



L'Université des Sciences et Technologies de Houari-Boumediène - Faculté
d'Informatique -
Département IA & SD

Rapport de Projet de TP Base de Données Avancées

Projet de TP

SQL3-Oracle et NoSQL (MongoDB)

Réalisé par :
THAROUMA Ryma
Et
ANANE Dalia Khadidja

Responsable du module :
Mme S. Boukhedouma

Enseignant du TP :
Mme Z. Challal

Table des matières

1	Introduction Générale	1
2	Partie I – Relationnel-Objet (SQL3 Oracle)	1
2.1	Modélisation Orientée Objet	1
2.1.1	Classes :	2
2.1.2	Cardinalités :	2
2.2	Transformation en Schéma Relationnel	2
2.3	Création des Tablespaces et de l'Utilisateur	3
2.4	Définition des Types	4
2.5	Ajout de Méthodes Membres aux Types Objets	6
2.6	Exécution des Méthodes Objets	9
2.7	Création des Tables	11
2.8	Insertion des Données	11
2.9	Mise à jour des collections imbriquées par références	12
2.10	Requêtes SQL	13
3	Partie II – NoSQL Orienté Documents (MongoDB)	17
3.1	Modélisation Orientée Document	17
3.1.1	Introduction	17
3.1.2	Modélisation orientée document	17
3.2	Insertion des Données	19
3.3	Requêtes MongoDB	21
3.4	Analyse	24
3.5	Conclusion	25
4	Ajout d'un Menu et Application Web	25
4.1	Partie SQL3	25
4.2	Partie Mongo DB	26
.1	Diagramme de classes UML	29
.2	Lien vers le projet	29

1 Introduction Générale

Ce projet s'inscrit dans le cadre du module de Bases de Données Avancées et porte sur la gestion d'un réseau de transport urbain dans une ville intelligente. L'objectif principal est de modéliser, implémenter et interroger une base de données complexe en utilisant deux approches complémentaires : le modèle relationnel-objet avec SQL3 Oracle, et le modèle orienté documents avec MongoDB. À travers cette étude de cas, nous avons manipulé des entités telles que les moyens de transport, lignes, stations, navettes et voyages, tout en mettant en œuvre des requêtes avancées et des structures adaptées à chaque paradigme.

2 Partie I – Relationnel-Objet (SQL3 Oracle)

2.1 Modélisation Orientée Objet

— Diagramme de classes UML avec attributs et méthodes.

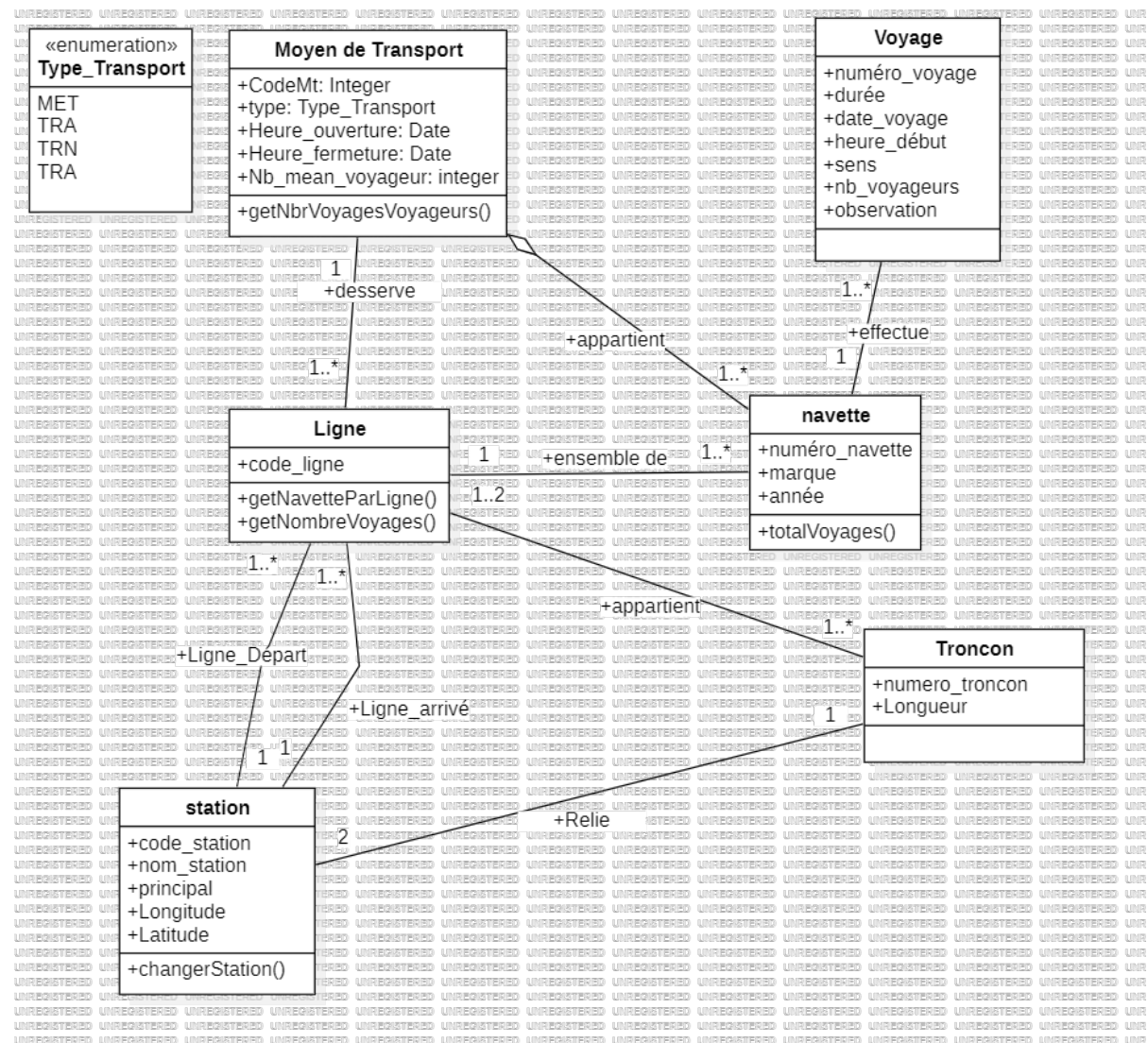


FIGURE 1 – Diagramme de classes UML

2.1.1 Classes :

- **MoyenTransport** :
 - Attributs : `CodeMt`, `type` (*enumération* : {MET, BUS, TRN, TRA}),
`Heure_ouverture`, `Heure_fermeture`, `Nb_mean_voyageur`
 - Méthode : `getNbrVoyagesVoyageurs()`
- **Ligne** :
 - Attributs : `code_ligne`
 - Méthodes : `getNavetteParLigne()`, `getNombreVoyages()`
- **Voyage** :
 - Attributs : `numéro_voyage`, `durée`, `date_voyage`, `heure_début`, `sens`,
`nb_voyageurs`, `observation`
- **Navette** :
 - Attributs : `numéro_navette`, `marque`, `année`
 - Méthode : `totalVoyages()`
- **Troncon** :
 - Attributs : `numéro_troncon`, `longueur`
- **Station** :
 - Attributs : `code_station`, `nom_station`, `principal`, `longitude`, `latitude`
 - Méthode : `changerStation()`

2.1.2 Cardinalités :

Entité 1	Entité 2	Cardinalité Entité 1	Cardinalité Entité 2
MoyenTransport	Ligne	1	1..*
MoyenTransport	Navette	1	1..*
Navette	Voyage	1	1..*
Ligne	Navette	1	1..*
Ligne	Troncon	1..2	1..*
Troncon	Station	1	2
Ligne	Station (départ)	1..*	1
Ligne	Station (arrivée)	1..*	1

TABLE 1 – Tableau des cardinalités entre les entités du système

2.2 Transformation en Schéma Relationnel

Modèle Logique de Données (MLD) :

- MoyenTransport(codeMT, type, heure_ouverture, heure_fermeture, nb_moyen_voyageurs)
- Station(codeStation, nom, principale, longitude, latitude, numTroncon*)
- Ligne(codeLigne, codeMT*, stationDepart*, stationArrivee*)
- Troncon(numTroncon, longueur, Ligne1*, Ligne2*)
- Navette(numNavette, marque, annee, codeMT*)
- Voyage(numVoyage, date_voyage, heureDebut, duree, sens, nbVoyageurs, observation)
- LigneNavette(numNavette*, codeLigne*)

- VoyageNavette(numVoyage*, numNavette*)

2.3 Création des Tablespaces et de l'Utilisateur

On commence par se connecter en tant qu'administrateur Oracle:

```
SQL> CONNECT sys/your_password AS SYSDBA;
Connected.
SQL>
```

FIGURE 2 – Connexion avec les droits SYSDBA

Ensuite, on procède à la création d'un environnement dédié pour un utilisateur nommé SQL3 dans la base de données Oracle:

- Création du tablespace principal : la commande CREATE TABLESPACE SQL3_TBS crée un espace de stockage permanent, avec un fichier qui peut s'étendre automatiquement jusqu'à 500 mégaoctets.

```
SQL> CREATE TABLESPACE SQL3_TBS
2 DATAFILE 'sql3_tbs.dbf'
3 SIZE 100M
4 AUTOEXTEND ON NEXT 10M MAXSIZE 500M;

Tablespace created.
```

FIGURE 3 – Création du tablespace principal SQL3_TBS

- Création du tablespace temporaire : la commande CREATE TEMPORARY TABLESPACE SQL3_TempTBS crée un espace utilisé pour les opérations intermédiaires.

```
SQL> CREATE TEMPORARY TABLESPACE SQL3_TempTBS
2 TEMPFILE 'sql3_temp_tbs.dbf'
3 SIZE 50M
4 AUTOEXTEND ON NEXT 5M MAXSIZE 200M;

Tablespace created.
```

FIGURE 4 – Création du tablespace temporaire SQL3_TempTBS

- Création de l'utilisateur SQL3 : la commande CREATE USER permet de créer l'utilisateur SQL3 avec le mot de passe sql3pass et les tablespaces définis.

```
SQL> CREATE USER SQL3
2 IDENTIFIED BY sql3pass
3 DEFAULT TABLESPACE SQL3_TBS
4 TEMPORARY TABLESPACE SQL3_TempTBS
5 QUOTA UNLIMITED ON SQL3_TBS;

User created.
```

FIGURE 5 – Création de l'utilisateur SQL3

- Attribution des privilèges : plusieurs droits sont accordés à l'utilisateur, tels que :
 - CONNECT, RESOURCE pour créer des objets et se connecter,
 - UNLIMITED TABLESPACE pour lever les limites de quotas,
 - CREATE SESSION pour autoriser les connexions,
 - ALL PRIVILEGES pour tous les droits.

```
SQL> conn
Enter user-name: SQL3
Enter password:
Connected.
SQL>
```

FIGURE 6 – Affectation des privilèges SQL3

- Connexion à la session : l'utilisateur peut maintenant se connecter avec CONNECT SQL3/sql3pass.

```
SQL> GRANT CONNECT, RESOURCE TO SQL3;
Grant succeeded.

SQL> GRANT UNLIMITED TABLESPACE TO SQL3;
Grant succeeded.

SQL> GRANT CREATE SESSION TO SQL3;
Grant succeeded.

SQL> GRANT ALL PRIVILEGES TO SQL3;
Grant succeeded.
```

FIGURE 7 – Connexion en tant que SQL3

2.4 Définition des Types

- Types abstraits: on définit tous les types par des types abstraits , puis en les redéfinie plus tard avec les attributs nécessaire. Les classes sont reliées avec plusieurs liens entres elles ce qui nécessite de les définir d'abord comme type abstrait.

Listing 1 – Déclaration anticipée des types objets

```
CREATE TYPE tMoyenTransport;
/
CREATE TYPE tLigne;
/
CREATE TYPE tStation;
/
CREATE TYPE tNavette;
/
CREATE TYPE tVoyage;
/
CREATE TYPE tTroncon;
/
```

- Associations. Chaque relation sera interprété par une Nested Table si plusieurs lignes sont référencées ou bien un simple attribut est référencée.

Entité A	Entité B	Cardinalité	Références
Ligne	MoyenTransport	1	REF(codeMT)
MoyenTransport	Ligne	1..*	t_set_ref(codeLigne)
Navette	MoyenTransport	1	REF(codeMT)
MoyenTransport	Navette	1..*	t_set_ref(numNavette)
Voyage	Navette	1	REF(numNavette)
Navette	Voyage	1..*	t_set_ref(numVoyage)
Navette	Ligne	1	REF(codeLigne)
Ligne	Navette	1..*	t_set_ref(numNavette)
Troncon	Ligne	1..2	REF(codeLigne) <i>Ligne1</i> , REF(codeLigne) <i>Ligne2</i>
Ligne	Troncon	1..*	t_set_ref(numTroncon)
Troncon	Station	2	REF(codeStation) <i>Station1</i> , REF(codeStation) <i>Station2</i>
Station	Troncon	1	REF(numTroncon)
Ligne	Station (départ)	1	REF(codeStation)
Ligne	Station (arrivée)	1	REF(codeStation)
Station (départ)	Ligne	1..*	REF(codeLigne)
Station (arrivée)	Ligne	1..*	REF(codeLigne)

TABLE 2 – Relations bidirectionnelles avec cardinalités et types de références

Listing 2 – Déclaration des types collections de références

```

Create or replace type t_set_ref_lignes as table of
  ref tLigne;
/
Create or replace type t_set_ref_voyage as table of
  ref tVoyage;
/
Create or replace type t_set_ref_navette as table of
  ref tNavette;
/
Create or replace type t_set_ref_troncon as table of
  ref tTroncon;
/

```

- Exemple de création d'un type objet et utilisation de NOT FINAL
La commande:

```

CREATE OR REPLACE TYPE tNavette AS OBJECT (
  numNavette      INTEGER,
  marque          VARCHAR2(20),
  annee           INTEGER,
  codeMT          REF tMoyenTransport,
  navette_voyage  t_set_ref_voyage,
  navette_ligne   REF tLigne
)

```

```
) NOT FINAL;
```

permet de définir un nouveau type objet Oracle nommé tNavette avec six attributs, dont deux références à d'autres types (tMoyenTransport et tLigne) et une collection de références (t_set_ref_voyage).

Le mot-clé NOT FINAL indique que ce type est extensible : d'autres types pourront hériter de ses attributs et méthodes, ce qui facilite la création d'hiérarchies de types et la réutilisation de la modélisation objet.

2.5 Ajout de Méthodes Membres aux Types Objets

Dans cette section, on enrichit les types objets définis précédemment par des méthodes membres, permettant d'ajouter de l'intelligence métier directement au sein des objets.

1. totalVoyages dans tNavette

- Cette fonction membre retourne le nombre total de voyages réalisés par une navette.
- La clause TABLE(navette_voyage) permet d'accéder à la collection imbriquée contenant les références des voyages.
- Le mot-clé Deref est utilisé pour désigner les objets pointés par les références.
- Le résultat est stocké dans la variable total, puis retourné.

```
CREATE OR REPLACE TYPE BODY tNavette AS
    MEMBER FUNCTION totalVoyages RETURN NUMBER IS
        total NUMBER := 0;
    BEGIN
        SELECT COUNT(*) INTO total
        FROM TABLE(navette_voyage);
        RETURN total;
    END;
END;
```

2. getNavettesParLigne dans tLigne

- Cette fonction retourne un curseur SYS_REFCURSOR qui pourrait contenir la liste des navettes d'une ligne.
- Dans l'exemple, un affichage via DBMS_OUTPUT.PUT_LINE est utilisé pour imprimer les navettes associées.
- La méthode utilise une boucle sur la collection SELF.ligne_navette pour accéder aux références des objets navettes.

```
MEMBER FUNCTION getNavettesParLigne RETURN
    SYS_REFCURSOR IS
    lignes_navettes SYS_REFCURSOR;
    r_navette tNavette;
    r_ref      REF tNavette;
```



```

        temp_tab  SYS_REFCURSOR;
BEGIN
    FOR i IN 1 .. SELF.ligne_navette.COUNT LOOP
        SELECT Deref(SELF.ligne_navette(i)) INTO r_navette
            FROM DUAL;
        DBMS_OUTPUT.PUT_LINE('Ligne: ' || SELF.codeLigne ||
            ', Navette: ' || r_navette.numNavette);
    END LOOP;

    RETURN NULL; -- pas de curseur car on n'a pas de
        table cible
END;

```

3. getNombreVoyages dans tLigne

- Cette fonction calcule le nombre total de voyages effectués par les navettes d'une ligne entre deux dates données.
- La fonction parcourt chaque navette, puis ses voyages, et compte ceux dont la date est dans l'intervalle spécifié.
- Elle utilise deux boucles imbriquées pour accéder aux collections `ligne_navette` et `navette_voyage`.

```

MEMBER FUNCTION getNombreVoyages(p_date_debut DATE,
    p_date_fin DATE) RETURN NUMBER IS
total NUMBER := 0;
r_navette tNavette;
r_voyage tVoyage;
BEGIN
    FOR i IN 1 .. SELF.ligne_navette.COUNT LOOP
        SELECT Deref(SELF.ligne_navette(i)) INTO r_navette FROM
            DUAL;
        IF r_navette.navette_voyage IS NOT NULL THEN
            FOR j IN 1 .. r_navette.navette_voyage.COUNT LOOP
                SELECT Deref(r_navette.navette_voyage(j)) INTO
                    r_voyage FROM DUAL;
                IF r_voyage.date_voyage BETWEEN p_date_debut AND
                    p_date_fin THEN
                    total := total + 1;
                END IF;
            END LOOP;
        END IF;
    END LOOP;
    RETURN total;
END;

```

4. changerNomStation dans tStation

- Cette procédure membre permet de modifier dynamiquement le nom d'une station.
- Si le nom courant est "BEZ", il est remplacé par "Univ".

- Le résultat est affiché avec DBMS_OUTPUT.PUT_LINE.

```

MEMBER PROCEDURE changerNomStation IS
BEGIN
    -- Changer le nom de la station si elle est 'BEZ'
    IF SELF.nom_station = 'BEZ' THEN
        SELF.nom_station := 'Univ';
        DBMS_OUTPUT.PUT_LINE('Station name changed to: ' ||
            SELF.nom_station);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Aucune modification : la
            station n''est pas BEZ.');
```

5. getNbrVoyagesVoyageurs dans tMoyenTransport

- Cette fonction retourne le nombre de voyages effectués pour une date donnée par toutes les navettes d'un moyen de transport.
- Elle parcourt la collection MoyenTransport_Navette, puis pour chaque navette, elle accède aux voyages via la collection navette_voyage.
- Deux variables, total et totalVoyageurs, sont utilisées pour compter respectivement les voyages et les passagers.

```

CREATE OR REPLACE TYPE BODY tMoyenTransport AS
    MEMBER FUNCTION getNbrVoyagesVoyageurs(p_date DATE)
        RETURN NUMBER IS
        total NUMBER := 0;
        totalVoyageurs NUMBER := 0;
        r_navette tNavette;
        r_voyage tVoyage;
    BEGIN
        FOR i IN 1 .. SELF.MoyenTransport_Navette.COUNT LOOP
            SELECT Deref(SELF.MoyenTransport_Navette(i)) INTO
                r_navette FROM DUAL;

            IF r_navette.navette_voyage IS NOT NULL THEN
                FOR j IN 1 .. r_navette.navette_voyage.COUNT LOOP
                    SELECT Deref(r_navette.navette_voyage(j)) INTO
                        r_voyage FROM DUAL;

                    IF r_voyage.date_voyage = p_date THEN
                        total := total + 1;
                        totalVoyageurs := totalVoyageurs + r_voyage.
                            nbVoyageurs;
                    END IF;
                END LOOP;
            END IF;
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('Total voyages: ' || total || ',
            Total voyageurs: ' || totalVoyageurs);
```

```
        RETURN total;  
    END;  
END;
```

Remarques :

- Le mot-clé CASCADE dans ALTER TYPE est requis lorsqu'un type est référencé par d'autres types.
- Deref permet d'accéder aux objets référencés via une référence.
- DBMS_OUTPUT.PUT_LINE est utilisé à des fins de débogage ou de retour d'information console.

2.6 Exécution des Méthodes Objets

Dans cette section, nous démontrons l'appel des méthodes définies dans les types objets Oracle via des blocs PL/SQL. Chaque méthode est invoquée en respectant les étapes suivantes :

1. Obtenir une référence à l'objet à traiter via une requête de type SELECT REF(...).
2. Déréférencer cette référence à l'aide de Deref pour manipuler un objet complet.
3. Appeler la méthode via l'objet instancié (objet.méthode(...)).
4. Afficher le résultat, soit directement via DBMS_OUTPUT.PUT_LINE, soit via une variable.

Les captures suivantes illustrent l'exécution des différentes méthodes :

```
SQL> SET SERVEROUTPUT ON;  
SQL> DECLARE  
2     o_navette tNavette;  
3     total NUMBER;  
4 BEGIN  
5     SELECT Deref(Ref(nav)) INTO o_navette  
6     FROM Navette nav  
7     WHERE nav.numNavette = 1;  
8  
9     total := o_navette.totalVoyages();  
10  
11     DBMS_OUTPUT.PUT_LINE('Nombre total de voyages : ' || total);  
12 END;  
13 /  
Nombre total de voyages : 2  
PL/SQL procedure successfully completed.
```

FIGURE 8 – Exécution de la méthode totalVoyages() de l'objet tNavette

```

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2     r_ligne REF tLigne;
  3     o_ligne tLigne;
  4     dummy SYS_REFCURSOR;
  5 BEGIN
  6     SELECT REF(l) INTO r_ligne FROM Ligne l WHERE l.codeLigne = 'M001';
  7
  8     SELECT Deref(r_ligne) INTO o_ligne FROM DUAL;
  9
 10     dummy := o_ligne.getNavettesParLigne();
 11 END;
 12 /
Ligne: M001, Navette: 1

PL/SQL procedure successfully completed.

```

FIGURE 9 – Exécution de la méthode `getNavetteParLignes()` de l'objet `tLigne`

```

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2     r_ligne REF tLigne;
  3     o_ligne tLigne;
  4     total NUMBER;
  5 BEGIN
  6     SELECT REF(l) INTO r_ligne FROM Ligne l WHERE l.codeLigne = 'M001';
  7
  8     SELECT Deref(r_ligne) INTO o_ligne FROM DUAL;
  9
 10     total := o_ligne.getNombreVoyages(DATE '2025-01-01', DATE '2025-01-31');
 11
 12     DBMS_OUTPUT.PUT_LINE('Nombre de voyages en janvier 2025 : ' || total);
 13 END;
 14 /
Nombre de voyages en janvier 2025 : 2

PL/SQL procedure successfully completed.

```

FIGURE 10 – Exécution de la méthode `getNombreVoyages()` avec intervalle de dates

```

SQL> INSERT INTO Station VALUES (
  2     tStation(99, 'BEZ', 3.200, 36.750, 'TRUE', t_set_ref_lignes(), t_set_ref_lignes(), NULL)
  3 );

1 row created.

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2     r_station REF tStation;
  3     o_station tStation;
  4 BEGIN
  5     SELECT REF(s) INTO r_station FROM Station s WHERE s.nom_station = 'BEZ';
  6
  7     SELECT Deref(r_station) INTO o_station FROM DUAL;
  8
  9     o_station.changerNomStation;
 10 END;
 11 /
Station name changed to: Univ

PL/SQL procedure successfully completed.

```

FIGURE 11 – Exécution de la procédure `changerNomStation()` de l'objet `tStation`

```

SQL> DECLARE
2   r_mt REF tMoyenTransport;
3   o_mt tMoyenTransport;
4   total NUMBER;
5 BEGIN
6   SELECT REF(mt) INTO r_mt FROM MoyenTransport mt WHERE mt.codeMT = 'MET';
7
8   SELECT Deref(r_mt) INTO o_mt FROM DUAL;
9
10  total := o_mt.getNbrVoyagesVoyageurs(DATE '2025-01-01');
11
12  DBMS_OUTPUT.PUT_LINE('Retour de la fonction (total voyages) : ' || total);
13 END;
14 /
Total voyages: 2, Total voyageurs: 210
Retour de la fonction (total voyages) : 2
PL/SQL procedure successfully completed.

```

FIGURE 12 – Exécution de la fonction `getNbrVoyagesVoyageurs()` pour un moyen de transport

2.7 Création des Tables

La table `Navette` est créée à partir du type objet `tNavette`. Elle contient une clé primaire sur l'attribut `numNavette` et une clé étrangère `codeMT` qui référence la table `MoyenTransport`. De plus, la collection imbriquée `navette_voyage` est stockée dans une table de stockage spécifique.

Listing 3 – Création de la table `Navette`

```

CREATE TABLE Navette OF tNavette (
    PRIMARY KEY (numNavette),
    CONSTRAINT fk_navette_moyen_transport
        FOREIGN KEY (codeMT) REFERENCES MoyenTransport
)
NESTED TABLE navette_voyage STORE AS navette_voyage_NT;

```

2.8 Insertion des Données

Dans cette section, nous avons inséré des données cohérentes et représentatives dans les différentes tables du système. Le but est de disposer d'un jeu de données de test pour valider la structure et exécuter les requêtes demandées. la liste des insertions se trouvent dans le fichier `SQL3/inserts.sql`

- Moyens de transport : quatre types ont été insérés avec des horaires et des capacités distinctes (MET, BUS, TRA, TRN). Chaque enregistrement est instancié à partir du type objet `tMoyenTransport`.
- Stations : quatre stations ont été insérées avec des noms significatifs tels que Cité U, Hai El Badr, El Harrach et Univ. Les positions géographiques (longitude et latitude) sont précisées et les stations principales sont identifiées par la valeur 'TRUE' dans l'attribut principale.
- Lignes : deux lignes (M001 et B001) ont été créées en associant à chaque ligne une station de départ, une station d'arrivée et un moyen de transport. Les références sont correctement instanciées via des sous-requêtes `SELECT REF(...)`.

- Tronçons : deux tronçons ont été insérés pour relier les stations d'une ligne. Chaque tronçon contient les références aux deux stations qu'il relie, ainsi qu'à la ligne à laquelle il est associé.
- Navettes : deux navettes sont insérées, chacune appartenant à un moyen de transport spécifique et affectée à une ligne.
- Voyages : deux voyages sont enregistrés. Le premier se déroule à 8h00 avec 120 voyageurs et une observation RAS, le second à 10h00 avec 90 voyageurs et une observation Retard. Les dates sont correctement formatées avec la fonction TO_DATE(...).

Voici un exemple d'insertion :

Listing 4 – Insertion d'un moyen de transport et d'une station

```
INSERT INTO MoyenTransport VALUES (
    tMoyenTransport('MET', 'MET', '06:00', '23:00', 80000,
    t_set_ref_lignes(), t_set_ref_navette())
);

INSERT INTO Voyage VALUES (
    tVoyage(
        1,
        DATE '2025-01-01',
        TO_DATE('2025-01-01 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),
        45,
        'A',
        120,
        'RAS',
        (SELECT REF(n) FROM Navette n WHERE n.numNavette = 1)
    )
);
```

2.9 Mise à jour des collections imbriquées par références

Une fois les données principales insérées dans les différentes tables objets (Ligne, Navette, Troncon, etc.), il est nécessaire de mettre à jour les collections imbriquées de type TABLE OF REF pour garantir l'intégrité des relations entre objets.

Par exemple, chaque objet Ligne contient deux collections :

- ligne_navette : contient les références vers les navettes qui desservent cette ligne.
- ligne_troncon : contient les références vers les tronçons qui composent cette ligne.

On utilise donc des requêtes UPDATE pour remplir ces collections avec des ensembles d'objets déréférencés. Voici un exemple concret pour la ligne M001 :

```
UPDATE Ligne l
SET l.ligne_navette = t_set_ref_navette(
    (SELECT REF(n) FROM Navette n
    WHERE n.navette_ligne = REF(l) AND ROWNUM = 1)
),
```

```

l.ligne_troncon = t_set_ref_troncon(
    (SELECT REF(t) FROM Troncon t
     WHERE (t.Ligne1 = REF(l) OR t.Ligne2 = REF(l)) AND ROWNUM = 1)
)
WHERE l.codeLigne = 'M001';

```

Cette requête :

- cherche les navettes associées à la ligne M001 et insère leur REF dans la collection ligne_navette.
- cherche également les tronçons reliés à cette ligne et les ajoute dans ligne_troncon.

Le même principe est appliqué pour les entités Station, Troncon et MoyenTransport afin de lier correctement toutes les références nécessaires via leurs collections internes.

Ces opérations sont indispensables pour permettre un fonctionnement correct des méthodes objets définies dans le corps des types.

2.10 Requêtes SQL

Dans cette section, nous illustrons trois requêtes SQL exécutées sur notre base de données objet-relationnelle afin d'extraire des informations pertinentes sur les voyages, les lignes et les navettes.

Requête 1 : Voyages avec problèmes Cette requête permet de lister tous les voyages ayant rencontré un quelconque problème (tels que panne, retard, ou accident). Elle affiche le numéro du voyage, la date, le code du moyen de transport utilisé ainsi que le numéro de la navette correspondante.

Listing 5 – Lister les voyages ayant un problème

```

SELECT
    v.numVoyage,
    TO_CHAR(v.date_voyage, 'DD-MM-YYYY') AS date_voyage,
    Deref(Deref(v.voyage_navette).codeMT).codeMT AS
        moyen_transport_code,
    Deref(v.voyage_navette).numNavette AS navette_num
FROM Voyage v
WHERE LOWER(v.observation) LIKE '%probleme%'
OR LOWER(v.observation) LIKE '%panne%'
OR LOWER(v.observation) LIKE '%retard%'
OR LOWER(v.observation) LIKE '%accident%';

```

```

SQL> SELECT
2   v.numVoyage,
3   TO_CHAR(v.date_voyage, 'DD-MM-YYYY') AS date_voyage,
4   Deref(Deref(v.voyage_navette).codeMT).codeMT AS moyen_transport_code,
5   Deref(v.voyage_navette).numNavette AS navette_num
6 FROM Voyage v
7 WHERE LOWER(v.observation) LIKE '%probleme%'
8      OR LOWER(v.observation) LIKE '%panne%'
9      OR LOWER(v.observation) LIKE '%retard%'
10     OR LOWER(v.observation) LIKE '%accident%';

NUMVOYAGE DATE_VOYAG MOYEN_TRANSPORT_CODE
-----
NAVETTE_NUM
-----
2 01-01-2025 MET
1

```

FIGURE 13 – Execution requête 1 SQL3

Requête 2 : Lignes desservant une station principale

Cette requête retourne toutes les lignes dont la station de départ ou d'arrivée est une station principale. Cela permet de repérer les lignes stratégiques du réseau.

Listing 6 – Lister les lignes avec station principale

```

SELECT l.codeLigne ,
       Deref(l.stationDepart).nom_station AS station_depart ,
       Deref(l.stationArrivee).nom_station AS station_arrivee
FROM Ligne l
WHERE Deref(l.stationDepart).principale = 'TRUE'
      OR Deref(l.stationArrivee).principale = 'TRUE';

```

```

SQL> SELECT l.codeLigne,
2   Deref(l.stationDepart).nom_station AS station_depart,
3   Deref(l.stationArrivee).nom_station AS station_arrivee
4 FROM Ligne l
5 WHERE Deref(l.stationDepart).principale = 'TRUE'
6      OR Deref(l.stationArrivee).principale = 'TRUE';

CODELIGNE
-----
STATION_DEPART
-----
STATION_ARRIVEE
-----
M001
Cit@ U
Univ

```

FIGURE 14 – Execution requête 2 SQL3

Requête 3 : Navettes les plus actives Cette requête identifie la ou les navettes ayant effectué le plus grand nombre de voyages pendant le mois de janvier 2025. Elle affiche le numéro de la navette, son type de transport, son année de mise en service, ainsi que le nombre de voyages correspondants.

Listing 7 – Navette la plus active en janvier 2025

```

SELECT
  Deref(v.voyage_navette).numNavette AS num_navette ,
  Deref(Deref(v.voyage_navette).codeMT).typeMt AS type_transport ,
  Deref(v.voyage_navette).annee AS annee_mise_service ,
  COUNT(*) AS nombre_voyages

```



```

FROM
    Voyage v
WHERE
    v.date_voyage BETWEEN TO_DATE('01-01-2025', 'DD-MM-YYYY')
                    AND TO_DATE('31-01-2025', 'DD-MM-YYYY')
GROUP BY
    Deref(v.voyage_navette).numNavette,
    Deref(Deref(v.voyage_navette).codeMT).typeMt,
    Deref(v.voyage_navette).annee
HAVING
    COUNT(*) = (
        SELECT MAX(voyage_count)
        FROM (
            SELECT COUNT(*) as voyage_count
            FROM Voyage v2
            WHERE v2.date_voyage BETWEEN TO_DATE('01-01-2025', 'DD-
                MM-YYYY')
                                AND TO_DATE('31-01-2025', 'DD-
                MM-YYYY')
            GROUP BY Deref(v2.voyage_navette).numNavette
        )
    );

```

```

SQL> SELECT
2     Deref(v.voyage_navette).numNavette AS num_navette,
3     Deref(Deref(v.voyage_navette).codeMT).typeMt AS type_transport,
4     Deref(v.voyage_navette).annee AS annee_mise_service,
5     COUNT(*) AS nombre_voyages
6 FROM
7     Voyage v
8 WHERE
9     v.date_voyage BETWEEN TO_DATE('01-01-2025', 'DD-MM-YYYY')
10    AND TO_DATE('31-01-2025', 'DD-MM-YYYY')
11 GROUP BY
12     Deref(v.voyage_navette).numNavette,
13     Deref(Deref(v.voyage_navette).codeMT).typeMt,
14     Deref(v.voyage_navette).annee
15 HAVING
16     COUNT(*) = (
17         SELECT MAX(voyage_count)
18         FROM (
19             SELECT COUNT(*) as voyage_count
20             FROM Voyage v2
21             WHERE v2.date_voyage BETWEEN TO_DATE('01-01-2025', 'DD-MM-YYYY')
22                AND TO_DATE('31-01-2025', 'DD-MM-YYYY')
23             GROUP BY Deref(v2.voyage_navette).numNavette
24         )
25     );

```

NUM_NAVETTE	TYP	ANNEE_MISE_SERVICE	NOMBRE_VOYAGES
1	MET	2022	2

FIGURE 15 – Execution requête 3 SQL3

Requête 4 : Stations desservies par au moins deux moyens de transport

Cette requête identifie les stations qui sont connectées à au moins deux types de moyens de transport différents (par exemple : métro et bus). Elle vérifie pour chaque station si elle est point de départ ou d'arrivée d'une ligne, puis agrège les types de moyens de transport associés à ces lignes.

La fonction LISTAGG est utilisée pour concaténer les types de transport dans un seul champ, tandis que COUNT(DISTINCT ...) dans la clause HAVING filtre uniquement les stations associées à deux moyens ou plus.

Listing 8 – Navette la plus active en janvier 2025

```
SELECT
    s.codeStation,
    s.nom_station,
    LISTAGG(DEREF(l.Ligne_MoyenTransport).typeMt, ', ')
        WITHIN GROUP (ORDER BY Deref(l.Ligne_MoyenTransport).typeMt
            ) AS moyens_transport
FROM
    Station s,
    Ligne l
WHERE
    s.codeStation = Deref(l.stationDepart).codeStation
    OR s.codeStation = Deref(l.stationArrivee).codeStation
GROUP BY
    s.codeStation,
    s.nom_station
HAVING
    COUNT(DISTINCT Deref(l.Ligne_MoyenTransport).typeMt) >= 2;
```

```
SQL> SELECT
2      s.codeStation,
3      s.nom_station,
4      LISTAGG(Deref(l.Ligne_MoyenTransport).typeMt, ', ')
5          WITHIN GROUP (ORDER BY Deref(l.Ligne_MoyenTransport).typeMt) AS moyens_transport
6 FROM
7     Station s,
8     Ligne l
9 WHERE
10    s.codeStation = Deref(l.stationDepart).codeStation
11    OR s.codeStation = Deref(l.stationArrivee).codeStation
12 GROUP BY
13    s.codeStation,
14    s.nom_station
15 HAVING
16    COUNT(DISTINCT Deref(l.Ligne_MoyenTransport).typeMt) >= 2;

no rows selected
```

FIGURE 16 – Execution requête 4 SQL3

3 Partie II – NoSQL Orienté Documents (MongoDB)

3.1 Modélisation Orientée Document

3.1.1 Introduction

Cette partie explore l'approche NoSQL avec une modélisation orientée « documents », particulièrement adaptée aux bases de données nécessitant flexibilité et performance, comme dans le cas d'un système de gestion des voyages, des navettes, des lignes et des stations. Contrairement aux bases relationnelles, la modélisation orientée documents permet d'imbriquer les données connexes dans un même document, facilitant ainsi les requêtes en lecture.

Nous proposons une modélisation adaptée à ce cas d'usage, ajoutons des données à la base, puis effectuons des requêtes pour extraire des informations. Enfin, nous analysons les choix effectués et leurs limitations.

3.1.2 Modélisation orientée document

1. Proposer une modélisation orientée document Dans le cadre de la gestion des voyages, des navettes, des stations et des lignes, une modélisation orientée document s'avère adaptée car elle permet de regrouper des informations connexes dans un même document. En particulier, les voyages sont des entités complexes avec des données relatives aux navettes, aux moyens de transport, aux stations et aux lignes, qui peuvent toutes être intégrées dans un seul document. Cela permet d'éviter des jointures complexes et améliore les performances des requêtes de lecture, qui sont censées être les plus fréquentes dans ce contexte.

Chaque voyage est ainsi modélisé sous forme de document, intégrant toutes les informations pertinentes :

- Voyage : chaque document contient des informations spécifiques au voyage (date, heure, durée, nombre de voyageurs, observation, sens du trajet), ainsi que des détails relatifs à la navette et au moyen de transport associé.
- Navette : chaque voyage contient un sous-document *navette*, qui regroupe les informations concernant la navette (ID, marque, année, et lien vers la ligne).
- Moyen de transport : inclus dans le sous-document *navette*, il comporte des informations sur le type, le code, les horaires d'ouverture et de fermeture, ainsi que le nombre moyen de voyageurs.

Cette approche permet de stocker un seul document pour chaque voyage, facilitant l'accès rapide aux informations liées à ce voyage sans nécessiter de jointures supplémentaires.

2. Illustration de la modélisation Voici un exemple de voyage généré selon cette modélisation :

Listing 9 – Exemple de structure MongoDB

```
{
  "_id": "V0020",
  "date": ISODate("2025-01-03T00:00:00.000Z"),
  "heure_debut": "8:00",
  "duree": 40,
  "sens": "Aller",
  "nb_voyageurs": 579,
  "observation": "Accident",
  "navette": {
    "_id": "N001",
    "marque": "Alstom",
    "annee": 2022,
    "moyen_transport": {
      "code": "MET",
      "type": "Metro",
      "heure_ouverture": "06:00",
      "heure_fermeture": "23:00",
      "nb_moyen_voyageurs": 90000
    },
    "ligne_id": "M001"
  }
}
```

Dans cet exemple, chaque voyage est représenté par un document unique, intégrant non seulement les informations relatives au voyage lui-même, mais aussi celles concernant la navette et le moyen de transport. Le sous-document *navette* contient toutes les informations sur la navette, et à l'intérieur de celui-ci, un sous-document pour le *moyen de transport* est également présent.

3. Justification du choix de conception Le choix de cette modélisation orientée document repose sur plusieurs considérations :

Performance En regroupant toutes les données pertinentes d'un voyage dans un seul document, les requêtes de lecture sont rapides et ne nécessitent pas de jointures, contrairement à une approche relationnelle qui nécessiterait de combiner plusieurs tables. Cela est particulièrement utile pour des systèmes où les lectures dominent.

Simplicité Cette approche permet une gestion simple des données, avec une structure qui reflète directement les entités du système sans nécessiter de transformations complexes.

Flexibilité Les documents peuvent évoluer indépendamment les uns des autres, ce qui permet d'ajouter facilement de nouveaux attributs aux voyages ou aux navettes sans impacter l'ensemble de la base de données.

Scalabilité MongoDB, étant une base NoSQL, permet une montée en charge horizontale facile. Chaque voyage peut être stocké indépendamment, facilitant la distribution des données sur plusieurs serveurs.

4. Inconvénients de la conception Cependant, cette approche présente plusieurs inconvénients qu'il convient de considérer :

Redondance des données Les informations sur les navettes et les moyens de transport sont dupliquées pour chaque voyage. Par exemple, la même navette peut être associée à plusieurs voyages, ce qui conduit à une duplication des informations (par exemple, la marque et l'année de la navette).

Problèmes de mise à jour Si des informations sur une navette ou un moyen de transport changent, il faudra mettre à jour chaque document de voyage qui contient ces données, ce qui peut entraîner des incohérences si la mise à jour n'est pas correctement effectuée.

Limites de taille des documents MongoDB impose une limite de taille pour chaque document (16 Mo), ce qui peut poser problème si les documents deviennent trop volumineux, notamment en cas de voyages comportant de nombreux détails (par exemple, un grand nombre de voyageurs ou d'observations).

Difficultés analytiques Les données fortement imbriquées peuvent compliquer certaines requêtes analytiques, notamment les agrégations complexes ou les croisements avec d'autres collections.

Conclusion Cette modélisation orientée document est particulièrement adaptée à la lecture rapide et à la gestion de données très imbriquées, mais elle nécessite une gestion rigoureuse de la redondance et des mises à jour des documents.

3.2 Insertion des Données

B. Remplir la base de données

Afin de remplir la base de données et d'augmenter son volume, un script en JavaScript a été utilisé pour insérer des données dans toutes les collections concernées. Le processus suit plusieurs étapes pour garantir que les données couvrent une période étendue et incluent un grand nombre de voyages.

1. Génération de données initiales

- **Stations** : Le script commence par insérer des informations concernant plusieurs stations, comme "Cité U", "Centre Ville", et "Gare Centrale". Ces stations sont associées à des informations géographiques (latitude, longitude) et à un statut qui détermine si elles sont principales ou non.
- **Lignes de transport** : Des lignes de transport sont définies entre différentes stations. Chaque ligne a un identifiant unique et associe un départ et une arrivée, ainsi que des tronçons spécifiques.
- **Navettes** : Les navettes de transport (par exemple, des métros, des bus, etc.) sont ensuite ajoutées, chacune étant associée à une ligne

de transport et à un moyen de transport spécifique (métro, bus, tramway, train).

2. Insertion de voyages

- Le script génère des voyages de transport sur une période allant du 1er janvier au 1er mars 2025. Chaque navette effectue 2 voyages par jour, un dans chaque sens (Aller et Retour). Pour chaque voyage, des informations telles que la durée, le nombre de passagers et l'observation (par exemple, "RAS", "Retard", "Panne", etc.) sont ajoutées.
- De plus, le script insère également des voyages spécifiques avec l'observation "RAS" sur des dates prédéfinies (1er janvier, 2 janvier et 3 janvier 2025).
- Chaque voyage est attribué à une navette, en incluant des détails sur la marque, l'année de fabrication et le moyen de transport.

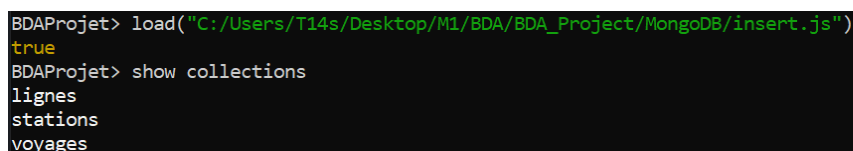
3. Augmentation du volume des données

- Pour augmenter le volume des voyages dans la base de données, le script génère de manière aléatoire des informations sur la durée des voyages, le nombre de voyageurs (qui varie entre 80 et 180 passagers) et l'observation pour chaque voyage. De plus, des voyages sont ajoutés pour plusieurs navettes chaque jour sur toute la période spécifiée.
- En plus des voyages quotidiens, un ensemble de voyages avec un nombre de passagers supérieur à 10 000 est inséré dans la collection voyages afin d'étendre la diversité et la taille des données.

4. Insertion des données dans la base

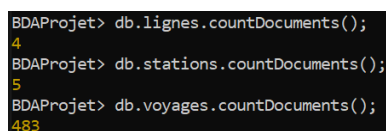
- Une fois que les données sont générées, elles sont insérées dans les collections MongoDB à l'aide des méthodes insertMany. Cela permet d'ajouter efficacement de grands volumes de données dans les collections stations, lignes, et voyages.

Les captures d'écran ci-dessous montrent l'exécution du script, avec l'insertion de données dans les différentes collections de la base de données MongoDB.



```
BDAProjet> load("C:/Users/T14s/Desktop/M1/BDA/BDA_Project/MongoDB/insert.js")
true
BDAProjet> show collections
lignes
stations
voyages
```

FIGURE 17 – Insertion mongoDB



```
BDAProjet> db.lignes.countDocuments();
4
BDAProjet> db.stations.countDocuments();
5
BDAProjet> db.voyages.countDocuments();
483
```

FIGURE 18 – Verifications de l'insertion mongoDB

3.3 Requêtes MongoDB

1. Afficher tous les voyages effectués en date du 01-01-2025

Pour afficher tous les voyages effectués à une date spécifique (01-01-2025), nous avons utilisé une requête de recherche (find) dans la collection voyages. La requête filtre les documents en fonction de la date de chaque voyage. Ensuite, les détails des voyages sont affichés, y compris l’ID du voyage, la date, l’heure de départ, la durée, le nombre de voyageurs, l’observation, et les informations liées à la navette et au moyen de transport. Un formatage a été appliqué pour afficher ces informations de manière lisible.

```
BDAProjet> db.voyages.find({ date: new Date("2025-01-01") }).forEach(voyage => {
...   print(' Voyage ID: ${voyage._id}`');
...   print(' Date: ${voyage.date}`');
...   print(' Heure de début: ${voyage.heure_debut}`');
...   print(' Durée: ${voyage.duree} minutes`');
...   print(' Sens: ${voyage.sens}`');
...   print(' Nombre de voyageurs: ${voyage.nb_voyageurs}`');
...   print(' Observation: ${voyage.observation}`');
...   print(' Navette ID: ${voyage.navette._id}`');
...   print(' Marque: ${voyage.navette.marque}`');
...   print(' Année: ${voyage.navette.annee}`');
...   print(' Moyen de transport: ${voyage.navette.moyen_transport.type}`');
...   print(' Ligne ID: ${voyage.navette.ligne_id}`');
...   print("-----");
... });
Voyage ID: V0001
Date: Wed Jan 01 2025 01:00:00 GMT+0100 (heure normale d'Afrique de l'Ouest)
Heure de début: 08:00
Durée: 45 minutes
Sens: Aller
Nombre de voyageurs: 100
Observation: RAS
Navette ID: N001
Marque: Alstom
Année: 2022
Moyen de transport: Métro
Ligne ID: M001
-----
Voyage ID: V0004
BDAProjet>
Heure de début: 8:00
Durée: 50 minutes
Sens: Aller
Nombre de voyageurs: 124
Observation: Accident
Navette ID: N001
Marque: Alstom
Année: 2022
Moyen de transport: Métro
Ligne ID: M001
```

FIGURE 19 – Exécution de Requête 1 mongoDB

2. Dans une collection BON-Voyage, récupérer tous les voyages n’ayant enregistré aucun problème

Dans cette requête, nous avons utilisé l’agrégation pour filtrer les voyages qui n’ont enregistré aucun problème (indiqués par “RAS” dans le champ observation). Après avoir filtré les voyages, nous avons projeté les champs pertinents, comme le numéro du voyage, la ligne associée, la date, l’heure, le sens du voyage, le moyen de transport, et le numéro de la navette. Enfin, les résultats ont été sauvegardés dans une nouvelle collection appelée BON-Voyage.

```

BDAProjet> db.voyages.aggregate([
...   {
...     $match: { observation: "RAS" } // Garder uniquement les voyages sans problème
...   },
...   {
...     $project: {
...       _id: 1, // numéro du voyage
...       numligne: "$navette.ligne_id", // ligne associée à la navette
...       date: 1, // date du voyage
...       heure_debut: 1, // heure de début
...       sens: 1, // sens (Aller / Retour)
...       moyen_transport: "$navette.moyen_transport.code", // code du moyen de transport
...       numNavette: "$navette._id" // numéro de la navette
...     }
...   },
...   {
...     $out: "BON-Voyage" // Sauvegarder directement dans une nouvelle collection BON-Voyage
...   }
... ]);

BDAProjet> show collections
BON-Voyage
lignes
stations
voyages

```

FIGURE 20 – Exécution de Requête 2 mongoDB

3. Récupérer dans une nouvelle collection Ligne-Voyages, les numéros de lignes et le nombre total de voyages effectués (par ligne), trié par ordre décroissant

Ici, nous avons utilisé une requête d'agrégation pour regrouper les voyages par ligne_id et compter le nombre total de voyages par ligne à l'aide de l'opérateur \$group. Ensuite, les résultats ont été triés par ordre décroissant du nombre de voyages à l'aide de l'opérateur \$sort. Enfin, les résultats ont été stockés dans une nouvelle collection appelée Ligne-Voyages pour permettre une consultation future.

```

BDAProjet> db.voyages.aggregate([
...   {
...     $group: {
...       _id: "$navette.ligne_id", // grouper par numéro de ligne
...       total_voyages: { $sum: 1 } // compter le nombre de voyages
...     }
...   },
...   {
...     $sort: { total_voyages: -1 } // trier en ordre décroissant
...   },
...   {
...     $out: "Ligne-Voyages" // stocker le résultat dans la collection Ligne-Voyages
...   }
... ])
BDAProjet> show collections
BON-Voyage
Ligne-Voyages
lignes
stations
voyages
BDAProjet> db["Ligne-Voyages"].countDocuments();
4

```

FIGURE 21 – Exécution de Requête 3 mongoDB

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

Dans cette étape, nous avons effectué une mise à jour de masse sur les voyages effectués par métro avant le 15 janvier 2025. La mise à jour a incrémenté le nombre de voyageurs de 100 pour tous les voyages correspondants à ces critères. L'opérateur \$inc a été utilisé pour augmenter le champ nb_voyageurs de 100, et le filtre s'est basé sur le moyen de transport et la date du voyage.

```
BDAProjet> db.voyages.updateMany(  
...   {  
...     "navette.moyen_transport.code": "MET", // Rechercher les voyages effectués par métro  
...     date: { $lt: new Date("2025-01-15") } // Comparer avec la date avant le 15 janvier 2025  
...   },  
...   {  
...     $inc: { nb_voyageurs: 100 } // Augmenter de 100 le nombre de voyageurs  
...   }  
... );  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 31,  
  modifiedCount: 31,  
  upsertedCount: 0  
}
```

FIGURE 22 – Exécution de Requête 4 mongoDB

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

Afin d'implémenter une version de la requête 3 avec Map-Reduce, nous avons utilisé la fonction mapReduce de MongoDB. La fonction map émet les identifiants de ligne pour chaque voyage, et la fonction reduce additionne le nombre de voyages pour chaque ligne. Les résultats ont été stockés dans une collection distincte nommée Ligne-Voyages-MapReduce, et les données ont été triées par le nombre total de voyages pour chaque ligne.

```
BDAProjet> db.voyages.mapReduce(  
...   mapFunction,  
...   reduceFunction,  
...   {  
...     out: { replace: "Ligne-Voyages-MapReduce" }  
...   }  
... );  
{ result: 'Ligne-Voyages-MapReduce', ok: 1 }  
BDAProjet>  
[  
  { _id: 'M001', value: 123 },  
  { _id: 'T001', value: 120 },  
  { _id: 'R001', value: 120 },  
  { _id: 'B001', value: 120 }  
]
```

FIGURE 23 – Exécution de Requête 5 mongoDB

6. a) Afficher les navettes ayant effectué un nombre maximum de voyages, en précisant le moyen de transport associé

Pour cette requête, nous avons utilisé l'agrégation afin de regrouper les voyages par navette.id. Ensuite, nous avons compté le nombre de voyages effectués par chaque navette et trié les résultats pour afficher celle ayant effectué le plus grand nombre de voyages. En plus du nombre de voyages, les informations sur le moyen de transport associé à chaque navette ont été affichées.

```
BDAProjet> db.voyages.aggregate([ { $group: { _id: { navette_id: "$navette_id", navette_marque: "$navette.marque", moyen_transport: "$navette.moyen_transport.type" }, nombre_voyages: { $sum: 1 } } }, { $sort: { nombre_voyages: -1 } }, { $limit: 1 } ] ).forEach((doc) => { print( 'Navette ID: ${doc._id.navette_id} ); print( 'Marque: ${doc._id.navette_marque} ); print( 'Moyen de transport: ${doc._id.moyen_transport} ); print( 'Nombre de voyages: ${doc.nombre_voyages} );' });
```

FIGURE 24 – Exécution de Requête 6-a mongoDB

6. b) Afficher les moyens de transport dont le nombre de voyageurs dépasse toujours un seuil S donné (par exemple 10000) par jour

Cette requête utilise l'agrégation pour regrouper les voyages par moyen_transport et par date. Ensuite, nous avons calculé le nombre total de voyageurs pour chaque moyen de transport et chaque jour. Les résultats ont été filtrés pour ne garder que ceux où le nombre de voyageurs dépasse un seuil donné (par exemple 10000) tous les jours. Cette approche nous a permis d'identifier les moyens de transport avec un flux de voyageurs constant au-dessus du seuil.

```
[ { jours_depassement: [ { date: '2025-01-01', total: 13426 }, { date: '2025-01-02', total: 14477 }, { date: '2025-01-03', total: 16518 } ], nombre_jours: 3, moyen_transport: 'Métro' }
```

FIGURE 25 – Exécution de Requête 6-b mongoDB

3.4 Analyse

L'analyse de la conception montre que le modèle documentaire de MongoDB est bien adapté aux requêtes, notamment pour les filtrages simples et les agrégations. En regroupant les informations liées dans un même document, nous avons optimisé les performances des requêtes courantes. Cependant, l'augmentation du volume des données peut ralentir les requêtes, notamment pour les agrégations complexes, si l'indexation n'est pas correctement gérée.

Les mises à jour massives, comme l'ajustement du nombre de voyageurs, sont bien prises en charge par MongoDB, mais la concurrence pour l'accès aux documents pourrait poser problème à grande échelle, nécessitant une gestion fine des transactions.

Globalement, la conception actuelle répond efficacement aux besoins des requêtes, mais des ajustements, tels que l'indexation et le partitionnement, seront nécessaires pour garantir la performance à long terme.

3.5 Conclusion

Dans cette partie, nous avons conçu et exécuté plusieurs requêtes complexes sur notre base de données MongoDB, répondant à des besoins spécifiques en matière de gestion des voyages et des informations associées. Les résultats obtenus ont permis de valider l'efficacité de notre conception en termes de flexibilité et de performance pour gérer les données volumineuses. En conclusion, la structure choisie permet non seulement de répondre aux besoins d'interrogation de manière efficace, mais aussi de garantir une évolutivité et une gestion optimisée des informations.

4 Ajout d'un Menu et Application Web

4.1 Partie SQL3

Pour ce projet, un script Node.js a été développé afin de faciliter l'exécution de fichiers SQL à travers un menu interactif. Ce script s'appuie sur les bibliothèques `oracledb`, utilisée pour établir la connexion avec une base de données Oracle, et `readline-sync`, qui permet de capturer les choix de l'utilisateur en ligne de commande. Après configuration de l'Oracle Instant Client via `libDir` et définition des paramètres de connexion dans un objet `dbConfig`, le script affiche un menu listant les fichiers SQL présents dans le dossier SQL3. Les fichiers exécutables sont les suivants :

- `TableSpace_user.sql` : pour la création d'utilisateur, des tables spaces et affectation des droits.
- `TypesMethods.sql` : afin de créer tous les types toutes les méthodes.
- `tables.sql`: création des tables et ajoutes des contraintes.
- `inserts.sql`: insertions des tuples.
- `MethodsExecution.sql`: requêtes exécutant les méthodes.
- `queries.sql`: exécutions des 4 requêtes SQL3.

Avant toute exécution, le fichier `Delete.sql` est lancé automatiquement afin de supprimer les objets existants, garantissant ainsi un environnement propre. Le script gère également les erreurs fréquentes comme `ORA-00942` (table ou vue inexistante) ou `ORA-00900` (instruction invalide), ce qui permet une exécution sécurisée et continue même en cas d'incohérence dans les scripts SQL.

```
Cleanup completed successfully.
SQL Operations Menu:
1. TableSpace_user.sql
2. TypesMethods.sql
3. tables.sql
4. inserts.sql
5. MethodsExecution.sql
6. queries.sql

Enter the number of the file to execute: 1
Executing TableSpace_user.sql...
Executing: CREATE TABLESPACE SQL3_TBS
DATAFILE 'sql3_tbs.dbf'
SIZE 100M
```

FIGURE 26 – Affichage du menu interactif dans le terminal

4.2 Partie Mongo DB

Nous avons ajouté une application web simple permettant d'exécuter les requêtes définies précédemment à travers un menu interactif. L'interface offre à l'utilisateur la possibilité de sélectionner une requête spécifique, d'afficher les résultats correspondants et de visualiser les données sous forme de réponse dynamique. Les captures d'écran suivantes montrent l'interface :

- Page d'accueil : La vue globale de l'application, où l'utilisateur peut choisir une requête à exécuter.

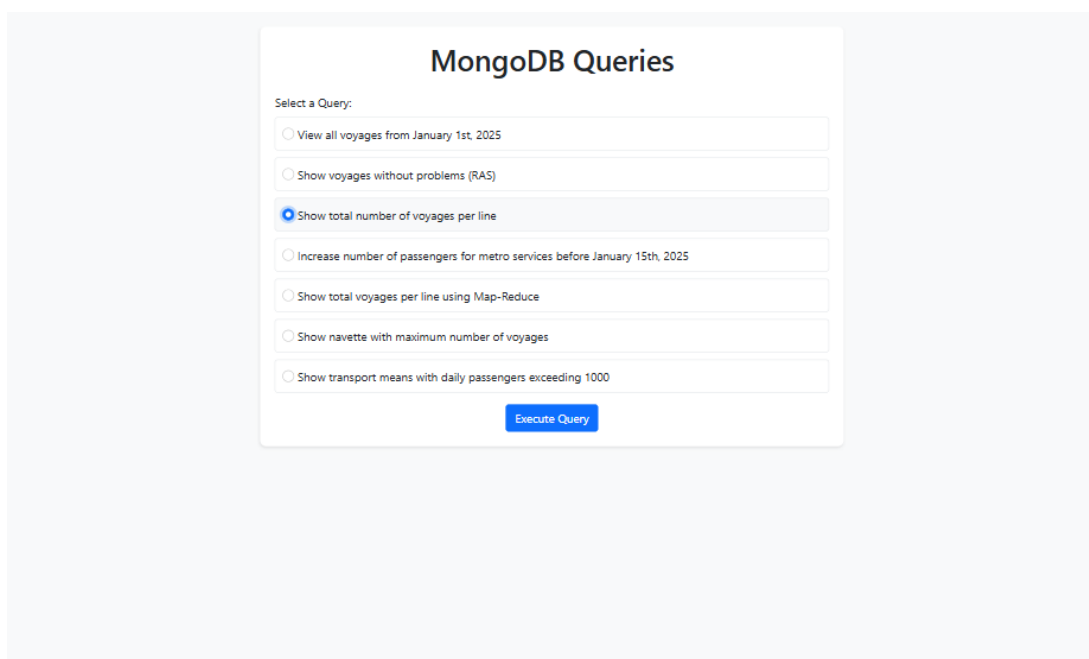


FIGURE 27 – Page index de l'interface web

- Résultats d'une requête exécutée : L'affichage des résultats après avoir sélectionné une requête, permettant une consultation facile des données retournées par la base.

Query Results	
_id	total_voyages
"M001"	126
"T001"	120
"R001"	120
"B001"	120

Back to Queries

FIGURE 28 – Page affichage des résultats des requêtes via l'interface web

Conclusion Général

Ce projet nous a permis d'explorer deux paradigmes complémentaires de gestion de bases de données appliqués à un système de transport urbain dans une ville intelligente : le modèle relationnel-objet avec Oracle SQL3 et le modèle orienté documents avec MongoDB.

La conception relationnelle-objet sous SQL3 s'est révélée rigoureuse et bien adaptée à la structure hiérarchique et fortement liée du système : les entités comme les moyens de transport, lignes, stations, tronçons, navettes et voyages ont pu être modélisées de façon claire à l'aide de types abstraits, de relations bien définies et de méthodes spécifiques. Ce modèle assure l'intégrité des données, une bonne normalisation et facilite les opérations complexes comme les jointures ou les calculs précis (par exemple, le nombre de voyages par navette ou les statistiques sur une période donnée).

En revanche, la modélisation NoSQL orientée documents nous a amenés à repenser la structure des données pour répondre efficacement aux besoins d'interrogation, principalement axés sur les voyages. Ce modèle favorise la redondance contrôlée et la dénormalisation pour optimiser les performances de lecture. Par exemple, en regroupant les informations des navettes, des lignes et des voyages dans un même document, les requêtes sont plus rapides mais au prix d'une cohérence plus difficile à maintenir. Ce choix est pertinent lorsque les lectures sont prioritaires et que la structure est plus souple, notamment pour des analyses orientées utilisateur ou pour des systèmes nécessitant une grande scalabilité.

Ainsi, dans ce cas de figure, SQL3 est plus adapté à une modélisation fine, cohérente et structurée, notamment dans un contexte transactionnel

ou multi-utilisateurs avec des contraintes fortes. MongoDB est plus flexible et performant pour des traitements analytiques sur de grandes volumétries de données ou pour des fonctionnalités orientées services (comme les requêtes géolocalisées sur les voyages ou l’affichage rapide de statistiques globales).

En conclusion, cette double approche nous a permis de mieux comprendre les forces et les limites de chaque technologie et d’adapter notre conception selon les objectifs visés : *cohérence et rigueur avec SQL3, flexibilité et performance avec MongoDB.*

Annexes

.1 Diagramme de classes UML

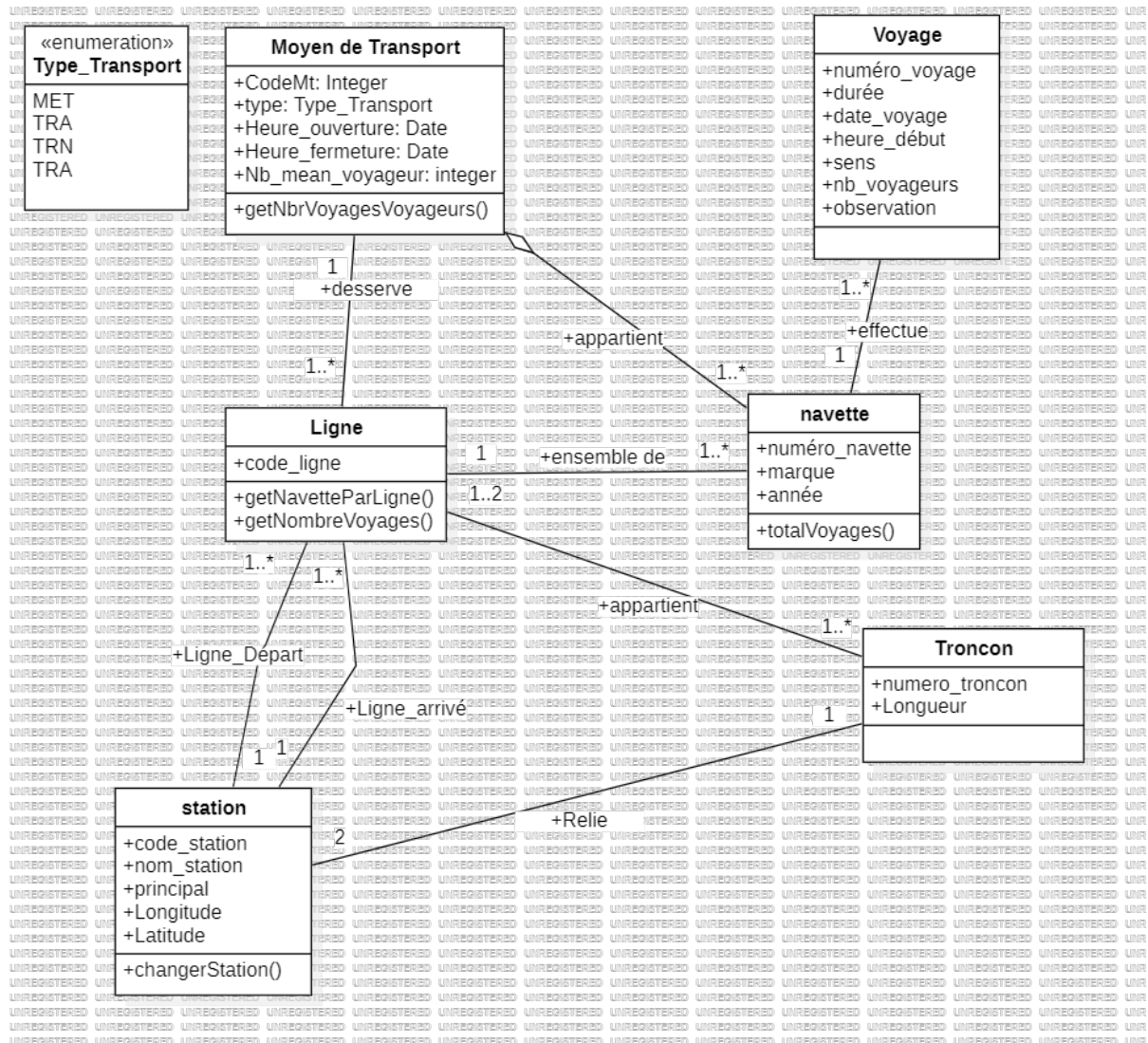


FIGURE 29 – Diagramme de classes du système de transport

.2 Lien vers le projet

Le code source complet du projet, incluant les scripts SQL et MongoDB, les exemples de données ainsi que les captures, est disponible sur le dépôt GitHub suivant :

https://github.com/ryma-tharouma/BDA_Project.git