



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des sciences et des technologies Houari Boumediene

Faculté d'informatique

Département IA &SD



Spécialité : Systèmes Informatiques Intelligents

Mini- Projet BDA

SQL3-Oracle et NoSQL (MongoDB)

Réalisé par :

Zenaini Afaf Farah
Mezioug Liza

Année Académique : 2024-2025

TABLE DES MATIÈRES

Introduction	
Partie I : Relationnel-Objet	
A. Modélisation orientée objet	
B. Création des Tablespaces et utilisateur	
C. Langage de définition de données	
D. Création des instances dans les tables	
E. Langage d'interrogation de données	
Partie II : NoSQL – Modèle orienté « documents »	
A. Modélisation orientée document	
B. Remplissage la base de données	
C. Requêtes MongoDB	
D. Analyse	
Conclusion	

Introduction

Dans le cadre de ce projet de base de données avancée (BDA), nous nous intéressons à la modélisation, la création et l'interrogation d'une base de données hybride combinant des technologies relationnelles (SQL3-Oracle) et NoSQL (MongoDB). Le projet porte sur la gestion d'un réseau de transport urbain au sein d'une ville intelligente (smart city), impliquant différents moyens de transport (métro, bus, tramway, train), des lignes, stations, tronçons, navettes et voyages associés.

La première partie du projet est axée sur la modélisation relationnelle-objet. Elle inclut la conception UML du système, sa transformation en schéma relationnel, la création des types abstraits et des associations, ainsi que la mise en œuvre de requêtes complexes en SQL3. Cette partie nous permet d'exploiter les capacités de modélisation objet offertes par Oracle et d'illustrer l'intégration entre les objets et les relations dans un contexte réel.

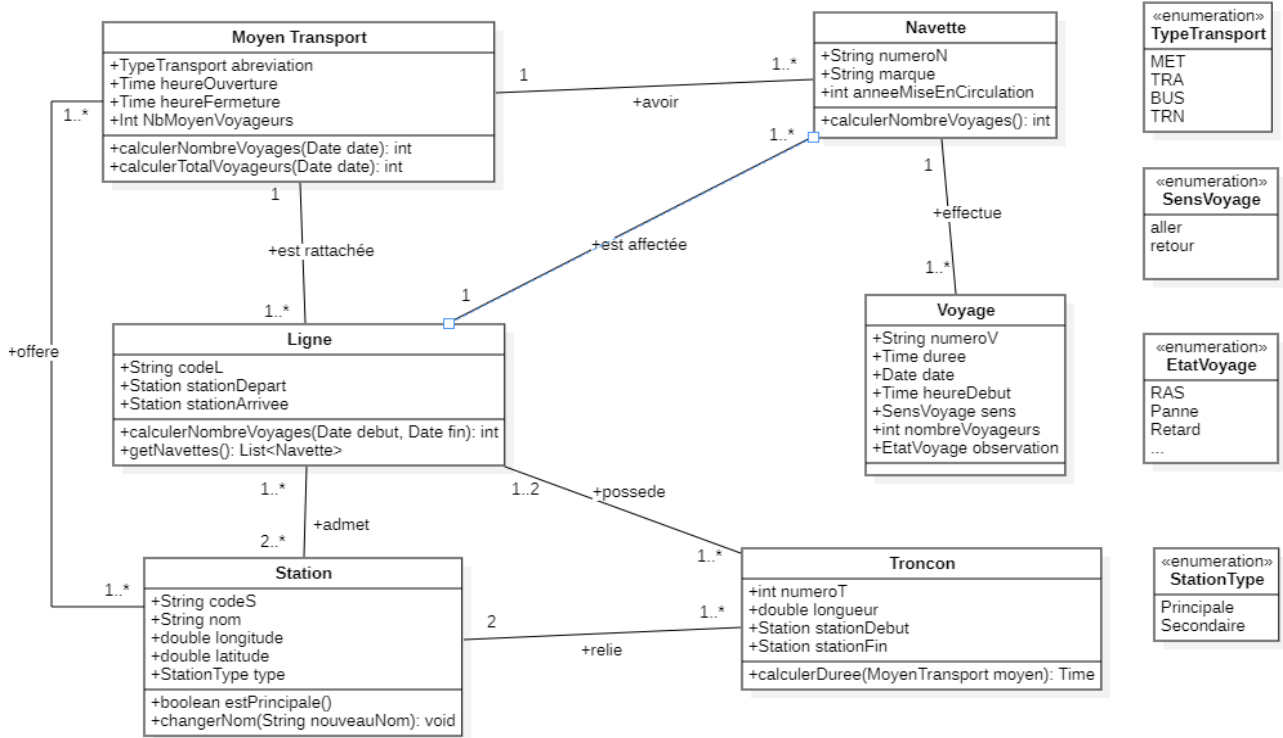
La seconde partie du projet explore une approche orientée documents avec MongoDB. Elle consiste à proposer une modélisation adaptée aux spécificités du NoSQL, à insérer les données, et à formuler des requêtes complexes en exploitant les avantages de la structure hiérarchique de MongoDB, notamment en termes de flexibilité et de performances pour certaines analyses.

À travers ce travail, nous mettons en pratique des compétences essentielles en conception de bases de données, en manipulation de données complexes et en comparaison entre les paradigmes relationnels et NoSQL. Ce rapport retrace les étapes principales de notre avancement, les choix techniques effectués, ainsi que les résultats obtenus jusqu'à présent.

Partie I : Relationnel-Objet

A. Modélisation orientée objet

1. Etablissement d'un diagramme de classes UML relatif à cette étude de cas :



2. Transformation de ce diagramme en un schéma relationnel :

MoyenTransport(abreviation, heureOuverture, heureFermeture, NbMoyenVoyageurs)

Navette (numeroN, marque, anneeMiseEnCirculation, codeL*, abreviation*)

Station (codeS, nom, longitude, latitude, type)

Ligne (codeL, stationDepart, stationArrivee, abreviation*)

Tronçon (numeroT, stationDebut, stationFin, longueur, codeS*, codeL*)

Voyage (numeroV, date, heureDebut, duree, sens, nombreVoyageurs, observation, numeroN*)

Ligne_Station (codeL*, codeS*)

MoyenTransport_Station (abreviation*, codeS*)

B. Création des Tablespaces et utilisateur

3. Création de deux Tablespaces SQL3_TBS et SQL3_TempTBS :

```
CREATE TABLESPACE SQL3_TBS DATAFILE 'sql3_tbs.dbf' SIZE 100M;  
CREATE TEMPORARY TABLESPACE SQL3_TempTBS TEMPFILE 'sql3_temp_tbs.dbf' SIZE  
100M;
```

4. Création d'un utilisateur SQL3 en lui attribuant les deux tablespaces créées précédemment :

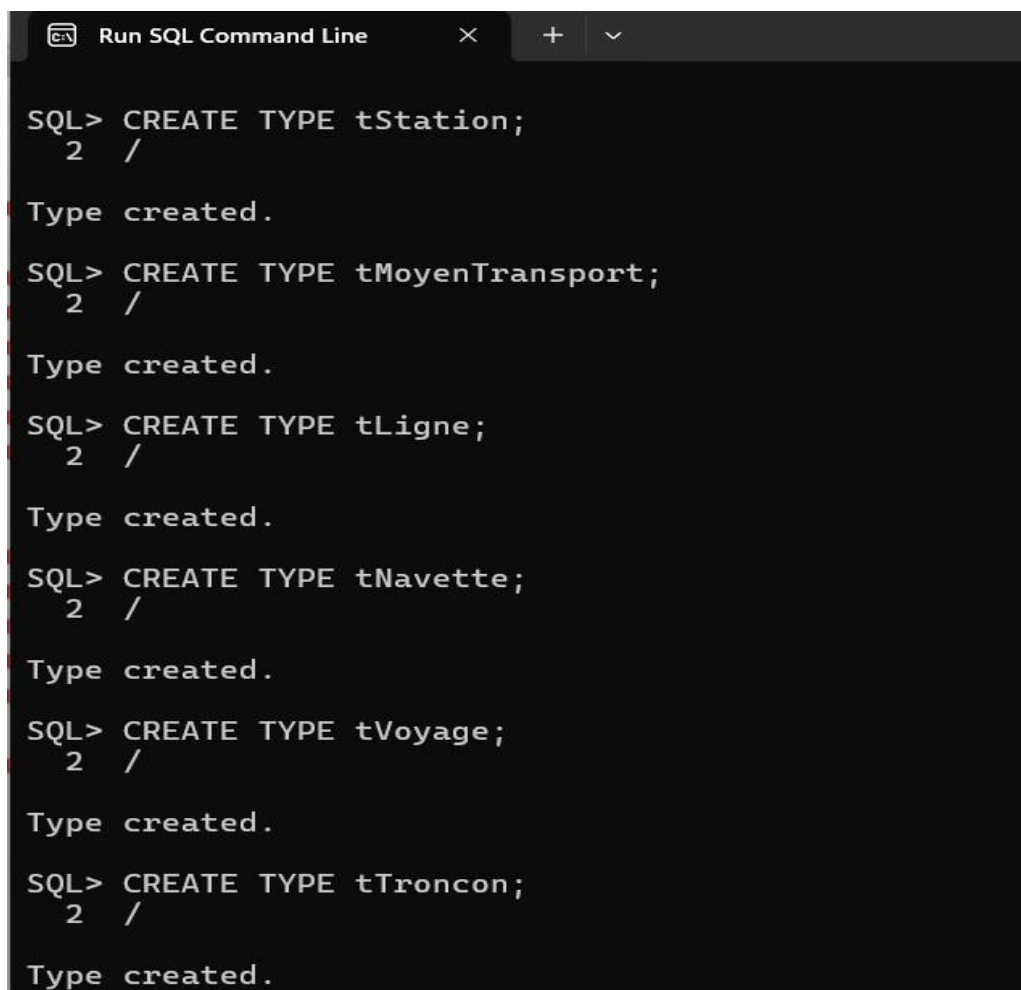
```
CREATE USER SQL3 IDENTIFIED BY sql3_password  
DEFAULT TABLESPACE SQL3_TBS  
TEMPORARY TABLESPACE SQL3_TempTBS;
```

5. En donnant tous les privilèges à cet utilisateur :

```
GRANT ALL PRIVILEGES TO SQL3;
```

C- Langage de définition de données

6. En se basant sur le diagramme de classes établi, voici tous les types abstraits nécessaires avec définition de toutes les associations qui existent :



```
Run SQL Command Line × + ~  
  
SQL> CREATE TYPE tStation;  
2 /  
  
Type created.  
  
SQL> CREATE TYPE tMoyenTransport;  
2 /  
  
Type created.  
  
SQL> CREATE TYPE tLigne;  
2 /  
  
Type created.  
  
SQL> CREATE TYPE tNavette;  
2 /  
  
Type created.  
  
SQL> CREATE TYPE tVoyage;  
2 /  
  
Type created.  
  
SQL> CREATE TYPE tTroncon;  
2 /  
  
Type created.
```

```

SQL> CREATE TYPE tSetLigne AS TABLE OF REF tLigne;
2 /

Type created.

SQL> CREATE TYPE tSetVoyage AS TABLE OF REF tVoyage;
2 /

Type created.

SQL> CREATE TYPE tSetTroncon AS TABLE OF REF tTroncon;
2 /

Type created.

SQL> CREATE TYPE tSetNavette AS TABLE OF REF tNavette;
2 /

Type created.

SQL> CREATE TYPE tSetStation AS TABLE OF REF tStation;
2 /

Type created.

SQL> CREATE TYPE tSetMoyenTransport AS TABLE OF REF tMoyenTransport;
2 /

Type created.

```

```

SQL> CREATE OR REPLACE TYPE tNavette AS OBJECT (
2     numeroN INTEGER,
3     marque VARCHAR2(50),
4     anneeMiseEnCirculation INTEGER,
5     MoyenTransportN REF tMoyenTransport,
6     LigneN REF tLigne,
7     Voyages tSetVoyage
8 );
9 /

Type created.

SQL> CREATE OR REPLACE TYPE tTroncon AS OBJECT (
2     numeroT INTEGER,
3     stationDebut VARCHAR2(100),
4     stationFin VARCHAR2(100),
5     longueur FLOAT,
6     StationsT tSetStation,
7     LignesT tSetLigne
8 );
9 /

Type created.

SQL> CREATE OR REPLACE TYPE tVoyage AS OBJECT (
2     numeroV INTEGER,
3     dateVoyage DATE,
4     heureDebut VARCHAR2(5),           -- exemple : '08:30'
5     duree VARCHAR2(5),               -- exemple : '01:30'
6     sens VARCHAR2(10),               -- Aller / Retour
7     nombreVoyageurs INTEGER,
8     observation VARCHAR2(20),
9     Navette REF tNavette
10 );
11 /

Type created.

```

```

CREATE OR REPLACE TYPE tStation AS OBJECT (

    codeS VARCHAR2(10),
    nom VARCHAR2(100),
    latitude FLOAT,
    longitude FLOAT,
    type VARCHAR2(20),
    lignes tSetLigne,
    troncons tSetTroncon,
    moyens tSetMoyenTransport
);

CREATE OR REPLACE TYPE tMoyenTransport AS OBJECT (
    abreviation VARCHAR2(10),
    heureOuverture VARCHAR2(5),      -- format HH:MI
    heureFermeture VARCHAR2(5),
    NbMoyenVoyageurs INTEGER,
    LigneMT tSetLigne,               -- REF vers lignes
    StationMT tSetStation,           -- REF vers stations
    NavetteMT tSetNavette            -- REF vers navettes
);
/

CREATE OR REPLACE TYPE tLigne AS OBJECT (
    codeL VARCHAR2(10),
    stationDepart VARCHAR2(100),
    stationArrivee VARCHAR2(100),
    MoyenTransportL REF tMoyenTransport,
    NavetteL tSetNavette,
    StationsL tSetStation,
    TronconL tSetTroncon
);
/

```

7. Définition de méthodes permettant de :

- ❖ Calculer pour chaque navette, le nombre total de voyages effectués :

```

ALTER TYPE tNavette ADD MEMBER FUNCTION nbVoyages RETURN INTEGER CASCADE;

CREATE OR REPLACE TYPE BODY tNavette AS
    MEMBER FUNCTION nbVoyages RETURN INTEGER IS
        total INTEGER;
    BEGIN
        SELECT COUNT(*) INTO total
        FROM TABLE(self.Voyages);

        RETURN total;
    END;
END;/

```

Cette méthode fonctionne ainsi :

- Elle utilise la clause TABLE (self.Voyages) pour parcourir la collection de voyages de la navette .
 - Elle applique une requête SQL COUNT (*) pour compter combien de voyages sont enregistrés.
 - Le résultat est stocké dans une variable locale total puis retourné.
- ❖ Retourner pour chaque ligne, la liste des navettes qui la desservent :

```
ALTER TYPE tLigne ADD MEMBER FUNCTION listeNavettes RETURN tSetNavette
CASCADE;

CREATE OR REPLACE TYPE BODY tLigne AS
  MEMBER FUNCTION listeNavettes RETURN tSetNavette IS
    resultat tSetNavette := tSetNavette();
  BEGIN
    SELECT CAST(COLLECT(REF(n)) AS tSetNavette)
    INTO resultat
    FROM Navette n
    WHERE Deref(n.LigneN) = SELF;

    RETURN resultat;
  END;
END;
/
```

La fonction listeNavettes:

- Crée d'abord une collection vide pour accueillir les résultats,
 - Puis elle effectue une recherche dans toutes les navettes,
 - Elle sélectionne uniquement celles qui sont liées à la ligne courante (SELF),
 - Et les rassemble dans une collection de type tSetNavette qu'elle retourne.
- ❖ Calculer pour une ligne (de numéro donné), le nombre de voyages effectués durant une période :

```
ALTER TYPE tLigne ADD MEMBER FUNCTION nbVoyagesPeriode(date_debut DATE,
date_fin DATE) RETURN INTEGER CASCADE;

CREATE OR REPLACE TYPE BODY tLigne AS
  MEMBER FUNCTION nbVoyagesPeriode(date_debut DATE, date_fin DATE) RETURN
INTEGER IS
  total INTEGER := 0;
  BEGIN
    SELECT COUNT(*)
    INTO total
    FROM Navette n
```



```

, TABLE(n.Voyages) v -- 'Voyages' est une collection
WHERE REF(n) = SELF -- La ligne actuelle est liée à la navette via REF
AND v.dateVoyage BETWEEN date_debut AND date_fin;
RETURN total;
END;
END;/

```

La méthode :

- Parcourt toutes les navettes associées à la ligne (SELF).
- Accède à la collection de voyages (Voyages) de chaque navette.
- Filtre les voyages dont la date (dateVoyage) est comprise entre date_debut et date_fin.
- Compte le nombre total de voyages correspondants.
- Retourne ce total en tant que valeur entière.

Changer le nom de la station « BEZ » par « Univ » dans toutes les lignes/tronçons comportant cette station :

```

ALTER TYPE tLigne ADD MEMBER PROCEDURE majNomStation CASCADE;

CREATE OR REPLACE TYPE BODY tLigne AS
  MEMBER PROCEDURE majNomStation IS
  BEGIN
    FOR s IN (
      SELECT VALUE(st)
      FROM TABLE(SELf.StationsL) st
      WHERE st.nom = 'BEZ')
    LOOP
      s.nom := 'Univ';
    END LOOP;
    FOR t IN (
      SELECT VALUE(tr)
      FROM TABLE(SELf.TronconL) tr
      )
    LOOP
      FOR st IN (
        SELECT VALUE(st2)
        FROM TABLE(t.StationsT) st2
        WHERE st2.nom = 'BEZ'
        )
      LOOP
        st.nom := 'Univ';
      END LOOP;
    END LOOP;
  END;
END;/

```

La méthode :

- Parcourt toutes les stations directement associées à la ligne (`StationsL`).
 - Si une station porte le nom 'BEZ', elle est renommée en 'Univ'.
- Parcourt tous les tronçons de la ligne (`TronconL`).
 - Pour chaque tronçon, parcourt les stations qu'il contient (`StationsT`).
 - Si une station porte le nom 'BEZ', elle est également renommée en 'Univ'
- ❖ Calculer pour un moyen de transport donné (Exemple Métro), le nombre de voyages effectués à une date donnée (Exemple le 28-02-2025) et le nombre de voyageurs total :

```
CREATE OR REPLACE TYPE tStatMT AS OBJECT (  
    total_voyages INTEGER,  
    total_voyageurs INTEGER  
);  
/
```

Cela crée un type objet pour stocker deux informations :

- Le nombre total de voyages.
- Le nombre total de voyageurs

```
ALTER TYPE tMoyenTransport ADD MEMBER FUNCTION VoyageVoyageur(nomMT  
VARCHAR2, date_cible DATE) RETURN tStatMT CASCADE;  
CREATE OR REPLACE FUNCTION VoyageVoyageur(nomMT VARCHAR2, date_cible DATE)  
RETURN tStatMT IS  
    total_voyages INTEGER := 0;  
    total_voyageurs INTEGER := 0;  
BEGIN  
  
    SELECT COUNT(v.numeroV), NVL(SUM(v.nombreVoyageurs), 0)  
    INTO total_voyages, total_voyageurs  
    FROM Voyage v  
    WHERE v.dateVoyage = date_cible  
    AND EXISTS (  
        SELECT 1  
        FROM MoyenTransport mt, TABLE(mt.NavetteMT) n  
        WHERE mt.abreviation = nomMT  
        AND v.Navette = REF(n)  
    );  
  
    RETURN tStatMT(total_voyages, total_voyageurs);  
END;/
```

La méthode :

- Parcourt les voyages effectués à la date donnée (date_cible).
- Vérifie, pour chaque voyage, si la navette utilisée appartient au moyen de transport dont l'abréviation est nomMT.
- Utilise une sous-requête avec EXISTS pour filtrer ces voyages selon la collection de navettes de MoyenTransport.
- Compte le nombre de voyages correspondants (COUNT(v.numeroV)).
- Additionne le nombre total de voyageurs de ces voyages (SUM(v.nombreVoyageurs)), en remplaçant NULL par 0 si besoin (NVL).
- Stocke les résultats dans deux variables locales.
- Retourne ces résultats sous forme d'un objet tStatMT.

8. Définition des tables nécessaires à la base de données :

```
SQL> CREATE TABLE MoyenTransport OF tMoyenTransport
2   (PRIMARY KEY (abreviation))
3   NESTED TABLE LigneMT STORE AS TableLigneM
4   NESTED TABLE StationMT STORE AS TableStationM
5   NESTED TABLE NavetteMT STORE AS TableNavetteM;

Table created.

SQL> CREATE TABLE Ligne OF tLigne
2   (PRIMARY KEY (codeL))
3   NESTED TABLE NavetteL STORE AS TableNavetteL
4   NESTED TABLE StationsL STORE AS TableStationL
5   NESTED TABLE TronconL STORE AS TableTronconL;

Table created.

SQL> CREATE TABLE Navette OF tNavette
2   (PRIMARY KEY (numeroN))
3   NESTED TABLE Voyages STORE AS TableVoyageN;

Table created.

SQL> CREATE TABLE Troncon OF tTroncon
2   (PRIMARY KEY (numeroT))
3   NESTED TABLE StationsT STORE AS TableStationT
4   NESTED TABLE LignesT STORE AS TableLigneT;

Table created.
```

```
CREATE TABLE Station OF tStation
(PRIMARY KEY (codeS))
NESTED TABLE lignes STORE AS TableLigne
NESTED TABLE troncons STORE AS TableTroncon
NESTED TABLE moyens STORE AS TableMoyenTransport;

CREATE TABLE Voyage OF tVoyage
(PRIMARY KEY (numeroV));
```

```

SQL> ALTER TABLE Voyage MODIFY (sens
2    CHECK (LOWER(sens) IN ('aller', 'retour')));

Table altered.

SQL> ALTER TABLE Station MODIFY (type
2    CHECK (LOWER(type) IN ('principale', 'secondaire')));

Table altered.

SQL> ALTER TABLE MoyenTransport MODIFY (abreviation
2    CHECK (abreviation IN ('MET', 'TRA', 'BUS', 'TRN')));

Table altered.

SQL> ALTER TABLE Voyage MODIFY (observation
2    CHECK (observation IN ('RAS', 'Panne', 'Retard', 'Accident', 'Normal')
));

Table altered.

```

D- Création des instances dans les tables :

```

1. INSERTION DES MOYENS DE TRANSPORT
INSERT INTO MoyenTransport (abreviation, heureOuverture, heureFermeture,
NbMoyenVoyageurs)
VALUES ('BUS', '05:00', '23:00', 50);

INSERT INTO MoyenTransport (abreviation, heureOuverture, heureFermeture,
NbMoyenVoyageurs)
VALUES ('MET', '05:30', '23:30', 120);

INSERT INTO MoyenTransport (abreviation, heureOuverture, heureFermeture,
NbMoyenVoyageurs)
VALUES ('TRA', '06:00', '22:30', 80);

INSERT INTO MoyenTransport (abreviation, heureOuverture, heureFermeture,
NbMoyenVoyageurs)
VALUES ('TRN', '04:30', '22:00', 150);

-- 2. INSERTION DES STATIONS
-- Stations principales
INSERT INTO Station (codeS, nom, latitude, longitude, type)
VALUES ('S001', 'Alger Centre', 36.7539, 3.0589, 'principale');

INSERT INTO Station (codeS, nom, latitude, longitude, type)
VALUES ('S002', 'El Harrach', 36.7167, 3.1500, 'principale');

INSERT INTO Station (codeS, nom, latitude, longitude, type)
VALUES ('S003', 'BEZ', 36.7222, 3.1811, 'principale');

-- Stations secondaires
INSERT INTO Station (codeS, nom, latitude, longitude, type)

```

```

VALUES ('S006', 'Les Fusillés', 36.7514, 3.0631, 'secondaire');

INSERT INTO Station (codeS, nom, latitude, longitude, type)
VALUES ('S007', 'Place des Martyrs', 36.7758, 3.0597, 'secondaire');

- 3. INSERTION DES LIGNES
-- Lignes de Bus
INSERT INTO Ligne (codeL, stationDepart, stationArrivee)
VALUES ('B001', 'Alger Centre', 'El Harrach');

INSERT INTO Ligne (codeL, stationDepart, stationArrivee)
VALUES ('B002', 'Alger Centre', 'Dar El Beida');

4. MISE À JOUR DES RÉFÉRENCES ENTRE LES MOYENS DE TRANSPORT ET LES LIGNES
-- Associer les lignes de bus au moyen de transport BUS
DECLARE
    mt_ref REF tMoyenTransport;
BEGIN
    SELECT REF(mt) INTO mt_ref FROM MoyenTransport mt WHERE mt.abreviation =
    'BUS';

    UPDATE Ligne l SET l.MoyenTransportL = mt_ref WHERE l.codeL LIKE 'B%';
END;
/

- 5. INSERTION DES TRONÇONS
-- Tronçons pour la ligne de bus B001
INSERT INTO Troncon (numeroT, stationDebut, stationFin, longueur)
VALUES (1, 'Alger Centre', 'Hussein Dey', 5.3);

INSERT INTO Troncon (numeroT, stationDebut, stationFin, longueur)
VALUES (2, 'Hussein Dey', 'El Harrach', 4.2);

6. INSERTION DES NAVETTES
-- Navettes de Bus
INSERT INTO Navette (numeroN, marque, anneeMiseEnCirculation)
VALUES (1, 'Hyundai Algérie', 2022);

INSERT INTO Navette (numeroN, marque, anneeMiseEnCirculation)
VALUES (2, 'Hyundai Algérie', 2023);

INSERT INTO Navette (numeroN, marque, anneeMiseEnCirculation)
VALUES (3, 'Isuzu', 2021);

-- INSERTION DES VOYAGES
-- Voyages pour les navettes de bus (Ligne B001, Navettes 1)
DECLARE
    nav_ref REF tNavette;

```

```

BEGIN
  -- Navette 1 (Ligne B001)
  SELECT REF(n) INTO nav_ref FROM Navette n WHERE n.numeroN = 1;

  -- Voyages aller (Alger Centre -> El Harrach)
  INSERT INTO Voyage (numeroV, dateVoyage, heureDebut, duree, sens,
nombreVoyageurs, observation, Navette)
  VALUES (1, TO_DATE('05-01-2025', 'DD-MM-YYYY'), '07:00', '00:45',
'Aller', 42, 'Normal', nav_ref);

  INSERT INTO Voyage (numeroV, dateVoyage, heureDebut, duree, sens,
nombreVoyageurs, observation, Navette)
  VALUES (2, TO_DATE('05-01-2025', 'DD-MM-YYYY'), '09:30', '00:50',
'Aller', 38, 'Normal', nav_ref);

  INSERT INTO Voyage (numeroV, dateVoyage, heureDebut, duree, sens,
nombreVoyageurs, observation, Navette)
  VALUES (3, TO_DATE('06-01-2025', 'DD-MM-YYYY'), '07:00', '00:45',
'Aller', 45, 'Normal', nav_ref);

```

De cette manière en créer les autres

E- Langage d'interrogation de données :

10. Lister tous les voyages (num, date, moyen de transport, navette) ayant enregistré un quelconque problème (panne, retard, accident, ...) :

Démarche :

- Sélectionner les informations des voyages où un problème a été signalé (observation IS NOT NULL et observation != 'RAS').
- Associer chaque voyage à son moyen de transport via la navette utilisée.
- Extraire les informations de la navette (numeroN) et du moyen de transport (abreviation).

Fonctions et techniques utilisées :

- Deref : Permet d'accéder aux attributs d'un objet référencé (ici, Navette).
- Ref : Crée une référence vers un objet (MoyenTransport).
- Where : Filtre les voyages ayant des observations différentes de 'RAS'.

*La requête et son exécution sont montrées dans la capture suivante :

PL/SQL procedure successfully completed.

```
SQL> SELECT v.numeroV AS num_voyage,  
2         v.dateVoyage AS date_voyage,  
3         mt.abreviation AS moyen_transport,  
4         Deref(v.Navette).numeroN AS num_navette,  
5         v.observation AS probleme  
6 FROM Voyage v,  
7         MoyenTransport mt  
8 WHERE v.observation IS NOT NULL  
9 AND v.observation != 'RAS'  
10 AND Deref(v.Navette).MoyenTransportN = Ref(mt);
```

NUM_VOYAGE	DATE_VOYA	MOYEN_TRAN	NUM_NAVETTE	PROBLEME
------------	-----------	------------	-------------	----------

3	06-JAN-25	BUS	1	Panne
4	05-JAN-25	BUS	1	Panne
5	05-JAN-25	BUS	1	Panne
8	06-JAN-25	BUS	2	Retard
12	11-JAN-25	BUS	3	Retard
18	16-JAN-25	BUS	5	Retard
19	15-JAN-25	BUS	5	Panne
20	16-JAN-25	BUS	5	Panne
21	20-JAN-25	MET	6	Retard
22	20-JAN-25	MET	6	Retard
23	20-JAN-25	MET	6	Retard

NUM_VOYAGE	DATE_VOYA	MOYEN_TRAN	NUM_NAVETTE	PROBLEME
------------	-----------	------------	-------------	----------

32	25-JAN-25	MET	8	Retard
41	02-FEB-25	TRA	10	Retard
43	05-FEB-25	TRA	11	Retard
44	05-FEB-25	TRA	11	Retard
45	05-FEB-25	TRA	11	Retard
63	19-FEB-25	BUS	16	Retard

17 rows selected.

11. Lister toutes les lignes (numéro, début et fin) comportant une station principale :

```
SELECT DISTINCT  
1.codeL AS "Numéro Ligne",  
1.stationDepart AS "Station Départ",  
1.stationArrivee AS "Station Arrivée"  
FROM  
1 Ligne 1  
WHERE  
1.stationDepart IN (SELECT nom FROM Station WHERE type = 'principale')  
OR 1.stationArrivee IN (SELECT nom FROM Station WHERE type = 'principale')  
ORDER BY 1.codeL
```

Démarche :

- Sélectionner les lignes dont la station de départ ou d'arrivée est une station principale (type = 'principale').
- Utiliser une sous-requête pour identifier les stations principales.

- Supprimer les doublons grâce à DISTINCT.
- Trier les lignes par numéro (codeL).

Fonctions et techniques utilisées :

- IN : Permet de vérifier si une valeur existe dans le résultat d'une sous-requête.
- DISTINCT : pour éliminer les doublons.
- ORDER BY : pour trier les résultats par codeL

Résultat:

```

-----
TR001
Hussein Dey
Bordj El Kiffan

TR002
Alger Centre
BEZ

Numéro Lig
-----
Station Départ
-----
-----
Station Arrivée
-----
-----

TR003
Mohammadia
Dar El Beida

TR004
Hussein Dey

Numéro Lig
-----
Station Départ
-----
-----
Station Arrivée
-----
-----

Oued Smar

18 rows selected.

```

12. Quelles sont les navettes (numéro, type de transport, année de mise en service) ayant effectué le maximum de voyages durant le mois de janvier 2025 ? Préciser le nombre de voyages :


```

SQL> WITH StatsVoyages AS (
2     SELECT
3         Deref(v.Navette).numeroN AS num_navette,
4         Deref(Deref(v.Navette).MoyenTransportN).abreviation AS type_tra
nsport,
5         Deref(v.Navette).anneeMiseEnCirculation AS annee_mise_service,
6         COUNT(*) AS nb_voyages
7     FROM Voyage v
8     WHERE v.dateVoyage BETWEEN TO_DATE('01/01/2025', 'DD/MM/YYYY')
9           AND TO_DATE('31/01/2025', 'DD/MM/YYYY')
10    GROUP BY
11        Deref(v.Navette).numeroN,
12        Deref(Deref(v.Navette).MoyenTransportN).abreviation,
13        Deref(v.Navette).anneeMiseEnCirculation
14 )
15 SELECT num_navette, type_transport, annee_mise_service, nb_voyages
16 FROM StatsVoyages
17 WHERE nb_voyages = (SELECT MAX(nb_voyages) FROM StatsVoyages);

```

NUM_NAVETTE	TYPE_TRANS	ANNEE_MISE_SERVICE	NB_VOYAGES
6	MET	2020	6
1	BUS	2022	6

La requête fait ceci :

- Crée une vue temporaire appelée StatsVoyages (avec WITH).
- Parcourt la table Voyage pour la période du 1er au 31 janvier 2025.
- Récupère pour chaque navette :
 - Son numéro (numeroN),
 - Le type de transport associé (abrev. Du MoyenTransport),
 - Son année de mise en circulation.
- Compte le nombre total de voyages effectués par chaque navette pendant la période (avec COUNT (*)).
- Regroupe les résultats par navette, type de transport et année de mise en service (GROUP BY).
- Ensuite :
 - Sélectionne les navettes ayant effectué le plus de voyages en comparant à la valeur maximale de nb_voyages.

13. Quelles sont les stations offrant au moins 2 moyens de transport ? (Préciser la station et les moyens de transport offerts) :

```

SELECT
    s.codes AS code_station,
    s.nom AS nom_station,
    (SELECT LISTAGG(mt.abreviation, ', ' ) WITHIN GROUP (ORDER BY mt.abreviation)
     FROM MoyenTransport mt
     WHERE REF(mt) IN (SELECT COLUMN_VALUE FROM TABLE(s.moyens))) AS moyens_transport,
    CARDINALITY(s.moyens) AS nombre_moyens
FROM Station s
WHERE CARDINALITY(s.moyens) >= 2
ORDER BY nombre_moyens DESC;

```

Démarche :

- Sélectionner les stations ayant au moins deux moyens de transport (CARDINALITY(s.moyens) >= 2).
- Lister les moyens de transport desservant chaque station en utilisant LISTAGG.
- Compter le nombre de moyens de transport (CARDINALITY).
- Trier les stations par ordre décroissant de moyens disponibles (ORDER BY nombre_moyens DESC).

Fonctions et techniques utilisées :

- CARDINALITY : Compte le nombre d'éléments dans le tableau s.moyens.
- ORDER BY : Trie les résultats par nombre_moyens.
- LISTAGG : Dans cette requête, LISTAGG est utilisée pour créer une liste de moyens de transport desservant chaque station sous forme d'une chaîne de caractères séparée par des virgules. Par exemple, "BUS, MET, TRA".
- Le WITHIN GROUP (ORDER BY mt.abreviation) permet de spécifier l'ordre dans lequel ces moyens sont listés. Ici, ils sont triés par ordre alphabétique des abréviations (mt.abreviation).

Résultat:

```
CODE_STATI
-----
NOM_STATION
-----
MOYENS_TRANSPORT
-----
NOMBRE_MOYENS
-----
S015
Ha |» El Badr
BUS, MET, TRA, TRN
      4

CODE_STATI
-----
NOM_STATION
-----
MOYENS_TRANSPORT
-----
NOMBRE_MOYENS
-----
S014
Birtouta
BUS, MET, TRA, TRN
      4
```

Partie II : NoSQL – Modèle orienté « documents »

A- Modélisation orientée document

Voici une modélisation orientée document de la base de données décrite dans la partie I :

```
{
  "_id": ObjectId("..."),
  "numeroV": 1,
  "dateVoyage": ISODate("2025-01-05T00:00:00Z"),
  "heureDebut": "07:00",
  "duree": "00:45",
  "sens": "Aller",
  "nombreVoyageurs": 42,
  "observation": "Normal",
  "navette": {
    "numeroN": 1,
    "marque": "Hyundai Algérie",
    "anneeMiseEnCirculation": 2022,
    "ligne": {
      "codeL": "B001",
      "stationDepart": "Alger Centre",
      "stationArrivee": "El Harrach",
      "moyenTransport": {
        "abreviation": "BUS",
        "heureOuverture": "05:00",
        "heureFermeture": "23:00",
        "NbMoyenVoyageurs": 50
      },
    },
  },
  "troncons": [
    {
      "numeroT": 1,
      "stationDebut": "Alger Centre",
      "stationFin": "Hussein Dey",
      "longueur": 5.3
    },
    {
      "numeroT": 2,
      "stationDebut": "Hussein Dey",
      "stationFin": "El Harrach",
      "longueur": 4.2
    }
  ]
},
"stations": [
  {
```

```

        "codeS": "S001",
        "nom": "Alger Centre",
        "latitude": 36.7539,
        "longitude": 3.0589,
        "type": "principale"
    },
    {
        "codeS": "S004",
        "nom": "Hussein Dey",
        "latitude": 36.7400,
        "longitude": 3.1078,
        "type": "principale"
    },
    {
        "codeS": "S002",
        "nom": "El Harrach",
        "latitude": 36.7167,
        "longitude": 3.1500,
        "type": "principale"
    }
]
}

```

Voici un exemple de la BD qu'on a générée :

```

[
  {
    "numeroV": 1,
    "dateVoyage": ISODate("2025-01-05T00:00:00Z"),
    "heureDebut": "07:00",
    "duree": "00:45",
    "sens": "Aller",
    "nombreVoyageurs": 42,
    "observation": "Normal",
    "navette": {
      "numeroN": 1,
      "marque": "Hyundai Algérie",
      "anneeMiseEnCirculation": 2022,
      "ligne": {
        "codeL": "B001",
        "stationDepart": "Alger Centre",
        "stationArrivee": "El Harrach",
        "moyenTransport": {
          "abreviation": "BUS",
          "heureOuverture": "05:00",
          "heureFermeture": "23:00",
          "NbMoyenVoyageurs": 50
        }
      },
      "troncons": [
        {

```

```

        "numeroT": 1,
        "stationDebut": "Alger Centre",
        "stationFin": "Hussein Dey",
        "longueur": 5.3
    },
    {
        "numeroT": 2,
        "stationDebut": "Hussein Dey",
        "stationFin": "El Harrach",
        "longueur": 4.2
    }
]
}
},
"stations": [
    {
        "codeS": "S001",
        "nom": "Alger Centre",
        "latitude": 36.7539,
        "longitude": 3.0589,
        "type": "principale"
    },
    {
        "codeS": "S004",
        "nom": "Hussein Dey",
        "latitude": 36.7400,
        "longitude": 3.1078,
        "type": "principale"
    },
    {
        "codeS": "S002",
        "nom": "El Harrach",
        "latitude": 36.7167,
        "longitude": 3.1500,
        "type": "principale"
    }
]
},
{
    "numeroV": 2,
    "dateVoyage": ISODate("2025-01-06T00:00:00Z"),
    "heureDebut": "08:30",
    "duree": "00:35",
    "sens": "Retour",
    "nombreVoyageurs": 30,
    "observation": "Retard de 5 minutes",
    "navette": {
        "numeroN": 2,

```

```
"marque": "Mercedes",
"anneeMiseEnCirculation": 2023,
"ligne": {
  "codeL": "B002",
  "stationDepart": "Bordj El Kiffan",
  "stationArrivee": "Bab Ezzouar",
  "moyenTransport": {
    "abreviation": "TRA",
    "heureOuverture": "06:00",
    "heureFermeture": "22:00",
    "NbMoyenVoyageurs": 80
  },
  "troncons": [
    {
      "numeroT": 1,
      "stationDebut": "Bordj El Kiffan",
      "stationFin": "Cité 5 Juillet",
      "longueur": 3.0
    },
    {
      "numeroT": 2,
      "stationDebut": "Cité 5 Juillet",
      "stationFin": "Bab Ezzouar",
      "longueur": 2.8
    }
  ]
},
"stations": [
  {
    "codeS": "S005",
    "nom": "Bordj El Kiffan",
    "latitude": 36.7600,
    "longitude": 3.1900,
    "type": "principale"
  },
  {
    "codeS": "S006",
    "nom": "Cité 5 Juillet",
    "latitude": 36.7500,
    "longitude": 3.1800,
    "type": "secondaire"
  },
  {
    "codeS": "S007",
    "nom": "Bab Ezzouar",
    "latitude": 36.7250,
    "longitude": 3.1800,
```

```
        "type": "principale"
    }
]
}
```

Justification de la Conception :

Données centralisées :

- Toutes les informations liées à un voyage sont regroupées dans un seul bloc de données. Cela permet d'éviter des lectures multiples et des croisements complexes avec d'autres ensembles de données.

Lecture rapide :

- Le système est conçu pour privilégier la vitesse de lecture, quitte à occuper plus d'espace.
- Idéal pour les cas où la performance est plus critique que l'optimisation du stockage.

Données embarquées intelligemment :

- Les tronçons associés à un voyage sont directement inclus, ce qui accélère leur récupération.
- Les éléments plus volumineux ou partagés, comme les stations, sont référencés, afin d'éviter les redondances excessives.

Adaptabilité :

- Possibilité d'ajouter facilement des informations spécifiques à certains voyages.
- La structure évolue naturellement avec les besoins du projet, sans nécessiter de migration complexe.

Inconvénients de l'approche centrée sur les voyages :

Redondance des données :

- Certaines informations comme celles des navettes, lignes ou moyens de transport sont copiées dans plusieurs blocs. Cela augmente la taille globale de la base, surtout si ces entités sont souvent réutilisées.

Complexité des mises à jour :

- Lorsqu'une entité partagée (ex. : une navette) change, tous les blocs qui la contiennent doivent être mis à jour.
- Les opérations de mise à jour deviennent plus coûteuses, plus longues et plus sujettes aux erreurs.

Limites pour les analyses globales :

- Les requêtes d'agrégation sur des éléments intégrés peuvent être moins efficaces.
- Difficile d'interroger rapidement des données transversales (par ex. : tous les voyages faits par un même moyen de transport).

B- Remplir la base de données

```
db.createCollection("moyensTransport");
const moyensTransport = [
  {
    abbreviation: "BUS",
    heureOuverture: "05:00",
    heureFermeture: "23:00",
    NbMoyenVoyageurs: 50
  },
  {
    abbreviation: "TRAM",
    heureOuverture: "06:00",
    heureFermeture: "22:00",
    NbMoyenVoyageurs: 180
  },
  {
    abbreviation: "METRO",
    heureOuverture: "05:30",
    heureFermeture: "23:30",
    NbMoyenVoyageurs: 300
  },
  {
    abbreviation: "TRAIN",
    heureOuverture: "04:30",
    heureFermeture: "22:30",
    NbMoyenVoyageurs: 400
  }
];
db.moyensTransport.insertMany(moyensTransport);
```

De cette manière on insère les autres.

Voici résultat :

```
C:\WINDOWS\system32\cmd. X + v
Microsoft Windows [version 10.0.22631.5189]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\ADMIN>mongosh --file "C:\Users\ADMIN\Desktop\S2\TPs\Bda\Projet\generate_transport_data.js"
Base de données générée avec succès:
- 4 moyens de transport
- 15 lignes
- 30 navettes
- 25 stations
- 5000 voyages

C:\Users\ADMIN>
```


C- Répondre aux requêtes suivantes

- ❖ Afficher tous les voyages effectués en date du 01-01-2025 (préciser les détails de chaque Voyage) :

```
mongosh mongodbr://127.0.0.0. x + v
smartcity> db.voyages.find({
...   dateVoyage: {
...     $gte: ISODate("2025-01-01T00:00:00Z"),
...     $lt: ISODate("2025-01-02T00:00:00Z")
...   }
... }).pretty();
...
[
  {
    _id: ObjectId('681694b0c72fcf7bcab60445'),
    numeroV: 2915,
    dateVoyage: ISODate('2025-01-01T11:50:02.129Z'),
    heureDebut: '11:31',
    duree: '00:46',
    sens: 'Aller',
    nombreVoyageurs: 35,
    observation: 'Retard',
    navette: {
      _id: ObjectId('6816948dc72fcf7bcab5f8db'),
      numeroN: 23,
      marque: 'Siemens',
      anneeMiseEnCirculation: 2017,
      ligne: {
        _id: ObjectId('6816948dc72fcf7bcab5f8ba'),
        codeL: 'BUS005',
        stationDepart: 'Cheraga',
        stationArrivee: 'Hussein Dey',
        moyenTransport: {
          _id: ObjectId('6816948dc72fcf7bcab5f899'),
          abreviation: 'BUS',
          heureOuverture: '05:00',
          heureFermeture: '23:00',
          NbMoyenVoyageurs: 50
        },
        troncons: [
          {
            numeroT: 1,
            stationDebut: 'Cheraga',
            stationFin: 'Staoueli',
            longueur: 3.6
          }
        ]
      }
    }
  }
]
```

- ❖ Dans une collection BON-Voyage, récupérer tous les voyages (numéro, numLigne, date, heure, sens) n'ayant enregistré aucun problème, préciser le moyen de transport, l'enumérode la navette associés au voyage :

```
db.voyages.aggregate([
  // Filtrer les voyages sans problème (observation "Normal")
  { $match: { observation: "RAS" } },

  // Projeter uniquement les champs nécessaires
  { $project: {
    numeroV: 1,
    numLigne: "$navette.ligne.codeL",
    dateVoyage: 1,
    heureDebut: 1,
    sens: 1,
```

```

    moyenTransport: "$navette.ligne.moyenTransport.abreviation",
    numeroNavette: "$navette.numeroN",
    _id: 0
  }},
  // Insérer les résultats dans une nouvelle collection
  { $out: "bonVoyage" }
]);

```

Voici résultat :

```

smartcity> db.bonVoyage.find().pretty();
[
  {
    _id: ObjectId('681699094f9d40d4074edca0'),
    numeroV: 2,
    dateVoyage: ISODate('2025-08-19T01:47:18.255Z'),
    heureDebut: '17:00',
    sens: 'Retour',
    numLigne: 'TRA002',
    moyenTransport: 'TRAM',
    numeroNavette: 7
  },
  {
    _id: ObjectId('681699094f9d40d4074edca1'),
    numeroV: 4,
    dateVoyage: ISODate('2025-01-10T01:00:35.114Z'),
    heureDebut: '13:23',
    sens: 'Retour',
    numLigne: 'TRA010',
    moyenTransport: 'TRAM',
    numeroNavette: 18
  },
  {
    _id: ObjectId('681699094f9d40d4074edca2'),
    numeroV: 6,
    dateVoyage: ISODate('2024-11-02T08:54:17.798Z'),
    heureDebut: '20:33',
    sens: 'Aller',
    numLigne: 'TRN012',
    moyenTransport: 'TRAIN',
    numeroNavette: 26
  },
  {
    _id: ObjectId('681699094f9d40d4074edca3'),
    numeroV: 7,
    dateVoyage: ISODate('2025-05-11T12:18:45.668Z'),
    heureDebut: '15:24',
    sens: 'Retour',
    numLigne: 'TRN008',

```

- ❖ Récupérer dans une nouvelle collection Ligne-Voyages, les numéros de lignes et le nombre total de voyages effectués (par ligne). La collection devra être ordonnée par ordre décroissant du nombre de voyages. Afficher le contenu de la collection :

```

smartcity> db.voyages.aggregate([
...   // Grouper par numéro de ligne
...   { $group: {
...     _id: "$navette.ligne.codeL",
...     totalVoyages: { $sum: 1 }
...   }},
...   // Renommer le champ _id en numeroLigne
...   { $project: {
...     numeroLigne: "$_id",
...     totalVoyages: 1,
...     _id: 0
...   }},
...   // Trier par nombre de voyages décroissant
...   { $sort: { totalVoyages: -1 } },
...   // Insérer les résultats dans une nouvelle collection
...   { $out: "ligneVoyages" }
... ]);

```

```

smartcity> db.ligneVoyages.find().pretty();
[
  {
    _id: ObjectId('68169c7a4f9d40d4074ee492'),
    totalVoyages: 702,
    numeroLigne: 'MET011'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee493'),
    totalVoyages: 644,
    numeroLigne: 'TRN008'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee494'),
    totalVoyages: 644,
    numeroLigne: 'TRN012'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee495'),
    totalVoyages: 346,
    numeroLigne: 'BUS009'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee496'),
    totalVoyages: 346,
    numeroLigne: 'BUS013'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee497'),
    totalVoyages: 336,
    numeroLigne: 'BUS001'
  },
  {
    _id: ObjectId('68169c7a4f9d40d4074ee498'),
    totalVoyages: 333,
    numeroLigne: 'TRN004'
  },
]

```

- ❖ Augmenter de 100, le nombre de voyageurs sur tous les voyages effectués par métro avant la date du 15 janvier 2025 :

```
smartcity> db.voyages.updateMany(
...   {
...     "navette.ligne.moyenTransport.abreviation": "MET",
...     "dateVoyage": { $lt: ISODate("2025-01-15T00:00:00Z") }
...   },
...   {
...     $inc: { "nombreVoyageurs": 100 }
...   }
... );
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 797,
  modifiedCount: 797,
  upsertedCount: 0
}
smartcity> |
```

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

```
smartcity> // 1. Fonction Map corrigée avec vérification des champs
... var mapFunction = function() {
...   if (this.navette && this.navette.ligne && this.navette.ligne.codeL)
...   {
...     emit(this.navette.ligne.codeL, 1);
...   }
... };
...
... // 2. Fonction Reduce (inchangée)
... var reduceFunction = function(codeLigne, compteurs) {
...   return Array.sum(compteurs);
... };
...
... // 3. Exécution avec gestion des erreurs
... try {
...   db.voyages.mapReduce(
...     mapFunction,
...     reduceFunction,
...     {
...       out: "ligneVoyagesMapReduce",
...       verbose: true // Affiche des détails d'exécution
...     }
...   );
...   print("MapReduce réussi. Résultats dans la collection 'ligneVoyagesM
apReduce'");
... } catch (e) {
...   print("Erreur lors du MapReduce:", e);
... }
...
... // 4. Vérification des résultats
... db.ligneVoyagesMapReduce.find().sort({ value: -1 }).limit(5);
MapReduce réussi. Résultats dans la collection 'ligneVoyagesMapReduce'
[
  { _id: 'MET011', value: 702 },
  { _id: 'TRN012', value: 644 },
  { _id: 'TRN008', value: 644 },
  { _id: 'BUS009', value: 346 },
  { _id: 'BUS013', value: 346 }
]
smartcity>
```

1. Fonction Map

- Parcourt chaque document de la collection voyages.
- Vérifie que le champ navette.ligne.codeL existe.
- Émet une paire :
 - **Clé** : le code de la ligne (codeL)
 - **Valeur** : 1 (unité de comptage)

2. Fonction Reduce

- Reçoit toutes les valeurs associées à chaque codeLigne.
- Additionne tous les 1 pour obtenir le nombre total de voyages par ligne.

3. Exécution du MapReduce

- Appelle mapReduce sur la collection voyages.
- Utilise :
 - La fonction mapFunction
 - La fonction reduceFunction
- Stocke le résultat dans la collection ligneVoyagesMapReduce
- Active verbose: true pour avoir un retour détaillé.
- Gère les erreurs avec try...catch.

Avec notre conception :

La requête a est-elle possible ? Oui, totalement réalisable.

Justification : Les données de navette sont disponibles dans chaque document de voyage via navette. Chaque navette est identifiée par son numeroN ,moyen de transport est accessible via navette.ligne.moyenTransport.abreviation

Approche : Compter les voyages par navette à l'aide d'une agrégation,déterminer le nombre maximum de voyages et filtrer pour ne garder que les navettes atteignant ce maximum

```
//a
db.voyages.aggregate([
  {
    $group: {
      _id: {
        navette: "$navette.numeroN",
        moyenTransport: "$navette.ligne.moyenTransport.abreviation"
      },
      totalVoyages: { $sum: 1 }
    }
  },
  {
    $sort: { totalVoyages: -1 }
  },
  {
```

```

    $group: {
      _id: null,
      maxVoyages: { $first: "$totalVoyages" },
      navettes: {
        $push: {
          navette: "$_id.navette",
          moyenTransport: "$_id.moyenTransport",
          totalVoyages: "$totalVoyages"
        }
      }
    },
    {
      $project: {
        navettes: {
          $filter: {
            input: "$navettes",
            as: "n",
            cond: { $eq: ["$$n.totalVoyages", "$maxVoyages"] }
          }
        }
      }
    },
    {
      $unwind: "$navettes"
    },
    {
      $replaceRoot: { newRoot: "$navettes" }
    }
  ]
})

```

Résultat :

```

...
[ { navette: 17, moyenTransport: 'MET', totalVoyages: 189 } ]
smartcity>

```

La requête b est-elle possible ? Oui, mais plus complexe.

Justification :

- Les données de voyageurs sont disponibles dans chaque document via nombreVoyageurs
- La date est disponible via dateVoyage
- Le moyen de transport est accessible via navette.ligne.moyenTransport.abreviation
- Pour le critère "toujours", nous devons analyser chaque jour séparément

Approche :

- Regrouper les données par jour et par moyen de transport
- Calculer le total journalier de voyageurs pour chaque moyen
- Identifier les moyens qui sont toujours au-dessus du seuil

Défis potentiels :

Vérifier que la condition est vraie pour tous les jours est plus complexe donc on a décidé de modifier cette requête par filtrer uniquement les jours où le seuil est dépassé, sachant que le seuil est fixé à 1000 car dans les données insérées dans notre base de données le nombre de voyageurs total pour chaque moyen de transport ne dépasse pas 2000. Et aussi la requête ne retourne rien quand elle vérifie que chaque jour le moyen de transport doit dépasser le 1000 à cause des données insérées où aucun moyen ne satisfait la contrainte donc nous avons opté à chercher les moyens de transport qui ont dépassé au moins un fois le seuil (le même moyen peut dépasser les 1000 voyageurs par jour en plusieurs reprises mais pas tout les jours) .

```
// b
db.voyages.aggregate([
  // Regrouper par date + moyen de transport (ID + abréviation)
  {
    $group: {
      _id: {
        date: { $dateToString: { format: "%Y-%m-%d", date: "$dateVoyage" }
      },
      moyenId: "$navette.ligne.moyenTransport._id",
      moyenAbrev: "$navette.ligne.moyenTransport.abreviation"
    },
    totalVoyageurs: { $sum: "$nombreVoyageurs" }
  },
  // Filtrer uniquement les jours où le seuil est dépassé (ex: 1000)
  { $match: { totalVoyageurs: { $gt: 1000 } } },
  { $sort: { "_id.date": 1 } },
  // le résultat
  {
    $project: {
      _id: 0,
      date: "$_id.date",
      moyenId: "$_id.moyenId",
      moyen: "$_id.moyenAbrev",
      totalVoyageurs: 1
    }
  }
])
```

Et voici le résultat d'exécution de cette requête :


```

smartcity> db.voyages.aggregate([
...   // Étape 1: Regrouper par date + moyen de transport (ID + abréviation)
...   {
...     $group: {
...       _id: {
...         date: { $dateToString: { format: "%Y-%m-%d", date: "$dateVoyage"
...       } },
...       moyenId: "$navette.ligne.moyenTransport._id",
...       moyenAbrev: "$navette.ligne.moyenTransport.abreviation"
...     },
...     totalVoyageurs: { $sum: "$nombreVoyageurs" }
...   },
...   // Étape 2: Filtrer uniquement les jours où le seuil est dépassé (ex:
1000)
...   { $match: { totalVoyageurs: { $gt: 1000 } } },
...   // Étape 3: Trier par date (optionnel)
...   { $sort: { "_id.date": 1 } },
...   // Étape 4: Formater le résultat
...   {
...     $project: {
...       _id: 0,
...       date: "$_id.date",
...       moyenId: "$_id.moyenId",
...       moyen: "$_id.moyenAbrev",
...       totalVoyageurs: 1
...     }
...   }
... ])
[
  {
    totalVoyageurs: 1306,
    date: '2024-01-19',
    moyenId: ObjectId('6816948dc72fcf7bcab5f89c'),
    moyen: 'TRN'
  },
  {
    totalVoyageurs: 1051,
    date: '2024-01-25',

```

D- Analyse

Analyse des requêtes :

- **Requête 1** : Simple recherche de voyages par date, utilisation standard.
- **Requête 2** : Agrégation pour filtrer les voyages normaux.
- **Requête 3** : Agrégation pour compter les voyages par ligne.
- **Requête 4** : Mise à jour de tous les métros (MET) pour augmenter le nombre de voyageurs.
- **Requête 5** : Utilisation de mapReduce pour compter les voyages par code de ligne.
- **Requête a** : Agrégation complexe pour trouver les navettes avec le plus grand nombre de voyages.
- **Requête b** : Script complexe pour identifier les moyens de transport dont le nombre de voyageurs dépasse un seuil

Problèmes avec la conception actuelle :

- Documents trop imbriqués : La structure actuelle crée des documents complexes avec beaucoup de niveaux d'imbrication, rendant les requêtes plus verbales (\$navette.ligne.moyenTransport.abreviation).
- Redondance des données : Les informations sur les stations, les lignes, et les moyens de transport sont répétées dans chaque document de voyage.
- Difficultés pour les mises à jour : Si une information sur une ligne ou une station change, il faudrait mettre à jour tous les documents de voyage concernés.
- Requêtes complexes : Les requêtes d'agrégation deviennent complexes et moins performantes à cause de la structure imbriquée.

Modèle de données amélioré pour le système de transport :

```
1. Collection moyenTransport
{
  "_id": ObjectId("..."),
  "abreviation": "BUS",
  "nom": "Bus urbain",
  "heureOuverture": "05:00",
  "heureFermeture": "23:00",
  "NbMoyenVoyageurs": 50
}

2. Collection station
{
  "_id": ObjectId("..."),
  "codeS": "S001",
  "nom": "Alger Centre",
  "latitude": 36.7539,
  "longitude": 3.0589,
  "type": "principale"
}

3. Collection troncon
{
  "_id": ObjectId("..."),
  "numeroT": 1,
  "stationDebut": "S001", // Référence au codeS de la station
  "stationFin": "S004",   // Référence au codeS de la station
  "longueur": 5.3
}

4. Collection ligne
{
  "_id": ObjectId("..."),
  "codeL": "B001",
  "stationDepart": "S001", // Référence au codeS de la station
```

```

    "stationArrivee": "S002", // Référence au codeS de la station
    "moyenTransportId": "BUS", // Référence à l'abréviation du moyen de
transport
    "troncons": ["T001", "T002"] // Références aux IDs des tronçons
}

5. Collection navette
{
    "_id": ObjectId("..."),
    "numeroN": 1,
    "marque": "Hyundai Algérie",
    "anneeMiseEnCirculation": 2022,
    "ligneId": "B001" // Référence au codeL de la ligne
}

6. Collection voyage
{
    "_id": ObjectId("..."),
    "numeroV": 1,
    "dateVoyage": ISODate("2025-01-05T00:00:00Z"),
    "heureDebut": "07:00",
    "duree": "00:45",
    "sens": "Aller",
    "nombreVoyageurs": 42,
    "observation": "Normal",
    "navetteId": 1, // Référence au numeroN de la navette
    "stationsDesservies": ["S001", "S004", "S002"] // Références aux codeS
des stations
}

```

Cette approche offre plusieurs avantages :

- Structure plus claire : Chaque entité (voyage, navette, ligne, station, etc.) a sa propre collection.
- Élimination de la redondance : Les informations sont stockées une seule fois et référencées par ID.
- Facilité de maintenance : Les mises à jour se font à un seul endroit.
- Meilleure indexation : Possibilité d'indexer efficacement chaque collection.

Conclusion

Ce projet nous a permis de mettre en œuvre deux approches complémentaires de gestion des données : relationnel-objet avec Oracle SQL3 et orienté documents avec MongoDB. À travers les différentes requêtes et manipulations, nous avons pu répondre efficacement aux besoins de modélisation et d'analyse liés à la gestion d'un réseau de transport urbain.

L'ensemble du travail a renforcé notre compréhension des modèles avancés de bases de données, tout en illustrant les enjeux de performance, de conception adaptée et de structuration des données selon les cas d'usage. Ce projet constitue ainsi une expérience complète et formatrice dans le contexte des systèmes d'information intelligents