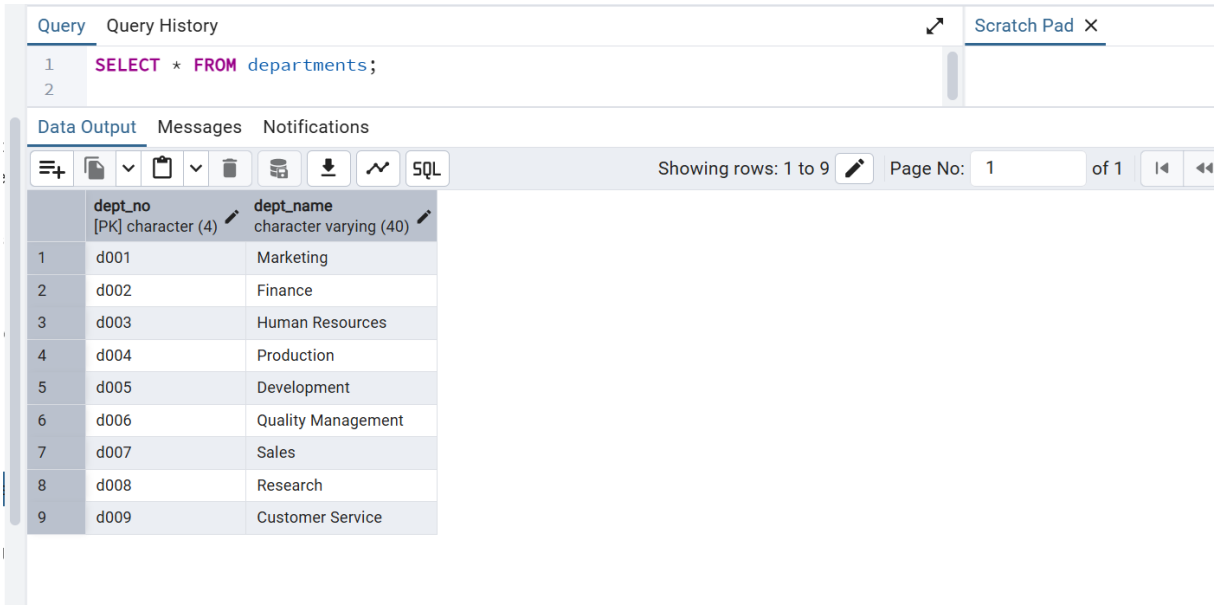
Projet Migration

1. Création et alimentation des tables avec PostgreSQL

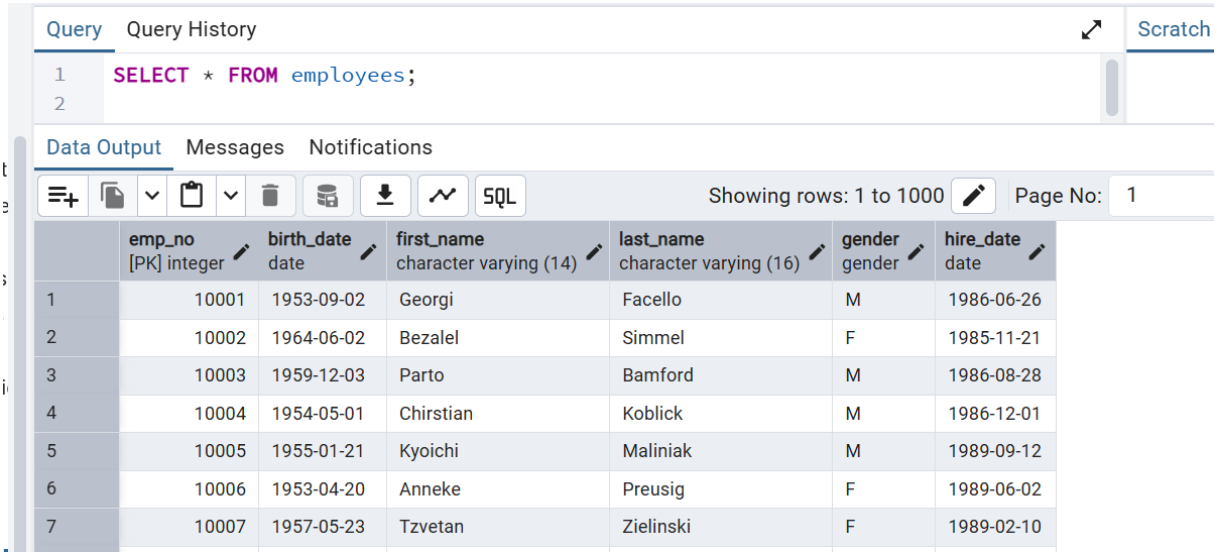
Table Département



```
SELECT * FROM departments;
```

	dept_no [PK] character (4)	dept_name character varying (40)
1	d001	Marketing
2	d002	Finance
3	d003	Human Resources
4	d004	Production
5	d005	Development
6	d006	Quality Management
7	d007	Sales
8	d008	Research
9	d009	Customer Service

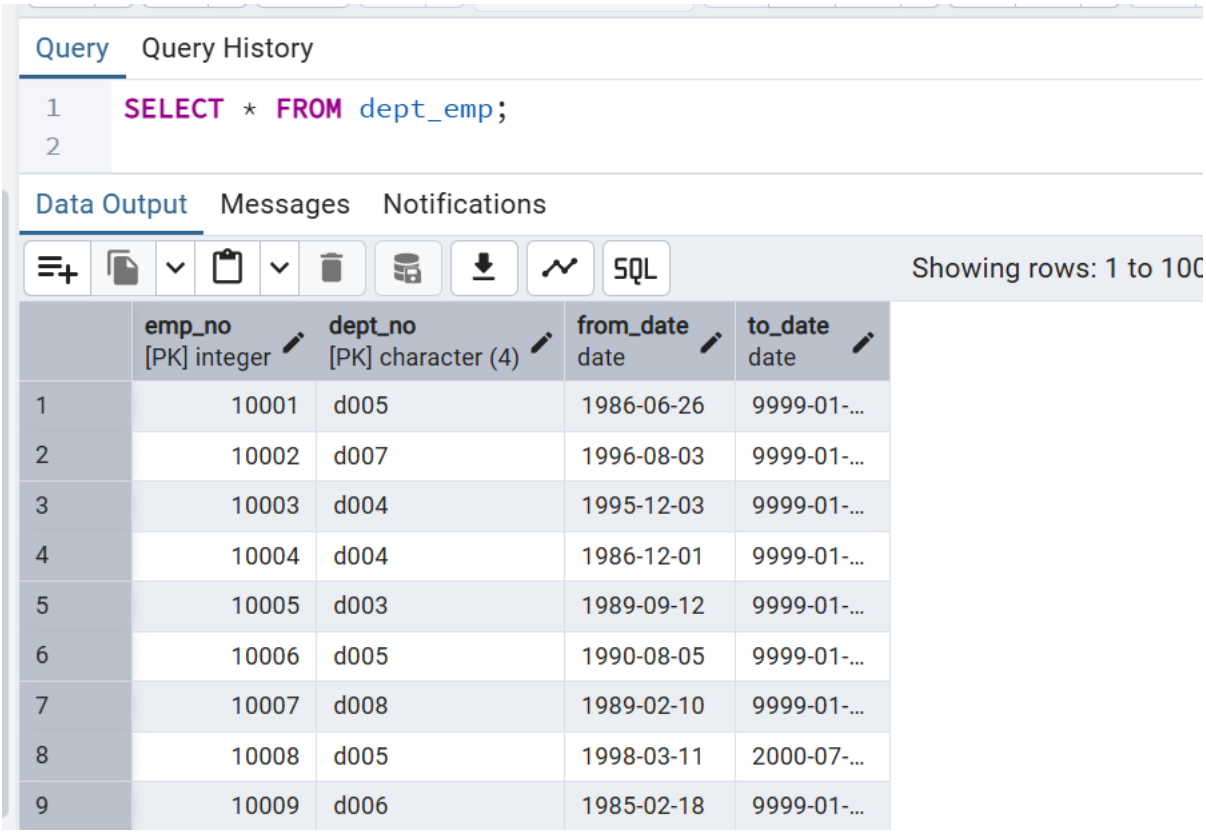
Table Employée



```
SELECT * FROM employees;
```

	emp_no [PK] integer	birth_date date	first_name character varying (14)	last_name character varying (16)	gender gender	hire_date date
1	10001	1953-09-02	Georgi	Facello	M	1986-06-26
2	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
3	10003	1959-12-03	Parto	Bamford	M	1986-08-28
4	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
5	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
6	10006	1953-04-20	Anneke	Preusig	F	1989-06-02
7	10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10

Tables relation entre Employée et Département



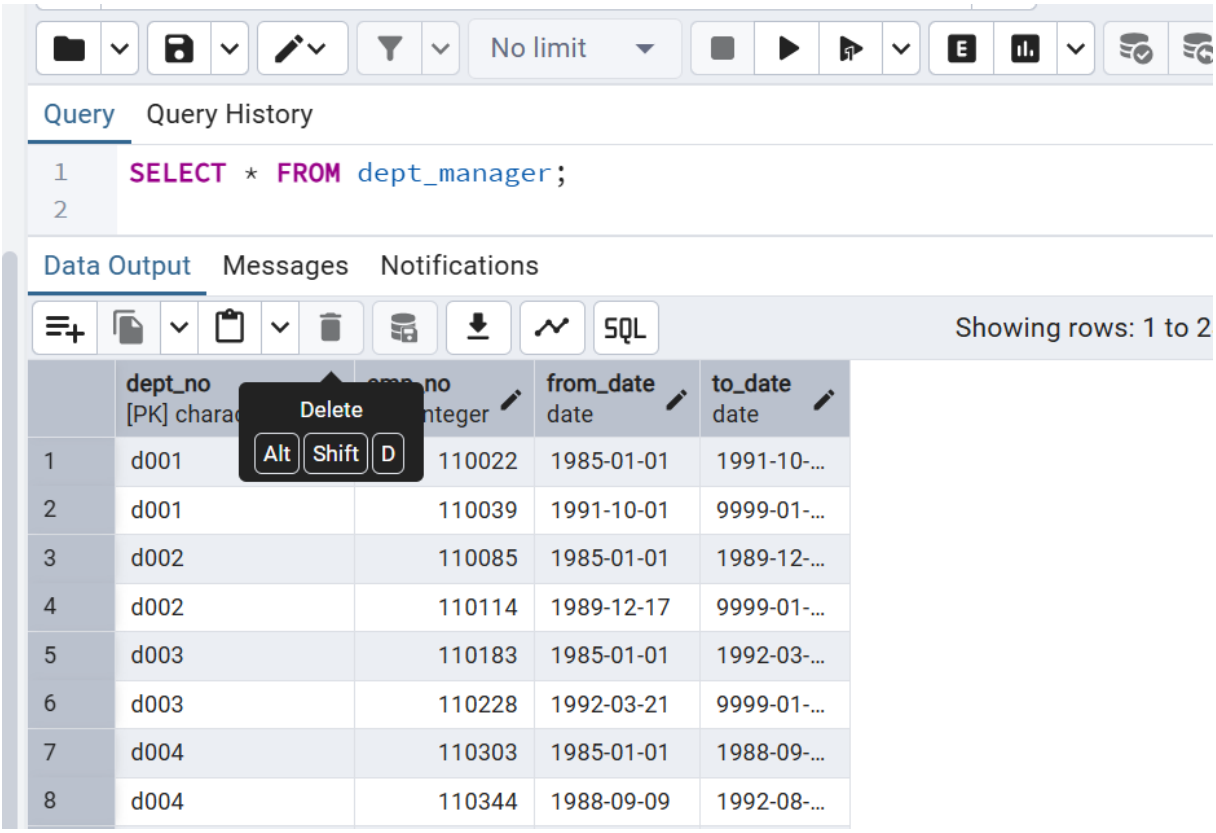
Query | Query History

```
1  SELECT * FROM dept_emp;
2
```

Data Output | Messages | Notifications

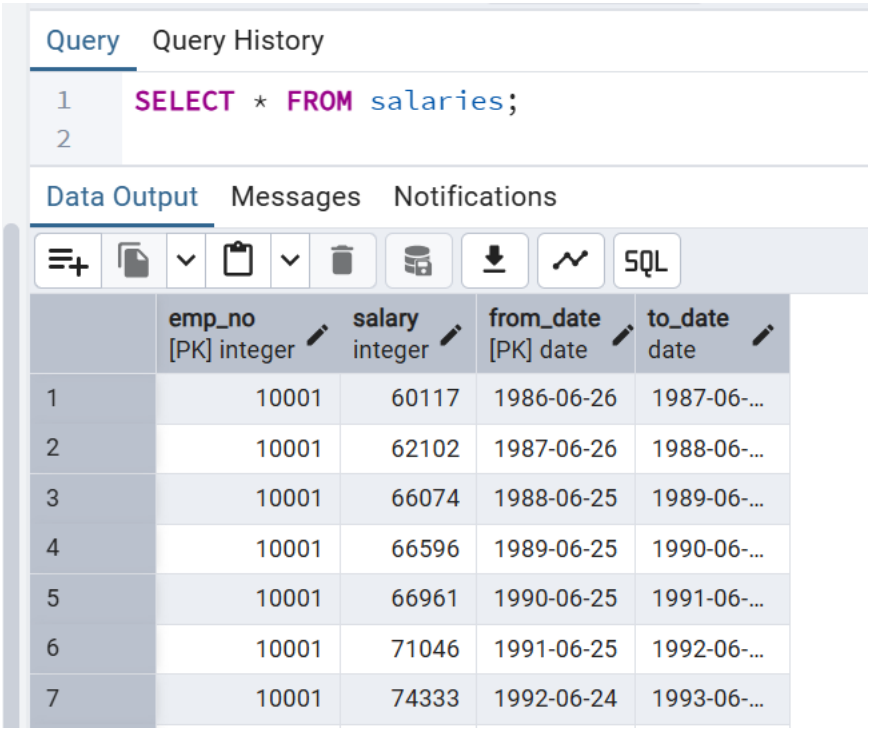
Showing rows: 1 to 100

	emp_no [PK] integer	dept_no [PK] character (4)	from_date date	to_date date
1	10001	d005	1986-06-26	9999-01-...
2	10002	d007	1996-08-03	9999-01-...
3	10003	d004	1995-12-03	9999-01-...
4	10004	d004	1986-12-01	9999-01-...
5	10005	d003	1989-09-12	9999-01-...
6	10006	d005	1990-08-05	9999-01-...
7	10007	d008	1989-02-10	9999-01-...
8	10008	d005	1998-03-11	2000-07-...
9	10009	d006	1985-02-18	9999-01-...



	dept_no [PK] charac	emp_no integer	from_date date	to_date date
1	d001	110022	1985-01-01	1991-10-...
2	d001	110039	1991-10-01	9999-01-...
3	d002	110085	1985-01-01	1989-12-...
4	d002	110114	1989-12-17	9999-01-...
5	d003	110183	1985-01-01	1992-03-...
6	d003	110228	1992-03-21	9999-01-...
7	d004	110303	1985-01-01	1988-09-...
8	d004	110344	1988-09-09	1992-08-...

Table Salarie



	emp_no [PK] integer	salary integer	from_date [PK] date	to_date date
1	10001	60117	1986-06-26	1987-06-...
2	10001	62102	1987-06-26	1988-06-...
3	10001	66074	1988-06-25	1989-06-...
4	10001	66596	1989-06-25	1990-06-...
5	10001	66961	1990-06-25	1991-06-...
6	10001	71046	1991-06-25	1992-06-...
7	10001	74333	1992-06-24	1993-06-...

Query Query History

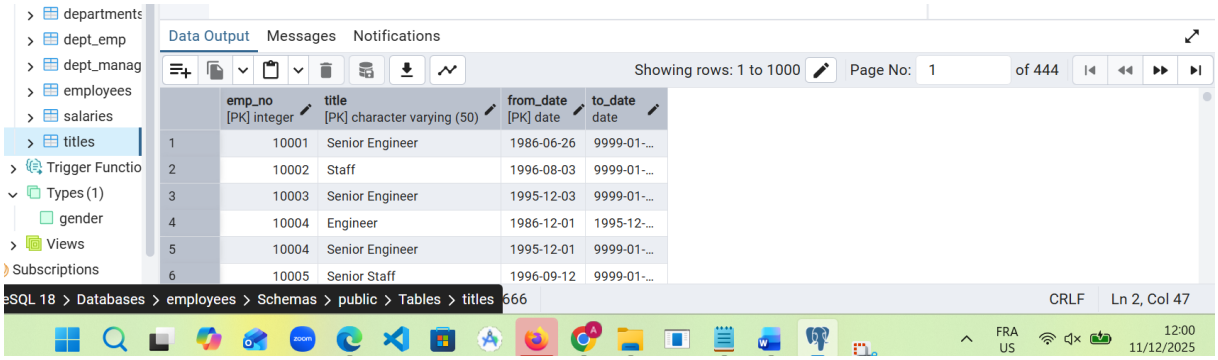
```
1  SELECT * FROM titles;
2
```

Data Output Messages Notifications

Showing rows: 1 t

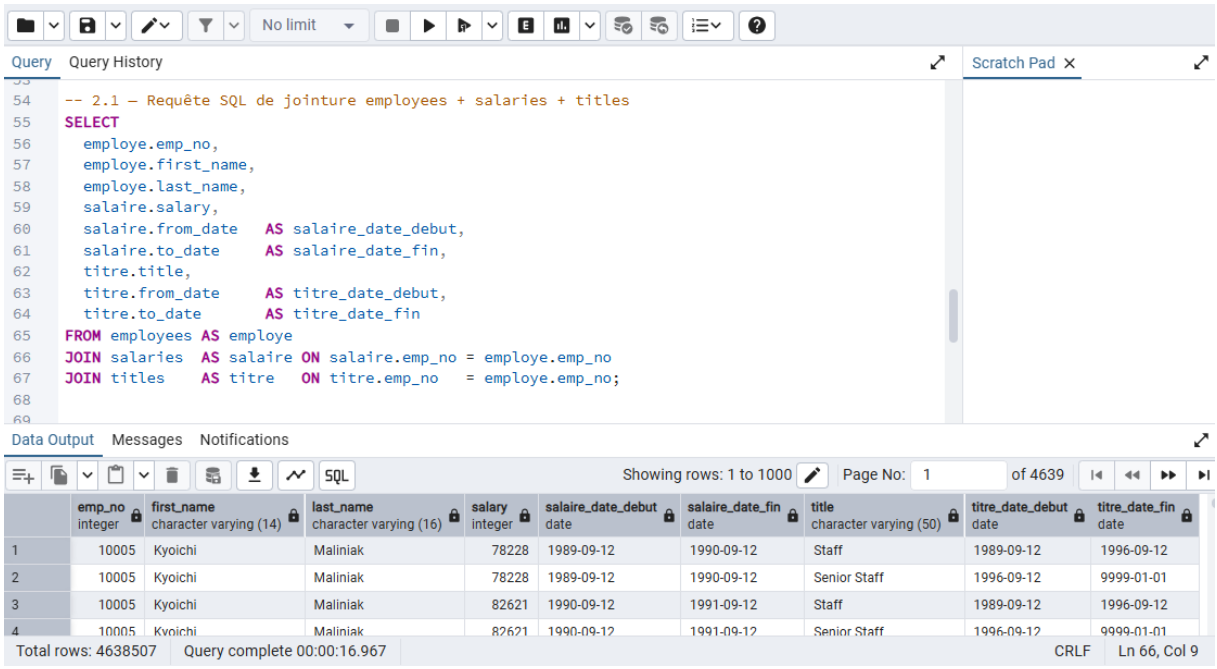
	emp_no [PK] integer	title [PK] character varying (50)	from_date [PK] date	to_date date
1	10001	Senior Engineer	1986-06-26	9999-01-...
2	10002	Staff	1996-08-03	9999-01-...
3	10003	Senior Engineer	1995-12-03	9999-01-...
4	10004	Engineer	1986-12-01	1995-12-...
5	10004	Senior Engineer	1995-12-01	9999-01-...
6	10005	Senior Staff	1996-09-12	9999-01-...

Table Title



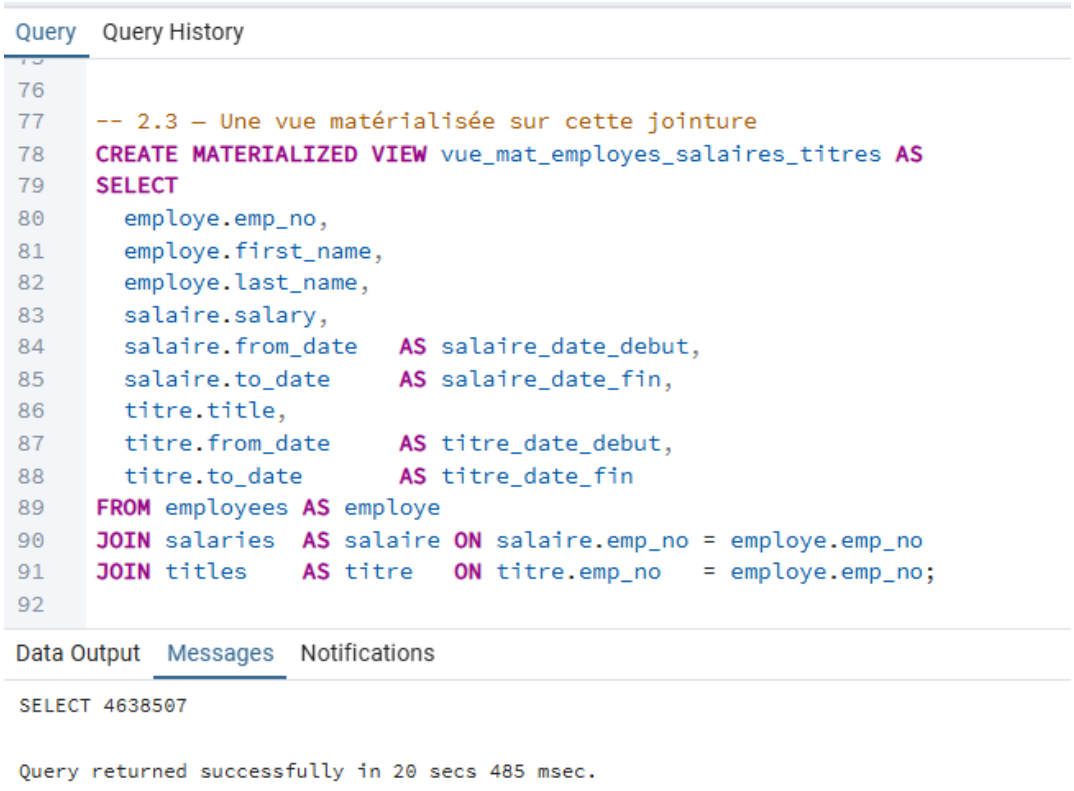
2. Cout des jointures en SQL

2.1. Requête SQL de jointure employees + salaries + titles

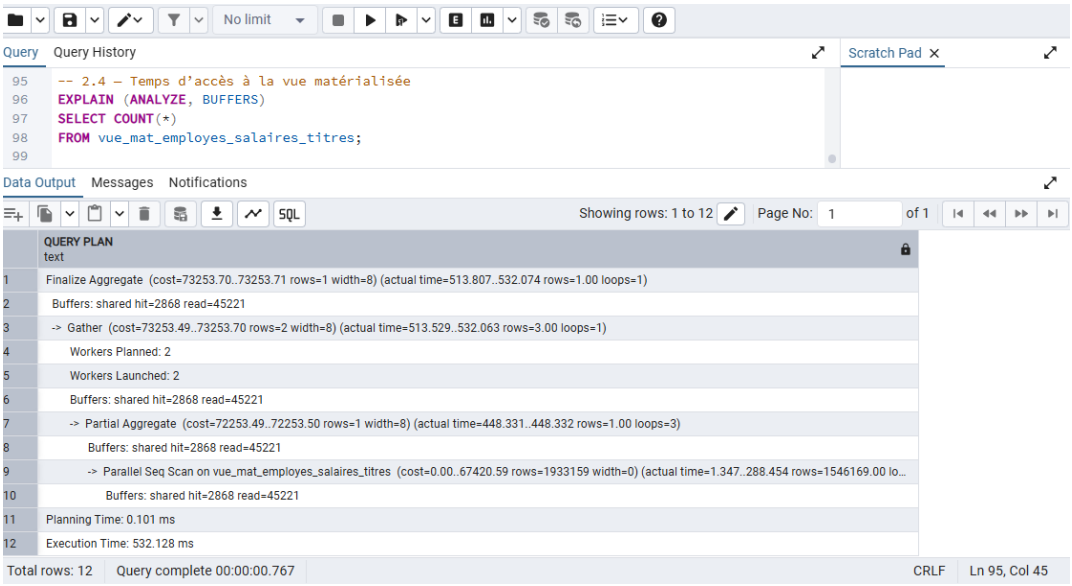


2.2. Total rows: 4638507 || 16 secs 967 msec

2.3. Une vue matérialisée



2.4. Temps d'accès à la vue matérialisée



Résultats chiffrés à reporter

Étape	Requête mesurée	Planning Time	Execution Time	Buffers (shared hit / read)
2.2	COUNT(*) sur la **jointure** employees ⋈ titles ⋈ salaries	**14.418 ms**	**1460.525 ms** (≈ 1.461 s)	**7610 / 13116**
2.4	COUNT(*) sur la **vue matérialisée** vue_mat_employes_salaires_titres	**0.101 ms**	**532.128 ms** (≈ 0.532 s)	**2868 / 45221**

3. Export de données PostgreSQL vers fichiers JSON

3.1. Sortie “1 document JSON par ligne” avec `row_to_json`

Query Query History

```sql
-- 3 Export de données PostgreSQL vers fichiers JSON
-- employees
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM employees ORDER BY emp_no) AS ligne;

-- departments
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM departments ORDER BY dept_no) AS ligne;

-- dept_emp
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM dept_emp ORDER BY emp_no, dept_no, from_date) AS ligne;

-- dept_manager
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM dept_manager ORDER BY dept_no, emp_no, from_date) AS ligne;

-- titles
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM titles ORDER BY emp_no, from_date, title) AS ligne;

-- salaries
SELECT row_to_json(ligne) AS document_json
FROM (SELECT * FROM salaries ORDER BY emp_no, from_date) AS ligne;
```

Data Output Messages Graph Visualiser Notifications

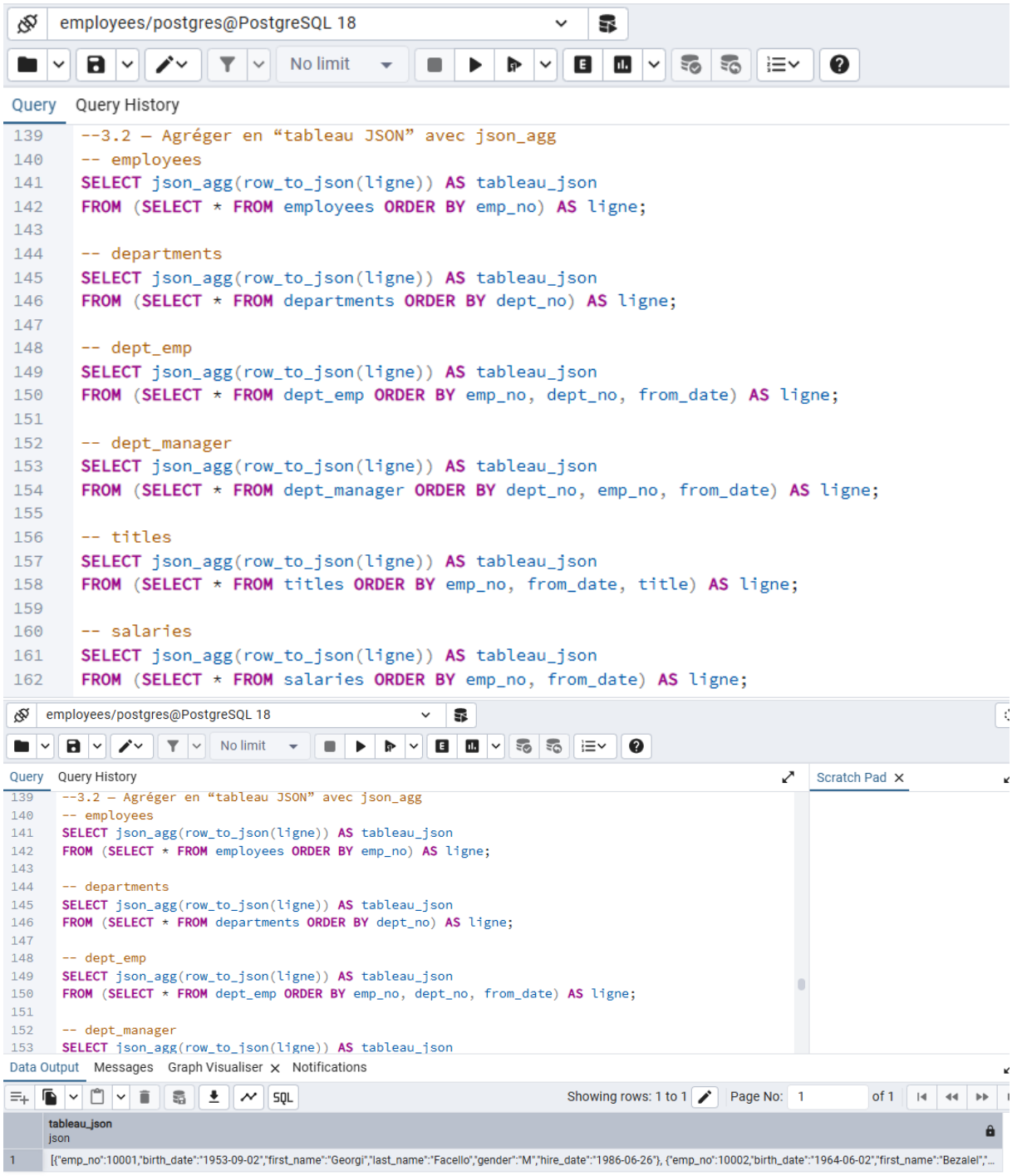
Total rows: 2844047 Query complete 00:00:04.864

Data Output Messages Graph Visualiser Notifications

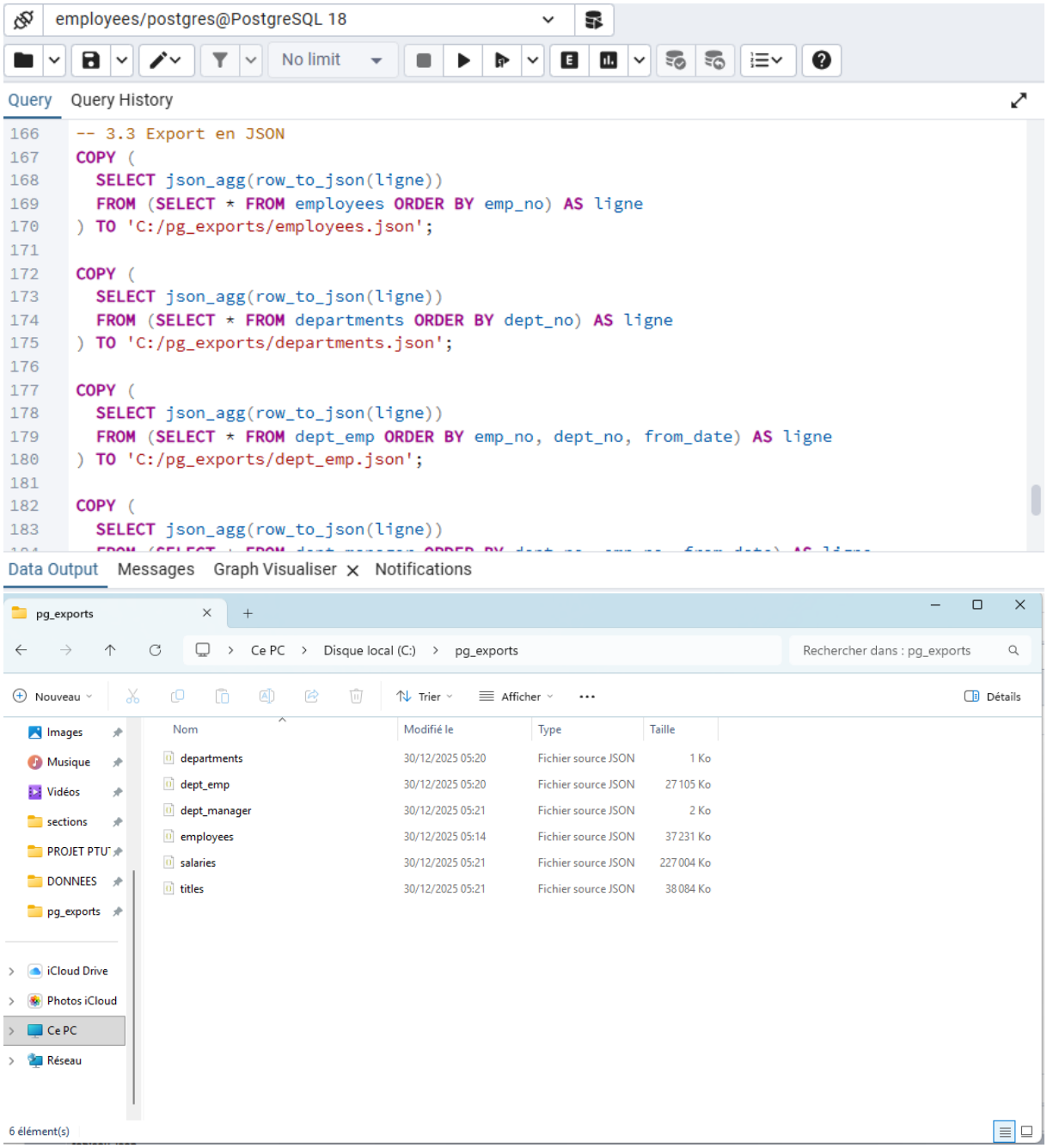
	document_json json
1	{"emp_no":10001,"salary":60117,"from_date":"1986-06-26","to_date":"1987-06-2...
2	{"emp_no":10001,"salary":62102,"from_date":"1987-06-26","to_date":"1988-06-2...
3	{"emp_no":10001,"salary":66074,"from_date":"1988-06-25","to_date":"1989-06-2...
4	{"emp_no":10001,"salary":66596,"from_date":"1989-06-25","to_date":"1990-06-2...
5	{"emp_no":10001,"salary":66961,"from_date":"1990-06-25","to_date":"1991-06-2...
6	{"emp_no":10001,"salary":71046,"from_date":"1991-06-25","to_date":"1992-06-2...
7	{"emp_no":10001,"salary":74333,"from_date":"1992-06-24","to_date":"1993-06-2...
8	{"emp_no":10001,"salary":75286,"from_date":"1993-06-24","to_date":"1994-06-2...
9	{"emp_no":10001,"salary":75994,"from_date":"1994-06-24","to_date":"1995-06-2...
10	{"emp_no":10001,"salary":76884,"from_date":"1995-06-24","to_date":"1996-06-2

Total rows: 2844047 Query complete 00:00:04.864

3.2. — Agréger en “tableau JSON” avec `json_agg`



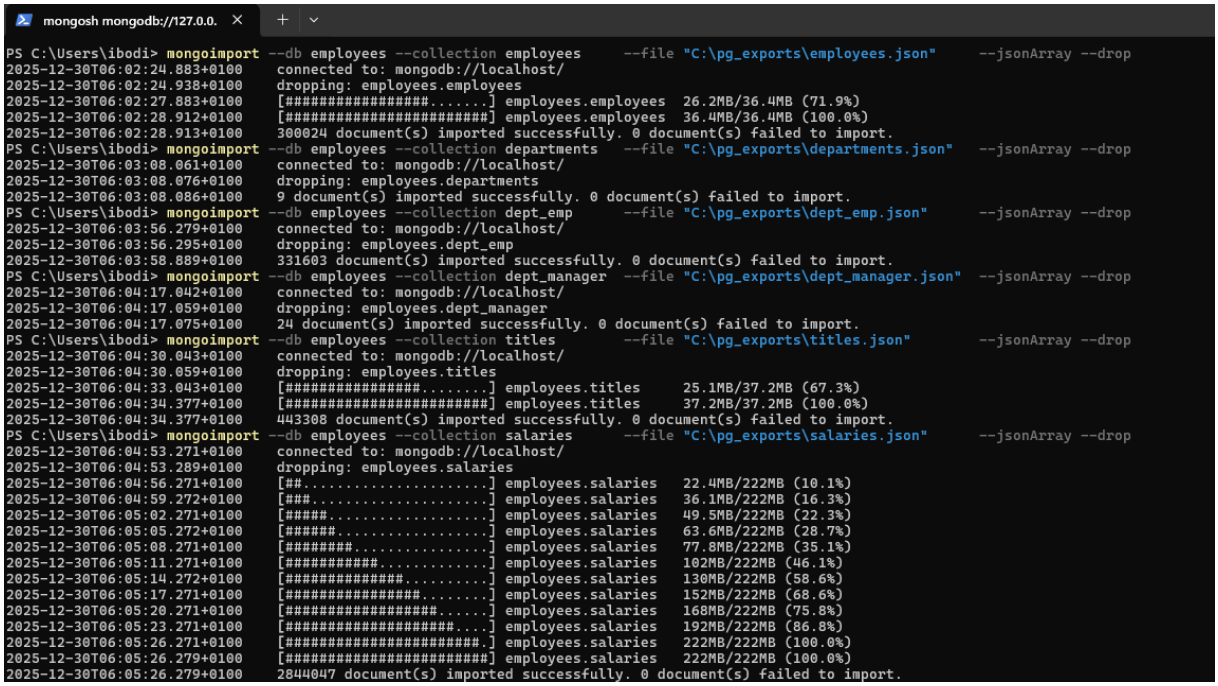
3.3. Export en JSON



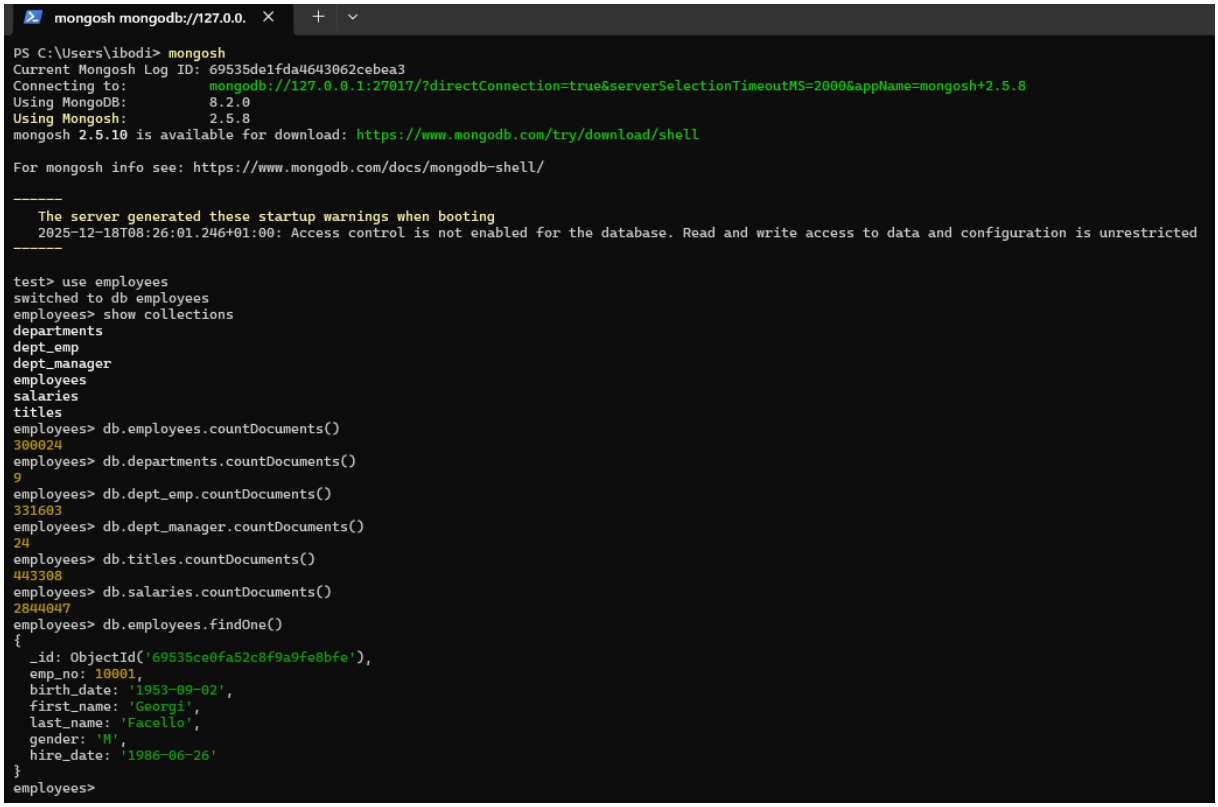
4. Import dans MongoDB des donn´ees des fichiers JSON

- Je lance mongod d’abord

4.1. Commandes mongoimport dans le CMD

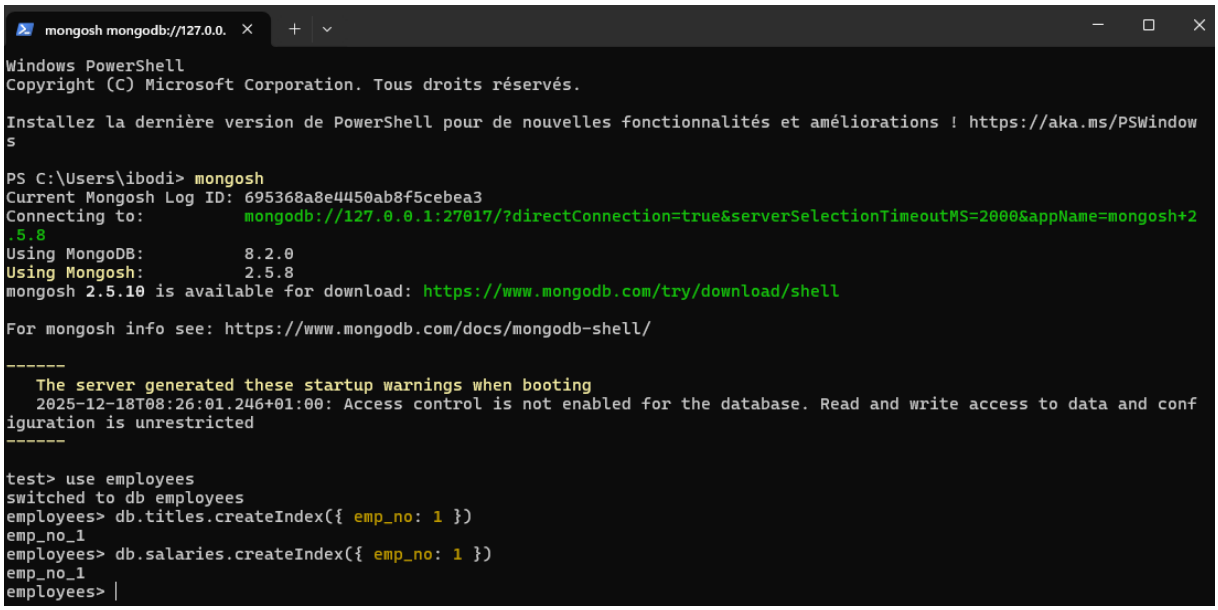


- Puis Je Vérifier que l'import est OK (avec mongosh)

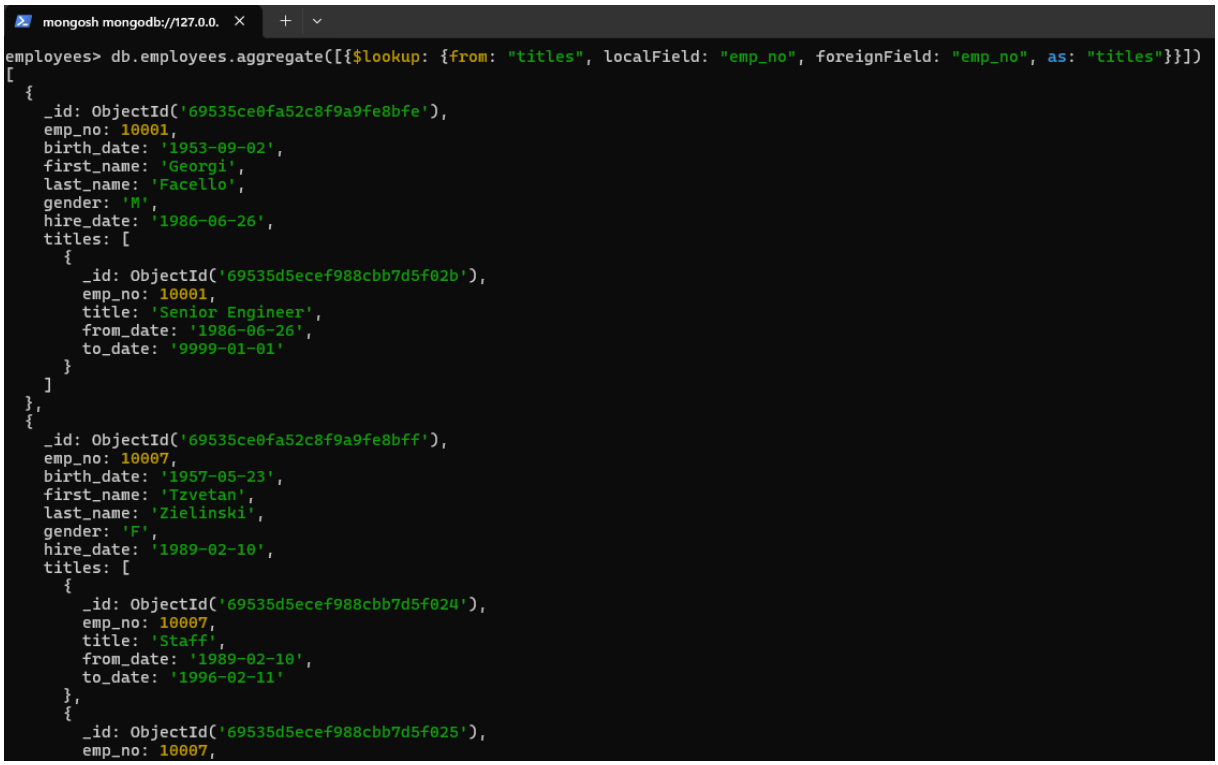


5. Import dans MongoDB des données des fichiers JSON

- 5.0 — Je vais créer des index pour accélérer les $lookup



5.1. Jointure employees ↔ titles (1 $lookup)



....

5.2. Pipeline à 2 étapes : jointure employees ↔ titles ↔ salaries

```
employees> const pipeline_jointure_3 = [{$lookup: {from: "titles", localField: "emp_no", foreignField: "emp_no", as: "titles"}},
...   {$lookup: {from: "salaries", localField: "emp_no", foreignField: "emp_no", as: "salaries"}}]

employees> db.employees.aggregate([{ $match: { emp_no: 10001 } }, ...pipeline_jointure_3]).toArray()
[
  {
    _id: ObjectId('69535ce0fa52c8f9a9fe8bfe'),
    emp_no: 10001,
    birth_date: '1953-09-02',
    first_name: 'Georgi',
    last_name: 'Facello',
    gender: 'M',
    hire_date: '1986-06-26',
    titles: [
      {
        _id: ObjectId('69535d5ecef988cbb7d5f02b'),
        emp_no: 10001,
        title: 'Senior Engineer',
        from_date: '1986-06-26',
        to_date: '9999-01-01'
      }
    ],
    salaries: [
      {
        _id: ObjectId('69535d75229d94d11c125a0f'),
        emp_no: 10001,
        salary: 60117,
        from_date: '1986-06-26',
```

...

5.3. Mesurer le temps d'exécution de la jointure (MongoDB)

```
mongosh mongodb://127.0.0.

employees> db.employees .explain("executionStats") .aggregate(pipeline_jointure_3, { allowDiskUse: true })
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'employees.employees',
    parsedQuery: {},
    indexFilterSet: false,
    queryHash: 'CDC0091B',
    planCacheShapeHash: 'CDC0091B',
    planCacheKey: 'BB9BAF92',
    optimizationTimeMillis: 1,
    optimizedPipeline: true,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      queryPlan: {
        stage: 'EQ_LOOKUP',
        planNodeId: 3,
        foreignCollection: 'employees.salaries',
        localField: 'emp_no',
        foreignField: 'emp_no',
        asField: 'salaries',
        strategy: 'IndexedLoopJoin',
        indexName: 'emp_no_1',
        indexKeyPattern: { emp_no: 1 },
        scanDirection: 'forward',
        inputStage: {
          stage: 'EQ_LOOKUP',
          planNodeId: 2,
          foreignCollection: 'employees.titles',
          localField: 'emp_no',
          foreignField: 'emp_no',
          asField: 'titles',
          strategy: 'IndexedLoopJoin',
          indexName: 'emp_no_1',
          indexKeyPattern: { emp_no: 1 },
          scanDirection: 'forward',
          inputStage: {
            stage: 'COLLSCAN',
            planNodeId: 1,
            filter: {},
```

.....

5.4. Dénormalisation + $project (supprimer doublons emp_no et _id)

```
employees> let pipeline_denormalisation = [
...   ...pipeline_jointure_3, {$project: { _id: "$emp_no", birth_date: 1, first_name: 1, last_name: 1,
...       gender: 1, hire_date: 1,
...       titles: {$map: {input: "$titles", as: "t", in: {title: "$$t.title", from_date: "$$t.from_date",
...             to_date: "$$t.to_date"}}},
...       salaries: {$map: {input: "$salaries", as: "s", in: {salary: "$$s.salary",
...             from_date: "$$s.from_date", to_date: "$$s.to_date"}}}}}];

employees> db.employees.aggregate([{ $match: { emp_no: 10001 } }, ...pipeline_denormalisation], { allowDiskUse: true }).toArray();
[
  {
    birth_date: '1953-09-02',
    first_name: 'Georgi',
    last_name: 'Facello',
    gender: 'M',
    hire_date: '1986-06-26',
    _id: 10001,
    titles: [
      {
        title: 'Senior Engineer',
        from_date: '1986-06-26',
        to_date: '9999-01-01'
      }
    ],
    salaries: [
      { salary: 60117, from_date: '1986-06-26', to_date: '1987-06-26' },
      { salary: 75994, from_date: '1994-06-24', to_date: '1995-06-24' },
      { salary: 76884, from_date: '1995-06-24', to_date: '1996-06-23' },
      { salary: 80013, from_date: '1996-06-23', to_date: '1997-06-23' },
      { salary: 81025, from_date: '1997-06-23', to_date: '1998-06-23' },
      { salary: 81097, from_date: '1998-06-23', to_date: '1999-06-23' },
      { salary: 62102, from_date: '1987-06-26', to_date: '1988-06-25' },
      { salary: 84917, from_date: '1999-06-23', to_date: '2000-06-22' },
      { salary: 85097, from_date: '2001-06-22', to_date: '2002-06-22' },
      { salary: 85112, from_date: '2000-06-22', to_date: '2001-06-22' },
```

5.5. Sauvegarder le résultat dans une nouvelle collection (`$merge` ou `$out`)

```
employees> console.time("creation_employees_denormalises");

employees> db.employees.aggregate([
...     ...pipeline_denormalisation,
...     {$merge: {into: "employees_denormalises", on: "_id", whenMatched: "replace", whenNotMatched: "insert"}}],
...   { allowDiskUse: true }).toArray();
[]
employees> console.timeEnd("creation_employees_denormalises");
%s: %s creation_employees_denormalises 2:13.200 (m:ss.mmm)

employees>
```

5.6. Mesurer le délai d'accès "complet" via la nouvelle collection

```
employees> db.employees_denormalises.explain("executionStats").find({ _id: 10001 }).limit(1);
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'employees.employees_denormalises',
    parsedQuery: { _id: { '$eq': 10001 } },
    indexFilterSet: false,
    queryHash: 'B774D27B',
    planCacheShapeHash: 'B774D27B',
    planCacheKey: '7635556C',
    optimizationTimeMillis: 0,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      stage: 'EXPRESS_IXSCAN',
      keyPattern: '{ _id: 1 }',
      indexName: '_id_'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 0,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      isCached: false,
      stage: 'EXPRESS_IXSCAN',
      keyPattern: '{ _id: 1 }',
      indexName: '_id_',
      keysExamined: 1,
```

6. Conclusions

La dénormalisation proposée (un document employee contenant deux tableaux titles et salaries) est opportune lorsque ces informations sont quasi systématiquement lues ensemble, car elle évite des `$lookup` fréquents et améliore les performances de lecture.

Elle est particulièrement adaptée aux charges read-heavy avec peu de mises à jour sur les données dupliquées (sinon la maintenance et la cohérence deviennent coûteuses).

Enfin, elle reste pertinente tant que le document final (employé + tableaux) reste d'une taille raisonnable et ne s'approche pas de la limite de 16 MiB par document BSON ; au-delà, un modèle par références est préférable.