

SAE 5.02 : Migration SQL vers NoSQL

Marie Challet

Table des matières

1	Introduction	3
2	Création et alimentation des tables avec PostgreSQL	4
2.1	Création de la base de données et des tables	4
2.2	Affichage des données	6
3	Coût des jointures en SQL	7
3.1	Requête de jointure	7
3.2	Analyse des performances	8
3.3	Création d'une vue matérialisée	9
3.4	Analyse des performances après optimisation	10
3.5	Conclusion	11
4	Export de données PostgreSQL vers fichiers JSON	11
4.1	Conversion des enregistrements individuels au format JSON	11
4.2	Agrégation des lignes en tableaux JSON	12
4.3	Export des données JSON dans des fichiers	12
5	Import des données JSON dans MongoDB	13
5.1	Script PowerShell pour reformater les fichiers JSON	13
5.2	Exécution du script PowerShell	14
5.3	Importation des données dans MongoDB	15
6	Coût des jointures avec MongoDB et dénormalisation des données	15
6.1	Jointure entre <code>employees</code> et <code>titles</code>	16
6.2	Jointure entre <code>employees</code> , <code>titles</code> et <code>salaries</code>	16
6.3	Analyse des performances de la jointure	17
6.4	Projection des champs nécessaires	18
6.5	Création d'une collection dénormalisée	20
6.6	Consultation des données dénormalisées	22

7	Conclusions	22
7.1	Avantages de la dénormalisation	22
7.2	Inconvénients de la dénormalisation	22

1 Introduction

Avec l'augmentation exponentielle des données générées par les entreprises, il devient crucial de choisir des bases de données adaptées aux besoins modernes. Les bases relationnelles comme PostgreSQL offrent une structure robuste pour des données normalisées, mais elles peuvent montrer leurs limites en termes de flexibilité et de performance dans des cas de données volumineuses et hétérogènes. À l'inverse, les bases NoSQL, comme MongoDB, permettent de structurer des données dénormalisées sous forme de documents, optimisant ainsi les lectures fréquentes et complexes.

Dans le cadre de cette SAE, j'ai réalisé une migration de données d'une base relationnelle PostgreSQL vers une base orientée documents utilisant MongoDB. Ce projet avait pour objectif de démontrer les avantages et les limites d'une telle transformation, en mettant l'accent sur les performances d'accès et la restructuration des données.

La base de données relationnelle initiale contenait six tables principales, modélisant les employés, les départements, les salaires et les relations entre ces entités. Mon objectif était de transformer cette structure normalisée en une base dénormalisée adaptée à MongoDB, tout en conservant l'intégrité des données et en améliorant les performances des requêtes.

Dans ce rapport :

- Je commence par détailler les étapes de création et d'alimentation des tables sous PostgreSQL.
- Ensuite, j'analyse le coût des jointures en SQL à travers des exemples concrets et des mesures de performance.
- Je décris ensuite le processus d'exportation des données PostgreSQL vers des fichiers JSON, en vue de leur import dans MongoDB.
- Je présente l'import des données dans MongoDB, suivi d'une étude sur le coût des jointures et les bénéfices de la dénormalisation.
- Enfin, je conclus sur les cas d'usage où la dénormalisation est pertinente, tout en abordant ses limites.

Cette expérience m'a permis d'illustrer les différences fondamentales entre les bases relationnelles et non relationnelles, tout en mettant en lumière les défis et opportunités liés à la migration vers MongoDB.

2 Création et alimentation des tables avec PostgreSQL

2.1 Création de la base de données et des tables

Dans cette étape, j'ai transformé les scripts MySQL proposés en syntaxe PostgreSQL afin d'assurer leur compatibilité. Voici le script final utilisé pour créer et alimenter les tables :

```
1 CREATE DATABASE employees;
2
3 \c employees;
4
5 -- Affiche une information
6 SELECT 'CREATING DATABASE STRUCTURE' AS "INFO";
7
8 -- Supprime les tables si elles existent
9 DROP TABLE IF EXISTS dept_emp,
10                      dept_manager,
11                      titles,
12                      salaries,
13                      employees,
14                      departments CASCADE;
15
16 -- Creation des tables
17 CREATE TABLE employees (
18     emp_no      SERIAL          PRIMARY KEY,
19     birth_date   DATE            NOT NULL,
20     first_name   VARCHAR(14)     NOT NULL,
21     last_name    VARCHAR(16)     NOT NULL,
22     gender       CHAR(1)         NOT NULL CHECK (gender IN ('M', 'F'))
23     ), -- ENUM remplace par une contrainte
24     hire_date    DATE            NOT NULL
25 );
26
27 CREATE TABLE departments (
28     dept_no      CHAR(4)         PRIMARY KEY,
29     dept_name     VARCHAR(40)    NOT NULL UNIQUE
30 );
31
32 CREATE TABLE dept_manager (
33     emp_no       INT            NOT NULL,
34     dept_no       CHAR(4)        NOT NULL,
35     from_date     DATE           NOT NULL,
36     to_date       DATE           NOT NULL,
37     PRIMARY KEY (emp_no, dept_no),
38     FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE
39         CASCADE,
40     FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE
```

```

39 CASCADE
40 );
41 CREATE TABLE dept_emp (
42     emp_no      INT          NOT NULL,
43     dept_no     CHAR(4)      NOT NULL,
44     from_date   DATE         NOT NULL,
45     to_date     DATE         NOT NULL,
46     PRIMARY KEY (emp_no, dept_no),
47     FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE
48         CASCADE,
49     FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE
50         CASCADE
51 );
52 CREATE TABLE titles (
53     emp_no      INT          NOT NULL,
54     title       VARCHAR(50)  NOT NULL,
55     from_date   DATE         NOT NULL,
56     to_date     DATE,
57     PRIMARY KEY (emp_no, title, from_date),
58     FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE
59         CASCADE
60 );
61 CREATE TABLE salaries (
62     emp_no      INT          NOT NULL,
63     salary      INT          NOT NULL,
64     from_date   DATE         NOT NULL,
65     to_date     DATE         NOT NULL,
66     PRIMARY KEY (emp_no, from_date),
67     FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE
68         CASCADE
69 );
70 -- Creation des vues
71 CREATE OR REPLACE VIEW dept_emp_latest_date AS
72     SELECT emp_no, MAX(from_date) AS from_date, MAX(to_date) AS
73         to_date
74     FROM dept_emp
75     GROUP BY emp_no;
76 CREATE OR REPLACE VIEW current_dept_emp AS
77     SELECT l.emp_no, dept_no, l.from_date, l.to_date
78     FROM dept_emp d
79     INNER JOIN dept_emp_latest_date l
80     ON d.emp_no = l.emp_no AND d.from_date = l.from_date AND l.

```

```

      to_date = d.to_date;
81
82 -- Chargement des donnees
83
84 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_departments.dump'
85 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_employees.dump'
86 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_dept_emp.dump'
87 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_dept_manager.dump'
88 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_titles.dump'
89 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_salaries1.dump'
90 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_salaries2.dump'
91 \i 'C:\\Users\\chall\\OneDrive\\Documents\\S5\\SAE NoSQL\\Employees
    \\load_salaries3.dump'
92
93 SELECT 'LOADING DATA COMPLETE' AS "INFO";

```

Listing 1 – Création des tables PostgreSQL

2.2 Affichage des données

J'ai utilisé les commandes suivantes pour vérifier la structure des tables et visualiser leur contenu :

```

1 \dt;
2 select * from departements;
3 select * from dept_emp;
4 select * from dept_manager;
5 select * from employees;
6 select * from salaries;
7 select * from titles;

```

Listing 2 – Affichage des tables

Ces commandes confirment que les tables ont été correctement créées et alimentées.

```
employees=# \dt;
```

Liste des relations			
Schéma	Nom	Type	Propriétaire
public	departments	table	postgres
public	dept_emp	table	postgres
public	dept_manager	table	postgres
public	employees	table	postgres
public	salaries	table	postgres
public	titles	table	postgres

(6 lignes)

Screenshot 1.2

3 Coût des jointures en SQL

Les bases relationnelles comme PostgreSQL permettent de manipuler des données structurées avec des relations complexes. Cependant, les jointures peuvent être coûteuses en termes de performances, en particulier lorsque les tables contiennent un grand nombre d'enregistrements. Cette section analyse les coûts des jointures à travers une requête impliquant trois tables principales.

3.1 Requête de jointure

Pour évaluer le coût des jointures en SQL, j'ai réalisé une requête combinant les tables `employees`, `salaries` et `titles`. Cette jointure relie chaque employé à ses salaires et à ses titres, en utilisant des relations basées sur le champ `emp_no` :

```
1 SELECT *
2 FROM employees e
3 NATURAL JOIN salaries s
4 NATURAL JOIN titles t;
```

Listing 3 – Requête de jointure

Le choix de `NATURAL JOIN` permet de simplifier la syntaxe en combinant automatiquement les tables sur les colonnes ayant des noms identiques. Cependant, cette approche peut entraîner des ambiguïtés ou des colonnes superflues si plusieurs colonnes partagent le même nom. Une jointure explicite avec des clés précises peut être envisagée pour optimiser la requête.

emp_no	from_date	to_date	birth_date	first_name	last_name	gender	hire_date	salary	title
10082	1990-01-03	1990-01-15	1963-09-09	Parviz	Lortz	M	1990-01-03	48935	Staff
10555	1992-05-22	1993-01-19	1962-09-25	Vitaly	Picci	M	1989-11-06	73486	Staff
12946	1998-10-07	1999-02-25	1957-11-30	Shim	Antonakopoulos	M	1997-02-14	58373	Staff
13940	1999-03-30	1999-04-30	1961-10-22	Leon	Servieres	F	1999-01-08	40000	Staff
13996	1999-12-23	2000-04-01	1952-03-05	Adib	Pews	F	1993-04-25	57054	Staff
14905	1987-09-14	1987-12-28	1953-04-19	Karsten	Baja	F	1987-09-14	41067	Engineer
15981	1997-02-10	1997-04-10	1962-11-27	Barna	Daescu	F	1986-10-06	40000	Technique Leade
16014	1987-10-15	1987-12-10	1958-05-11	Rance	Zizka	F	1985-12-09	72312	Staff
17047	1987-12-07	1988-03-06	1953-07-19	Qingxiang	Rijsenbrij	F	1987-12-07	43113	Engineer
17122	1992-06-30	1993-06-05	1958-02-18	Luerbio	Journel	F	1992-06-30	40131	Assistant Engin
19083	1987-02-22	1987-03-06	1958-02-17	Nobuyoshi	Asmuth	M	1987-02-22	40000	Engineer
19182	1986-07-28	1987-07-11	1963-01-30	Arvind	Suri	M	1986-07-28	40000	Engineer
20563	1996-03-09	1996-07-28	1955-08-21	Baby	Taegyun	M	1993-06-18	40614	Engineer
21866	1992-11-08	1993-09-08	1960-08-20	Uri	Peak	M	1985-11-25	62179	Staff
22706	1999-05-10	2000-02-07	1961-09-19	Paris	Lung	M	1989-04-01	40000	Engineer
25305	1999-11-12	2000-01-04	1952-02-17	Kish	Braunmuhl	M	1990-04-01	50727	Senior Staff
26090	1997-10-21	1998-01-18	1956-10-19	Douadi	Vural	F	1994-06-17	40000	Engineer
26694	1997-02-22	1997-07-07	1952-09-01	Yaghout	Hashii	F	1994-06-04	40000	Staff
26822	1992-10-01	1993-07-27	1956-08-02	Atreye	Strandh	M	1985-08-29	40000	Engineer
27600	1994-04-21	1994-09-19	1956-05-23	Hiroyasu	Pargas	F	1994-04-21	69682	Staff
27697	1997-04-21	1997-07-05	1957-03-04	Takahito	Shinomoto	M	1991-01-19	58471	Engineer
27701	1997-05-07	1998-01-21	1962-01-21	Ioana	Usery	M	1995-11-20	40000	Senior Staff
27727	1985-06-19	1985-11-11	1954-02-07	Heekeun	Rotem	M	1985-06-19	56078	Technique Leade
28438	1995-07-16	1995-09-11	1961-04-13	Leaf	Ranta	F	1993-05-11	49440	Engineer
29281	1995-09-09	1996-04-14	1957-06-15	Sanjiv	Manders	F	1991-01-18	57238	Staff
30026	1992-09-05	1993-07-18	1962-01-09	Pohua	Langford	M	1992-09-05	65539	Engineer
30835	1996-06-19	1997-05-06	1962-04-12	Candida	Gien	M	1987-02-06	40000	Engineer

Screenshot 2.1

3.2 Analyse des performances

Pour mesurer le coût d'exécution de cette requête, j'ai utilisé la commande `EXPLAIN ANALYZE`, qui fournit des détails sur le plan d'exécution et les temps d'exécution associés :

```

1 EXPLAIN ANALYZE (
2 SELECT *
3 FROM employees e
4 NATURAL JOIN salaries s
5 NATURAL JOIN titles t
6 );

```

Listing 4 – Analyse de la requête

Voici un résumé des résultats obtenus :

- **Temps total d'exécution** : 425.424 ms.
- **Plan d'exécution** : PostgreSQL a utilisé des scans séquentiels sur les tables `salaries` et `titles`, suivis d'une combinaison des résultats avec la table `employees`.
- **Problèmes observés** : Ce plan d'exécution montre que les jointures impliquent plusieurs étapes coûteuses, particulièrement pour des tables volumineuses.

Cette analyse met en évidence l'impact des jointures complexes sur les performances et justifie l'utilisation de solutions d'optimisation, comme les vues matérialisées.

```
employees=# EXPLAIN ANALYZE (
employees=# SELECT *
employees=# FROM employees e
employees=# NATURAL JOIN salaries s
employees=# NATURAL JOIN titles t);

QUERY PLAN

-----
Gather  (cost=12747.57..66414.78 rows=32 width=52) (actual time=258.581..425.003 rows=8435 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Nested Loop  (cost=11747.57..65411.58 rows=13 width=52) (actual time=227.936..380.701 rows=2812 loops=3)
    -> Parallel Hash Join  (cost=11747.15..65404.91 rows=15 width=31) (actual time=227.900..370.256 rows=2812 loops=3)
      Hash Cond: ((s.emp_no = t.emp_no) AND (s.from_date = t.from_date) AND (s.to_date = t.to_date))
      -> Parallel Seq Scan on salaries s  (cost=0.00..27224.20 rows=1185020 width=16) (actual time=0.451..74.854 rows=948
016 loops=3)
        -> Parallel Hash  (cost=5655.69..5655.69 rows=260769 width=23) (actual time=40.721..40.721 rows=147769 loops=3)
          Buckets: 131072  Batches: 4  Memory Usage: 7584kB
          -> Parallel Seq Scan on titles t  (cost=0.00..5655.69 rows=260769 width=23) (actual time=0.006..15.010 rows=1
47769 loops=3)
        -> Index Scan using employees_pkey on employees e  (cost=0.42..0.44 rows=1 width=29) (actual time=0.003..0.003 rows=1 loo
ps=8435)
          Index Cond: (emp_no = s.emp_no)
  Planning Time: 0.710 ms
  Execution Time: 425.424 ms
(14 lignes)
```

Screenshot 2.2

3.3 Création d'une vue matérialisée

Pour optimiser cette requête, j'ai créé une vue matérialisée. Contrairement aux vues classiques, les vues matérialisées stockent les résultats de la requête sous forme de table, permettant des lectures plus rapides :

```
1 CREATE MATERIALIZED VIEW emp_sal_titles AS
2 SELECT *
3 FROM employees e
4 NATURAL JOIN salaries s
5 NATURAL JOIN titles t;
```

Listing 5 – Création d'une vue matérialisée

Cette opération pré-calculée réduit considérablement le temps d'exécution des requêtes futures sur ces données combinées.

```

employees=# CREATE MATERIALIZED VIEW emp_sal_titles AS
employees=# SELECT *
employees=# FROM employees e
employees=# NATURAL JOIN salaries s
employees=# NATURAL JOIN titles t;
SELECT 8435
employees=# SELECT * FROM emp_sal_titles;

```

emp_no	from_date	to_date	birth_date	first_name	last_name	gender	hire_date	salary	title
10751	1999-12-14	2000-12-05	1952-12-02	Jordanka	Ushiana	M	1987-04-15	50052	Engineer
10773	1997-02-09	1997-11-29	1960-07-11	Avishai	Bahr	F	1985-12-01	40000	Staff
12156	1997-06-02	1997-08-11	1958-02-19	Gor	McFarlin	M	1995-11-26	55022	Senior Staff
13530	1993-05-26	1993-08-27	1961-07-15	Irene	Greibach	M	1988-10-14	54524	Engineer
13553	1999-11-27	2000-02-07	1958-02-10	Masato	Cusworth	M	1988-01-10	66601	Staff
14256	1994-05-31	1995-05-31	1952-09-05	Gino	Ferriere	F	1993-12-14	46348	Assistant Engin
14272	1993-07-16	1994-05-11	1957-02-07	Fan	Makinen	F	1993-07-16	44169	Senior Engineer
14315	1999-08-26	1999-09-24	1963-05-24	Baocai	Standera	F	1989-09-19	82500	Senior Staff
15137	1997-05-02	1998-04-02	1956-04-17	Pradeep	Menhardt	F	1997-02-16	55324	Staff
16432	1987-02-16	1987-06-24	1963-05-07	Mohd	Esteva	M	1985-07-10	40000	Engineer
16469	1998-03-06	1998-07-21	1956-01-14	Arto	Luan	F	1993-07-23	42198	Engineer
17196	1997-03-15	1997-10-08	1955-10-10	Quingbo	Fasbender	F	1991-07-10	42804	Senior Engineer
19343	1986-06-02	1986-11-18	1953-10-12	Heping	Hempstead	F	1986-06-02	44351	Staff
20554	1999-04-15	1999-04-20	1956-08-24	Qunsheng	Shumilov	M	1987-04-24	63698	Senior Staff
21241	1993-06-19	1993-09-07	1961-05-30	Tadahiko	Lovengreen	M	1987-01-11	57811	Engineer
21253	1994-08-23	1995-07-18	1962-04-19	Ult	Jakobus	F	1994-08-23	50553	Staff
22015	1995-10-11	1996-08-27	1958-07-15	Sanjoy	Marletta	F	1986-11-08	40000	Engineer
22778	1989-04-07	1989-06-06	1952-05-26	Susumu	Bernick	M	1989-04-07	45799	Senior Engineer
23361	1993-09-15	1994-06-29	1959-02-27	Przemyslaw	Solovay	M	1992-04-17	40000	Engineer
23917	1991-04-07	1991-12-19	1962-03-25	Dipayan	Serot	F	1991-04-07	53021	Staff
26259	1990-10-03	1991-02-12	1953-05-06	Ortrun	Vidya	M	1990-10-03	40000	Engineer
27606	1993-04-26	1993-08-30	1955-11-07	Murthy	Uhrlik	M	1993-04-26	40000	Technique Leade
28638	1998-09-23	1999-02-28	1962-04-26	Sreekrishna	Gaughan	M	1998-09-23	40000	Staff
28639	1999-10-20	2000-07-24	1964-05-22	Gregory	Jayawardene	M	1986-07-03	66736	Senior Engineer
29297	1988-04-19	1988-05-24	1962-05-20	Mayumi	Mahnke	M	1988-04-19	74161	Engineer

Screenshot 2.3

3.4 Analyse des performances après optimisation

La commande EXPLAIN ANALYZE a également été utilisée pour évaluer le coût d'exécution d'une requête sur la vue matérialisée :

```
1 EXPLAIN ANALYZE (SELECT * FROM emp_sal_titles);
```

Listing 6 – Analyse de la vue matérialisée

Voici les résultats obtenus après optimisation :

- **Temps total d'exécution** : 2.174 ms.
- **Amélioration** : La vue matérialisée a permis une réduction significative du temps d'exécution (plus de 99% de gain).
- **Impact** : Cette optimisation est particulièrement bénéfique pour les requêtes fréquentes sur les mêmes données jointes, au prix d'une maintenance accrue lors de la mise à jour des données sources.

```

employees=# EXPLAIN ANALYZE (SELECT * FROM emp_sal_titles);
QUERY PLAN
-----
Seq Scan on emp_sal_titles (cost=0.00..173.35 rows=8435 width=50) (actual time=0.049..1.383 rows=8435 loops=1)
Planning Time: 0.234 ms
Execution Time: 2.174 ms
(3 lignes)

```

Screenshot 2.4

3.5 Conclusion

L'utilisation des vues matérialisées démontre une nette amélioration des performances pour des requêtes complexes impliquant des jointures. Cependant, cette approche peut introduire une surcharge dans les processus de maintenance, car la vue matérialisée doit être actualisée régulièrement pour refléter les modifications dans les tables sources. Cette stratégie est particulièrement adaptée lorsque les données sont relativement statiques ou que les lectures fréquentes surpassent les mises à jour.

4 Export de données PostgreSQL vers fichiers JSON

Dans cette section, les données des tables PostgreSQL ont été converties et exportées au format JSON en vue de leur import dans MongoDB. L'utilisation des fonctions `row_to_json`, `json_agg` et de la commande `COPY` a permis de réaliser cette conversion.

4.1 Conversion des enregistrements individuels au format JSON

Pour convertir les enregistrements des tables `employees`, `salaries`, `titles`, `departments`, `dept_emp`, et `dept_manager` en format JSON, j'ai utilisé la fonction `row_to_json`. Cette fonction transforme chaque ligne en un document JSON.

```
1 SELECT row_to_json(e) FROM employees e;  
2 SELECT row_to_json(s) FROM salaries s;  
3 SELECT row_to_json(t) FROM titles t;  
4 SELECT row_to_json(d) FROM departments d;  
5 SELECT row_to_json(de) FROM dept_emp de;  
6 SELECT row_to_json(dm) FROM dept_manager dm;
```

Listing 7 – Conversion des lignes en JSON

Ces commandes retournent un document JSON pour chaque enregistrement, permettant ainsi de vérifier la conversion des données.

```

SQL Shell (psql)
employees=# SELECT row_to_json(e) FROM employees e;
row_to_json
-----
{"emp_no":10001,"birth_date":"1953-09-02","first_name":"Georgi","last_name":"Facello","gender":"M","hire_date":"1986-06-26"}
{"emp_no":10002,"birth_date":"1964-06-02","first_name":"Bezael","last_name":"Simmel","gender":"F","hire_date":"1985-11-21"}
{"emp_no":10003,"birth_date":"1959-12-03","first_name":"Parto","last_name":"Bamford","gender":"M","hire_date":"1986-08-28"}
{"emp_no":10004,"birth_date":"1954-05-01","first_name":"Chirstian","last_name":"Koblick","gender":"M","hire_date":"1986-12-01"}
{"emp_no":10005,"birth_date":"1955-01-21","first_name":"Kyoichi","last_name":"Maliniak","gender":"M","hire_date":"1989-09-12"}
{"emp_no":10006,"birth_date":"1953-04-20","first_name":"Anneke","last_name":"Preusig","gender":"F","hire_date":"1989-06-02"}
{"emp_no":10007,"birth_date":"1957-05-23","first_name":"Tzvetan","last_name":"Zielinski","gender":"F","hire_date":"1989-02-10"}
{"emp_no":10008,"birth_date":"1958-02-19","first_name":"Saniya","last_name":"Kalloufi","gender":"M","hire_date":"1994-09-15"}
{"emp_no":10009,"birth_date":"1952-04-19","first_name":"Sumant","last_name":"Peac","gender":"F","hire_date":"1985-02-18"}
{"emp_no":10010,"birth_date":"1963-06-01","first_name":"Duangkiew","last_name":"Piveteau","gender":"F","hire_date":"1989-08-24"}
{"emp_no":10011,"birth_date":"1983-11-07","first_name":"Mary","last_name":"Sluis","gender":"F","hire_date":"1990-01-22"}
{"emp_no":10012,"birth_date":"1960-10-04","first_name":"Patricio","last_name":"Bridgland","gender":"M","hire_date":"1992-12-18"}
{"emp_no":10013,"birth_date":"1963-06-07","first_name":"Eberhardt","last_name":"Terkki","gender":"M","hire_date":"1985-10-20"}
{"emp_no":10014,"birth_date":"1956-02-12","first_name":"Berni","last_name":"Genin","gender":"M","hire_date":"1987-03-11"}
{"emp_no":10015,"birth_date":"1959-08-19","first_name":"Guoxiang","last_name":"Nooteboom","gender":"M","hire_date":"1987-07-02"}
{"emp_no":10016,"birth_date":"1961-05-02","first_name":"Kazuhiro","last_name":"Cappelletti","gender":"M","hire_date":"1995-01-27"}
{"emp_no":10017,"birth_date":"1958-07-06","first_name":"Cristinel","last_name":"Bouloucos","gender":"F","hire_date":"1993-08-03"}
{"emp_no":10018,"birth_date":"1954-06-19","first_name":"Kazuhide","last_name":"Peha","gender":"F","hire_date":"1987-04-03"}
{"emp_no":10019,"birth_date":"1953-01-23","first_name":"Lillian","last_name":"Haddadi","gender":"M","hire_date":"1999-04-30"}
{"emp_no":10020,"birth_date":"1952-12-24","first_name":"Mayuko","last_name":"Warwick","gender":"M","hire_date":"1991-01-26"}
{"emp_no":10021,"birth_date":"1960-02-20","first_name":"Ramzi","last_name":"Erde","gender":"M","hire_date":"1988-02-10"}
{"emp_no":10022,"birth_date":"1952-07-08","first_name":"Shahaf","last_name":"Famili","gender":"M","hire_date":"1995-08-22"}
{"emp_no":10023,"birth_date":"1953-09-29","first_name":"Bojan","last_name":"Montemayor","gender":"F","hire_date":"1989-12-17"}
{"emp_no":10024,"birth_date":"1958-09-05","first_name":"Suzette","last_name":"Pettey","gender":"F","hire_date":"1997-05-19"}
{"emp_no":10025,"birth_date":"1958-10-31","first_name":"Prasadram","last_name":"Heyers","gender":"M","hire_date":"1987-08-17"}
{"emp_no":10026,"birth_date":"1953-04-03","first_name":"Yongqiao","last_name":"Berztiss","gender":"M","hire_date":"1995-03-20"}
{"emp_no":10027,"birth_date":"1962-07-10","first_name":"Divier","last_name":"Reistad","gender":"F","hire_date":"1989-07-07"}
{"emp_no":10028,"birth_date":"1963-11-26","first_name":"Domenick","last_name":"Tempesti","gender":"M","hire_date":"1991-10-22"}
{"emp_no":10029,"birth_date":"1956-12-13","first_name":"Otmar","last_name":"Herbst","gender":"M","hire_date":"1985-11-20"}
{"emp_no":10030,"birth_date":"1958-07-14","first_name":"Elvis","last_name":"Demeyer","gender":"M","hire_date":"1994-02-17"}
{"emp_no":10031,"birth_date":"1959-01-27","first_name":"Karsten","last_name":"Joslin","gender":"M","hire_date":"1991-09-01"}

```

Screenshot 3.1

4.2 Agrégation des lignes en tableaux JSON

Afin de regrouper tous les enregistrements d'une table dans un tableau JSON unique, j'ai utilisé la fonction `json_agg`. Cette agrégation simplifie l'export des données en créant un fichier unique par table.

```

1 SELECT json_agg(row_to_json(e)) FROM employees e;
2 SELECT json_agg(row_to_json(s)) FROM salaries s;
3 SELECT json_agg(row_to_json(t)) FROM titles t;
4 SELECT json_agg(row_to_json(d)) FROM departments d;
5 SELECT json_agg(row_to_json(de)) FROM dept_emp de;
6 SELECT json_agg(row_to_json(dm)) FROM dept_manager dm;

```

Listing 8 – Agrégation des enregistrements

Ces commandes produisent un tableau JSON contenant tous les documents d'une table.

4.3 Export des données JSON dans des fichiers

Pour sauvegarder les données JSON générées dans des fichiers, j'ai utilisé la commande `\copy`. Cette commande permet d'écrire directement le résultat d'une requête dans un fichier.

```

1 \copy (SELECT json_agg(row_to_json(e)) FROM employees e)
2 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\employees.json';
3
4 \copy (SELECT json_agg(row_to_json(s)) FROM salaries s)
5 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\salaries.json' WITH (
6     FORMAT text);
7
8 \copy (SELECT json_agg(row_to_json(t)) FROM titles t)
9 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\titles.json' WITH (FORMAT
10 text);
11
12 \copy (SELECT json_agg(row_to_json(d)) FROM departments d)
13 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\departments.json' WITH (
14     FORMAT text);
15
16 \copy (SELECT json_agg(row_to_json(de)) FROM dept_emp de)
17 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\dept_emp.json' WITH (
18     FORMAT text);
19
20 \copy (SELECT json_agg(row_to_json(dm)) FROM dept_manager dm)
21 TO 'C:\Users\chall\Documents\S5\SAE NoSQL\dept_manager.json' WITH (
22     FORMAT text);

```

Listing 9 – Exportation des données JSON dans des fichiers

Ces commandes exportent les données de chaque table vers un fichier JSON spécifique situé dans le chemin indiqué. Les fichiers générés sont prêts à être importés dans MongoDB.

5 Import des données JSON dans MongoDB

Lors de l'importation des fichiers JSON générés dans MongoDB, des erreurs ont été rencontrées car les fichiers JSON contenaient des tableaux. Pour résoudre ce problème, un script PowerShell a été utilisé pour reformater les fichiers en une structure adéquate pour MongoDB, où chaque document JSON est sur une ligne distincte.

5.1 Script PowerShell pour reformater les fichiers JSON

Le script suivant vérifie si un fichier JSON existe, lit son contenu, et si ce contenu est un tableau JSON, il reformate chaque élément du tableau en un document JSON compressé et écrit chaque document sur une ligne distincte dans le même fichier.

```

1 param(
2     [string]$filePath

```

```

3 )
4
5 # Verifier si le fichier existe
6 if (Test-Path $filePath) {
7     # Lire le contenu du fichier JSON
8     $jsonContent = Get-Content $filePath | Out-String | ConvertFrom-
        Json
9
10    # Si le JSON est un tableau, traiter chaque element
11    if ($jsonContent -is [System.Collections.IEnumerable]) {
12        # Creer ou ouvrir le fichier de sortie en mode ecriture (
            ecrase l'ancien fichier)
13        $outputFileStream = New-Object System.IO.StreamWriter(
            $filePath, $false)
14
15        # Parcourir chaque element du tableau et l'ecrire dans le
            fichier
16        foreach ($item in $jsonContent) {
17            $jsonItem = $item | ConvertTo-Json -Compress
18            $outputFileStream.WriteLine($jsonItem)
19        }
20
21        # Fermer le flux de sortie
22        $outputFileStream.Close()
23
24        Write-Host "Fichier modifie et sauvegarde sous: $filePath"
25    } else {
26        Write-Host "Le fichier ne contient pas un tableau JSON."
27    }
28 } else {
29     Write-Host "Le fichier specifie n'existe pas : $filePath"
30 }

```

Listing 10 – Script PowerShell pour reformater les fichiers JSON

5.2 Exécution du script PowerShell

Le script PowerShell a été exécuté pour chaque fichier JSON généré précédemment afin de les formater correctement pour MongoDB. Voici les commandes utilisées :

```

1 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\employees.
    json"
2 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\departments
    .json"
3 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\salaries.
    json"

```

```

4 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\titles.json
   "
5 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\
   dept_manager.json"
6 .\json_format.ps1 "C:\Users\chall\Documents\S5\SAE NoSQL\dept_emp.
   json"

```

Listing 11 – Exécution du script PowerShell

Ces commandes assurent que chaque fichier JSON est correctement formaté pour l'importation dans MongoDB.

5.3 Importation des données dans MongoDB

Une fois les fichiers JSON correctement formatés, ils ont été importés dans MongoDB à l'aide de la commande `mongoimport`. Chaque fichier a été importé comme une collection distincte :

```

1 mongoimport --db employee --collection employees --file employees.
   json
2 mongoimport --db employee --collection departments --file
   departments.json
3 mongoimport --db employee --collection salaries --file salaries.json
4 mongoimport --db employee --collection titles --file titles.json
5 mongoimport --db employee --collection dept_manager --file
   dept_manager.json
6 mongoimport --db employee --collection dept_emp --file dept_emp.json

```

Listing 12 – Importation des fichiers JSON dans MongoDB

6 Coût des jointures avec MongoDB et dénormalisation des données

Dans cette partie pour augmenter la rapidité d'exécution j'ai créé des index en plus. Toutes les requêtes MongoDB ont été faites en utilisant le logiciel Datagrip.

```

1 use employee
2
3 db.employees.createIndex({ emp_no: 1 })
4 db.titles.createIndex({ emp_no: 1 })
5 db.salaries.createIndex({ emp_no: 1 })

```

Listing 13 – connexion et création d'index

6.1 Jointure entre employees et titles

Pour lier les collections `employees` et `titles`, j'ai utilisé l'opérateur `$lookup` :

```
1 db.employees.aggregate([
2   {
3     $lookup: {
4       from: "titles",
5       localField: "emp_no",
6       foreignField: "emp_no",
7       as: "titles"
8     }
9   }
10 ]);
```

Listing 14 – Jointure entre employees et titles

Cette commande renvoie un document enrichi pour chaque employé, contenant un tableau de titres associés.

	_id	birth_date	emp_no	first_name	gender	hire_date	last_name	titles
1	675af8636ad0828bf5fd3247	1953-11-07	10011	Mary	F	1990-01-22	Sluis	[{"_id": new ObjectId("675af86ea21c5cf78
2	675af8636ad0828bf5fd3248	1963-06-07	10013	Eberhardt	M	1985-10-20	Terkki	[{"_id": new ObjectId("675af86ea21c5cf78
3	675af8636ad0828bf5fd3249	1956-02-12	10014	Berni	M	1987-03-11	Genin	[{"_id": new ObjectId("675af86ea21c5cf78
4	675af8636ad0828bf5fd324a	1953-09-02	10001	Georgi	M	1986-06-26	Facello	[{"_id": new ObjectId("675af86ea21c5cf78
5	675af8636ad0828bf5fd324b	1958-07-06	10017	Cristinel	F	1993-08-03	Bouloucos	[{"_id": new ObjectId("675af86ea21c5cf78
6	675af8636ad0828bf5fd324c	1959-08-19	10015	Guoxiang	M	1987-07-02	Nooteboom	[{"_id": new ObjectId("675af86ea21c5cf78
7	675af8636ad0828bf5fd324d	1961-05-02	10016	Kazuhiro	M	1995-01-27	Cappelletti	[{"_id": new ObjectId("675af86ea21c5cf78
8	675af8636ad0828bf5fd324e	1954-06-19	10018	Kazuhide	F	1987-04-03	Peha	[{"_id": new ObjectId("675af86ea21c5cf78
9	675af8636ad0828bf5fd324f	1953-01-23	10019	Lillian	M	1999-04-30	Haddadi	[{"_id": new ObjectId("675af86ea21c5cf78
10	675af8636ad0828bf5fd3250	1960-02-20	10021	Ramzi	M	1988-02-10	Erde	[{"_id": new ObjectId("675af86ea21c5cf78
11	675af8636ad0828bf5fd3251	1952-12-24	10020	Mayuko	M	1991-01-26	Warwick	[{"_id": new ObjectId("675af86ea21c5cf78
12	675af8636ad0828bf5fd3252	1952-07-08	10022	Shahaf	M	1995-08-22	Famili	[{"_id": new ObjectId("675af86ea21c5cf78
13	675af8636ad0828bf5fd3253	1953-09-29	10023	Bojan	F	1989-12-17	Montemayor	[{"_id": new ObjectId("675af86ea21c5cf78
14	675af8636ad0828bf5fd3254	1958-10-31	10025	Prasadram	M	1987-08-17	Heyers	[{"_id": new ObjectId("675af86ea21c5cf78
15	675af8636ad0828bf5fd3255	1953-04-03	10026	Yongqiao	M	1995-03-20	Berztiss	[{"_id": new ObjectId("675af86ea21c5cf78
16	675af8636ad0828bf5fd3256	1958-09-05	10024	Suzette	F	1997-05-19	Pettey	[{"_id": new ObjectId("675af86ea21c5cf78
17	675af8636ad0828bf5fd3257	1962-07-10	10027	Divier	F	1989-07-07	Reistad	[{"_id": new ObjectId("675af86ea21c5cf78
18	675af8636ad0828bf5fd3258	1963-11-26	10028	Domenick	M	1991-10-22	Tempesti	[{"_id": new ObjectId("675af86ea21c5cf78
19	675af8636ad0828bf5fd3259	1956-12-13	10029	Otmar	M	1985-11-20	Herbst	[{"_id": new ObjectId("675af86ea21c5cf78
20	675af8636ad0828bf5fd325a	1958-07-14	10030	Elvis	M	1994-02-17	Demeyer	[{"_id": new ObjectId("675af86ea21c5cf785

Screenshot 5.1

6.2 Jointure entre employees, titles et salaries

En ajoutant une étape supplémentaire au pipeline, j'ai lié la collection `salaries` aux résultats de la première jointure :

```
1 db.employees.aggregate([
2   {
3     $lookup: {
```



```

4         from: "titles",
5         localField: "emp_no",
6         foreignField: "emp_no",
7         as: "titles"
8     }
9 },
10 {
11     $lookup: {
12         from: "salaries",
13         localField: "emp_no",
14         foreignField: "emp_no",
15         as: "salaries"
16     }
17 }
18 });

```

Listing 15 – Jointure entre employees, titles et salaries

	_id	birth	gender	first_name	hire	last_name	salaries	titles
1	675af8636ad0828bf5fd3247	1953-11-07	M	Mary	1990-01-22	Sluis	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
2	675af8636ad0828bf5fd3248	1963-06-07	M	Eberhardt	1985-10-20	Terkki	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
3	675af8636ad0828bf5fd3249	1956-02-12	M	Berni	1987-03-11	Genin	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
4	675af8636ad0828bf5fd324a	1953-09-02	M	Georgi	1986-06-26	Facello	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
5	675af8636ad0828bf5fd324b	1958-07-06	F	Cristinel	1993-08-03	Bouloucos	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
6	675af8636ad0828bf5fd324c	1959-08-19	M	Guoxiang	1987-07-02	Nooteboom	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
7	675af8636ad0828bf5fd324d	1961-05-02	M	Kazuhito	1995-01-27	Cappelletti	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
8	675af8636ad0828bf5fd324e	1954-06-19	F	Kazuhide	1987-04-03	Peha	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
9	675af8636ad0828bf5fd324f	1953-01-23	M	Lillian	1999-04-30	Haddadi	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
10	675af8636ad0828bf5fd3250	1960-02-20	M	Ramzi	1988-02-10	Erde	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
11	675af8636ad0828bf5fd3251	1952-12-24	M	Mayuko	1991-01-26	Warwick	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
12	675af8636ad0828bf5fd3252	1952-07-08	M	Shahaf	1995-08-22	Famili	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
13	675af8636ad0828bf5fd3253	1953-09-29	F	Bojan	1989-12-17	Montenayor	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
14	675af8636ad0828bf5fd3254	1958-10-31	M	Prasadram	1987-08-17	Heyers	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
15	675af8636ad0828bf5fd3255	1953-04-03	M	Yongqiao	1995-03-20	Berztiss	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
16	675af8636ad0828bf5fd3256	1958-09-05	F	Suzette	1997-05-19	Pettye	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
17	675af8636ad0828bf5fd3257	1962-07-10	F	Divier	1989-07-07	Reistad	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
18	675af8636ad0828bf5fd3258	1963-11-26	M	Domenick	1991-10-22	Tempesti	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
19	675af8636ad0828bf5fd3259	1956-12-13	M	Otmar	1985-11-20	Herbst	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }
20	675af8636ad0828bf5fd325a	1958-07-14	M	Elvis	1994-02-17	Demeyer	{ "_id": new ObjectId("675af867407427b") }	{ "_id": new ObjectId("675af86ee21c5cf") }

Screenshot 5.2

6.3 Analyse des performances de la jointure

J'ai utilisé la méthode `explain()` pour mesurer les performances de cette jointure :

```

1 db.employees.aggregate([
2     { $lookup: { from: "titles", localField: "emp_no", foreignField:
3       "emp_no", as: "titles" } },
4     { $lookup: { from: "salaries", localField: "emp_no",
5       foreignField: "emp_no", as: "salaries" } }
6 ]).explain("executionStats");

```

Listing 16 – Analyse des performances

Le temps d'exécution mesuré est de 259 ms et de 540 ms avec le fetching.

```
employee> db.employees.aggregate([
  {
    $lookup: {
      from: "titles",
      localField: "emp_no",
      foreignField: "emp_no",
      as: "titles"
    }
  },
  {
    $lookup: {
      from: "salaries",
      localField: "emp_no",
      foreignField: "emp_no",
      as: "salaries"
    }
  }
]).explain("executionStats")
[2024-12-22 19:42:13] 1 row retrieved starting from 1 in 540 ms (execution: 259 ms, fetching: 281 ms)
```

Screenshot 5.3

6.4 Projection des champs nécessaires

Pour optimiser le résultat de la jointure, j'ai utilisé une étape de projection avec l'opérateur `$project` afin de limiter les champs retournés et réduire la taille des documents :

```
1 db.employees.aggregate([
2   {
3     $lookup: {
4       from: "titles",
5       localField: "emp_no",
6       foreignField: "emp_no",
7       as: "titles"
8     }
9   },
10  {
11    $lookup: {
12      from: "salaries",
13      localField: "emp_no",
14      foreignField: "emp_no",
15      as: "salaries"
16    }
17  },
18  {
```

```

19     $project: {
20         _id: 0,
21         emp_no: 1,
22         first_name: 1,
23         last_name: 1,
24         titles: {
25             $map: {
26                 input: "$titles",
27                 as: "t",
28                 in: {
29                     title: "$$t.title",
30                     from_date: "$$t.from_date",
31                     to_date: "$$t.to_date"
32                 }
33             }
34         },
35         salaries: {
36             $map: {
37                 input: "$salaries",
38                 as: "s",
39                 in: {
40                     salary: "$$s.salary",
41                     from_date: "$$s.from_date",
42                     to_date: "$$s.to_date"
43                 }
44             }
45         }
46     }
47 }
48 ]);

```

Listing 17 – Projection des champs

	emp_no	first_name	last_name	salaries	titles
1	10011	Mary	Sluis	[{"salary": new NumberInt("48214"), "from_date": "1992-01-22", "to_date": "1996-01-22"}]	[{"title": "Staff", "from_date": "1990-01-22", "to_date": "1996-01-22"}]
2	10013	Eberhardt	Terkki	[{"salary": new NumberInt("40000"), "from_date": "1985-10-20", "to_date": "1988-10-20"}]	[{"title": "Senior Staff", "from_date": "1985-10-20", "to_date": "1988-10-20"}]
3	10014	Berni	Genin	[{"salary": new NumberInt("46168"), "from_date": "1993-12-29", "to_date": "1999-12-29"}]	[{"title": "Engineer", "from_date": "1993-12-29", "to_date": "1999-12-29"}]
4	10001	Georgi	Facello	[{"salary": new NumberInt("71046"), "from_date": "1991-06-25", "to_date": "1998-06-26"}]	[{"title": "Senior Engineer", "from_date": "1986-06-26", "to_date": "1998-06-26"}]
5	10017	Cristinel	Bouloucos	[{"salary": new NumberInt("82163"), "from_date": "1996-08-02", "to_date": "2000-08-03"}]	[{"title": "Senior Staff", "from_date": "2000-08-03", "to_date": "2000-08-03"}]
6	10015	Guoxiang	Nooteboom	[{"salary": new NumberInt("40000"), "from_date": "1992-09-19", "to_date": "1999-09-19"}]	[{"title": "Senior Staff", "from_date": "1992-09-19", "to_date": "1999-09-19"}]
7	10016	Kazuhiro	Cappelletti	[{"salary": new NumberInt("70889"), "from_date": "1998-02-11", "to_date": "1999-02-11"}]	[{"title": "Staff", "from_date": "1998-02-11", "to_date": "1999-02-11"}]
8	10018	Kazuhide	Peha	[{"salary": new NumberInt("59206"), "from_date": "1988-04-02", "to_date": "1999-04-03"}]	[{"title": "Engineer", "from_date": "1987-04-03", "to_date": "1999-04-03"}]
9	10019	Lillian	Haddadi	[{"salary": new NumberInt("44276"), "from_date": "1999-04-30", "to_date": "1999-04-30"}]	[{"title": "Staff", "from_date": "1999-04-30", "to_date": "1999-04-30"}]
10	10021	Ramzi	Erde	[{"salary": new NumberInt("55025"), "from_date": "1988-02-10", "to_date": "1988-02-10"}]	[{"title": "Technique Leader", "from_date": "1988-02-10", "to_date": "1988-02-10"}]
11	10020	Mayuko	Warwick	[{"salary": new NumberInt("40647"), "from_date": "1998-12-30", "to_date": "1999-12-30"}]	[{"title": "Engineer", "from_date": "1997-12-30", "to_date": "1999-12-30"}]
12	10022	Shahaf	Famili	[{"salary": new NumberInt("40000"), "from_date": "1999-09-03", "to_date": "1999-09-03"}]	[{"title": "Engineer", "from_date": "1999-09-03", "to_date": "1999-09-03"}]
13	10023	Bojan	Montemayor	[{"salary": new NumberInt("47883"), "from_date": "1999-09-27", "to_date": "1999-09-27"}]	[{"title": "Engineer", "from_date": "1999-09-27", "to_date": "1999-09-27"}]
14	10025	Presadram	Heyers	[{"salary": new NumberInt("40680"), "from_date": "1989-08-16", "to_date": "1989-08-16"}]	[{"title": "Technique Leader", "from_date": "1987-08-17", "to_date": "1989-08-17"}]
15	10026	Yongqiao	Bertziss	[{"salary": new NumberInt("51730"), "from_date": "1996-03-19", "to_date": "1999-03-20"}]	[{"title": "Engineer", "from_date": "1995-03-20", "to_date": "1999-03-20"}]
16	10024	Suzette	Pettey	[{"salary": new NumberInt("83733"), "from_date": "1998-06-14", "to_date": "1998-06-14"}]	[{"title": "Assistant Engineer", "from_date": "1998-06-14", "to_date": "1998-06-14"}]
17	10027	Divier	Reistad	[{"salary": new NumberInt("40000"), "from_date": "1995-04-02", "to_date": "1999-04-02"}]	[{"title": "Engineer", "from_date": "1995-04-02", "to_date": "1999-04-02"}]
18	10028	Domenick	Tempesti	[{"salary": new NumberInt("40859"), "from_date": "1991-10-22", "to_date": "1999-10-22"}]	[{"title": "Engineer", "from_date": "1991-10-22", "to_date": "1999-10-22"}]
19	10029	Otmar	Herbst	[{"salary": new NumberInt("63163"), "from_date": "1991-09-18", "to_date": "2000-09-17"}]	[{"title": "Senior Engineer", "from_date": "2000-09-17", "to_date": "2000-09-17"}]
20	10030	Elvis	Demeyer	[{"salary": new NumberInt("72795"), "from_date": "1996-02-17", "to_date": "1999-02-17"}]	[{"title": "Engineer", "from_date": "1996-02-17", "to_date": "1999-02-17"}]

Screenshot 5.4

6.5 Création d'une collection dénormalisée

Pour sauvegarder les documents résultants dans une nouvelle collection, j'ai utilisé l'opérateur \$merge :

```

1 db.employees.aggregate([
2   {
3     $lookup: {
4       from: "titles",
5       localField: "emp_no",
6       foreignField: "emp_no",
7       as: "titles"
8     }
9   },
10  {
11    $lookup: {
12      from: "salaries",
13      localField: "emp_no",
14      foreignField: "emp_no",
15      as: "salaries"
16    }
17  },
18  {
19    $project: {
20      _id: 0,
21      emp_no: 1,
22      first_name: 1,
23      last_name: 1,
24      titles: {

```

```

25         $map: {
26             input: "$titles",
27             as: "t",
28             in: { title: "$$t.title", from_date: "$$t.
                    from_date", to_date: "$$t.to_date" }
29         }
30     },
31     salaries: {
32         $map: {
33             input: "$salaries",
34             as: "s",
35             in: { salary: "$$s.salary", from_date: "$$s.
                    from_date", to_date: "$$s.to_date" }
36         }
37     }
38 },
39 {
40     $merge: {
41         into: "denormalized_employees",
42         whenMatched: "merge",
43         whenNotMatched: "insert"
44     }
45 }
46 }
47 ]);

```

Listing 18 – Création de la collection dénormalisée

	_id	birth_date	emp_no	first_name	gender	hire_date	last_name	titles
1	675af8636ad0828bf5fd3247	1953-11-07	10011	Mary	F	1990-01-22	Sluis	["_id": new ObjectId("675af06ea21c5cf78
2	675af8636ad0828bf5fd3248	1963-06-07	10013	Eberhardt	M	1985-10-20	Terkki	["_id": new ObjectId("675af06ea21c5cf78
3	675af8636ad0828bf5fd3249	1956-02-12	10014	Berni	M	1987-03-11	Genin	["_id": new ObjectId("675af06ea21c5cf78
4	675af8636ad0828bf5fd324a	1953-09-02	10001	Georgi	M	1986-06-26	Facello	["_id": new ObjectId("675af06ea21c5cf78
5	675af8636ad0828bf5fd324b	1958-07-06	10017	Cristinel	F	1993-08-03	Bouloucos	["_id": new ObjectId("675af06ea21c5cf78
6	675af8636ad0828bf5fd324c	1959-08-19	10015	Guoxiang	M	1987-07-02	Nooteboom	["_id": new ObjectId("675af06ea21c5cf78
7	675af8636ad0828bf5fd324d	1961-05-02	10016	Kazuhiro	M	1995-01-27	Cappelletti	["_id": new ObjectId("675af06ea21c5cf78
8	675af8636ad0828bf5fd324e	1954-06-19	10018	Kazuhide	F	1987-04-03	Peha	["_id": new ObjectId("675af06ea21c5cf78
9	675af8636ad0828bf5fd324f	1953-01-23	10019	Lillian	M	1999-04-30	Haddadi	["_id": new ObjectId("675af06ea21c5cf78
10	675af8636ad0828bf5fd3250	1960-02-20	10021	Ramzi	M	1988-02-10	Erde	["_id": new ObjectId("675af06ea21c5cf78
11	675af8636ad0828bf5fd3251	1952-12-24	10020	Mayuko	M	1991-01-26	Warwick	["_id": new ObjectId("675af06ea21c5cf78
12	675af8636ad0828bf5fd3252	1952-07-08	10022	Shahaf	M	1995-08-22	Famili	["_id": new ObjectId("675af06ea21c5cf78
13	675af8636ad0828bf5fd3253	1953-09-29	10023	Bojan	F	1989-12-17	Montemayor	["_id": new ObjectId("675af06ea21c5cf78
14	675af8636ad0828bf5fd3254	1958-10-31	10025	Prasadram	M	1987-08-17	Heyers	["_id": new ObjectId("675af06ea21c5cf78
15	675af8636ad0828bf5fd3255	1953-04-03	10026	Yongqiao	M	1995-03-20	Bertziss	["_id": new ObjectId("675af06ea21c5cf78
16	675af8636ad0828bf5fd3256	1958-09-05	10024	Suzette	F	1997-05-19	Petty	["_id": new ObjectId("675af06ea21c5cf78
17	675af8636ad0828bf5fd3257	1962-07-10	10027	Divier	F	1989-07-07	Reistad	["_id": new ObjectId("675af06ea21c5cf78
18	675af8636ad0828bf5fd3258	1963-11-26	10028	Domenick	M	1991-10-22	Tempesti	["_id": new ObjectId("675af06ea21c5cf78
19	675af8636ad0828bf5fd3259	1956-12-13	10029	Otmar	M	1985-11-20	Herbst	["_id": new ObjectId("675af06ea21c5cf78
20	675af8636ad0828bf5fd325a	1958-07-14	10030	Elvis	M	1994-02-17	Demeyer	["_id": new ObjectId("675af06ea21c5cf78

Screenshot 5.5

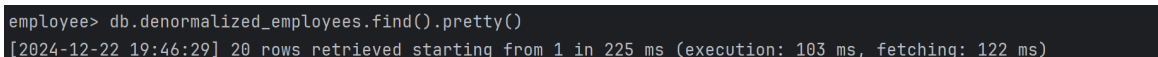
6.6 Consultation des données dénormalisées

Les documents de la collection `denormalized_employees` peuvent être consultés avec la commande suivante :

```
1 db.denormalized_employees.find().pretty();
```

Listing 19 – Consultation des documents dénormalisés

Le temps d'accès aux informations de cette collection a été mesuré à environ 103 ms et 225 ms avec le fetching.



```
employee> db.denormalized_employees.find().pretty()  
[2024-12-22 19:46:29] 20 rows retrieved starting from 1 in 225 ms (execution: 103 ms, fetching: 122 ms)
```

Screenshot 5.6

7 Conclusions

7.1 Avantages de la dénormalisation

La dénormalisation est avantageuse dans les cas suivants :

- **Amélioration des performances en lecture** : Les jointures sont effectuées en amont, réduisant le coût des requêtes.
- **Simplicité d'accès** : Les informations nécessaires sont regroupées dans un seul document.
- **Données stables** : Pour des données peu volatiles, la maintenance est simplifiée.

7.2 Inconvénients de la dénormalisation

Cependant, elle présente des limites dans certains cas :

- **Surcoût en écriture** : Toute mise à jour nécessite une synchronisation des données dénormalisées.
- **Coût de stockage** : La duplication des données peut devenir problématique pour de grands volumes.
- **Usage limité** : Si les requêtes concernent des sous-ensembles de données, la dénormalisation peut être inutile.

En résumé, la dénormalisation est pertinente pour optimiser les lectures fréquentes, à condition de maîtriser ses contraintes sur le stockage et les mises à jour.