheance",
      // Add more fields here that you want to display
    }
  }
])
```

**Après la modification :**

```
[
  { NumPret: 10011, montantEcheance: 2500.5 },
  { NumPret: 10032, montantEcheance: 2600.5 },
  { NumPret: 1009, montantEcheance: 2500.5 },
  { NumPret: 10098, montantEcheance: 2600.5 }
]
```

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

### Requête :

```
//map function
var mapFunction = function() {
  var numAgence = this.NumAgence;
  this.Prets.forEach(function(pret) {
    emit(numAgence, 1);
  });
};

// Reduce function
var reduceFunction = function(key, values) {
  var total = Array.sum(values);
  emit(key, total);
  return total;
};

// Run Map-Reduce operation
db.Compte.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "question3again",
    finalize: function(key, reducedValue) {
      return { _id: key, totalPrets: reducedValue };
    }
  }
);

var result = db.question3again.find();

result.sort({ "value.totalPrets": -1 }).forEach(printjson);
```

### Explication de la Requête (Ou vérification si besoin) :

1. **Fonction de Map** : La fonction `mapFunction` itère à travers chaque document de la collection `Compte`, extrait le numéro d'agence et émet une paire clé-valeur pour chaque prêt, avec la clé étant le numéro d'agence et la valeur étant 1.
2. **Fonction de Réduction** : La fonction `reduce Function` prend en entrée une clé (numéro d'agence) et une liste de valeurs (le nombre de prêts pour cette agence) et calcule la somme des valeurs pour chaque clé.
3. **Exécution de l'opération Map-Reduce** : L'opération Map-Reduce est lancée sur la collection `Compte` en utilisant les fonctions de map et de réduction définies. Les résultats sont stockés dans la collection `question3again`.

4. Finalisation des résultats : La fonction `finalize` est utilisée pour formater les résultats finaux en un objet avec une clé `\_id` représentant le numéro d'agence et une valeur `totalPrets` représentant le nombre total de prêts pour cette agence.

5. Récupération et tri des résultats : Les résultats sont récupérés de la collection `question3again` et triés par nombre total de prêts dans l'ordre décroissant, puis chaque document est imprimé au format JSON.

Capture d'écran de résultat sur le mongo shell :

```
{
  _id: 113,
  value: {
    _id: 113,
    totalPrets: 9
  }
}
{
  _id: 101,
  value: {
    _id: 101,
    totalPrets: 8
  }
}
{
  _id: 105,
  value: {
    _id: 105,
    totalPrets: 8
  }
}
{
  _id: 120,
  value: {
    _id: 120,
    totalPrets: 7
  }
}
{
  _id: 121,
  value: {
    _id: 121,
    totalPrets: 6
  }
}
```

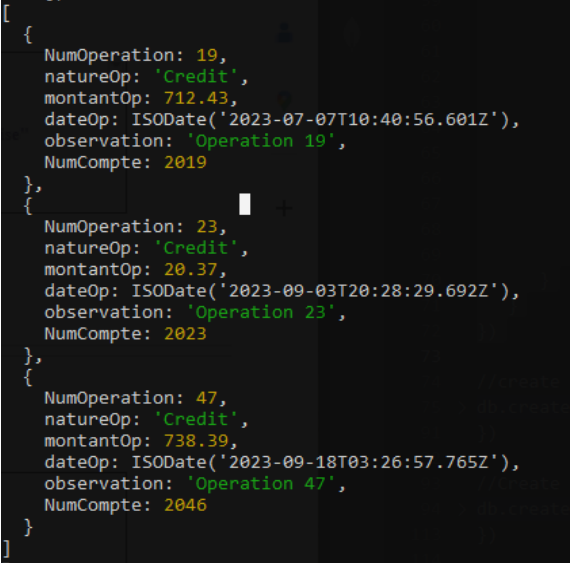
**8- Avec votre conception, peut-on répondre à la requête suivante : Afficher toutes les opérations de crédit effectuées sur les comptes des clients de type « Entreprise » pendant l'année 2023. Justifiez votre réponse.**

Oui on peut répondre à la question 8 par la requête suivante :

Requête :

```
db.Compte.aggregate([
  { $unwind: "$Operations" },
  {
    $match: {
      "Client.TypeClient": "Entreprise",
      "Operations.natureOp": "Credit",
      "Operations.dateOp": {
        $gte: ISODate("2023-01-01"),
        $lt: ISODate("2024-01-01")
      }
    }
  },
  {
    $project: {
      _id: 0,
      NumOperation: "$Operations.NumOperation",
      natureOp: "$Operations.natureOp",
      montantOp: "$Operations.montantOp",
      dateOp: "$Operations.dateOp",
      observation: "$Operations.observation",
      NumCompte: "$Operations.NumCompte"
    }
  }
])
```

Capture d'écran de résultat sur le mongo shell :



```
[
  {
    NumOperation: 19,
    natureOp: 'Credit',
    montantOp: 712.43,
    dateOp: ISODate('2023-07-07T10:40:56.601Z'),
    observation: 'Operation 19',
    NumCompte: 2019
  },
  {
    NumOperation: 23,
    natureOp: 'Credit',
    montantOp: 20.37,
    dateOp: ISODate('2023-09-03T20:28:29.692Z'),
    observation: 'Operation 23',
    NumCompte: 2023
  },
  {
    NumOperation: 47,
    natureOp: 'Credit',
    montantOp: 738.39,
    dateOp: ISODate('2023-09-18T03:26:57.765Z'),
    observation: 'Operation 47',
    NumCompte: 2046
  }
]
```

### Explication de la Requête (Ou vérification si besoin ) :

1. **Décomposition des opérations** : Chaque opération est décomposée en documents individuels à l'aide de l'opérateur ``$unwind``.
2. **Filtrage des opérations** : On filtre les opérations pour ne garder que celles effectuées par des clients de type "Entreprise", de nature "Credit" et ayant eu lieu entre le 1er janvier 2023 et le 1er janvier 2024.
3. **Projection des champs** : On sélectionne spécifiquement certains champs des opérations pour l'affichage, tels que NumOperation, natureOp, montantOp, dateOp, observation et NumCompte.

## **D- Analyse**

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

Effectivement, le traitement des données relatives aux prêts semble relativement simple et efficace dans la mesure où toutes les informations pertinentes sont déjà incluses dans la collection *Compte*, notamment celles concernant les prêts. Cela a permis d'exécuter toutes les requêtes sans difficulté majeure. Cependant, il est important de reconnaître que l'imbrication excessive des données peut présenter des inconvénients dans les bases de données volumineuses.

Malgré la facilité actuelle, l'utilisation intensive de l'imbrication peut atteindre ses limites et introduire des complexités supplémentaires, notamment en termes de performances et de gestion des redondances. Dans des bases de données plus volumineuses ou à mesure que le nombre de transactions et de clients augmente, cette approche peut devenir moins pratique et poser des défis en termes de maintenance et de scalabilité. Il est donc crucial de considérer les besoins à long terme et d'évaluer les différentes options de modélisation de données pour garantir une conception optimale et évolutive de la base de données.

# Conclusion :

**En conclusion, la comparaison entre SQL3 et MongoDB révèle deux approches distinctes pour la modélisation et la gestion de bases de données, chacune avec ses propres forces et faiblesses.** SQL3 offre une structure relationnelle robuste, idéale pour les applications nécessitant une cohérence et une intégrité des données rigoureuses. En revanche, MongoDB présente une flexibilité et une évolutivité horizontale qui en font un choix attrayant pour les applications où la structure des données peut évoluer rapidement ou est moins définie. **Le choix entre les deux dépendra donc des besoins spécifiques du projet**, mais il est crucial de veiller à ce que la base de données choisie soit capable de répondre aux exigences évolutives et de maintenance à long terme, assurant ainsi une gestion efficace des données sur la durée de vie de l'application.