Universidade do Minho
Mestrado em Engenharia Informática

Bases de Dados noSQL

Trabalho Prático

PG54708, Sérgio Ribeiro
PG50633, Mariana Marques
PG54780, Eduardo Henriques
PG50414, Henrique Alvelos

17 de junho de 2024

Conteúdo

1 Introdução

Presentemente existem várias bases de dados à nossa disposição com diferentes métodos e regras de operação, traduzindo-se isto em casos de uso diferenciados para situações e contextos específicos. A evolução das bases de dados relacionais elevou a necessidade de tal acontecer, pois estas podem não ser aplicáveis de modo eficiente para todas as aplicações ou objetivos, dependendo das características do problema a atacar.

Neste documento iremos relatar a nossa análise de uma base de dados relacional que contém dados da informação utilizada no funcionamento de um hospital, e migrar os seus dados para uma base de dados orientada a documentos(MongoDB) assim como uma orientada a grafos(Neo4J), e adaptar a informação transferida para estas de modo a tirar proveito das vantagens destes tipos de bases de dados. Para tal criaremos contextos no qual as bases de dados irão ser utilizadas e também criamos um conjunto de pesquisas para cada tipo de base de dados, assim como adaptar os dados armazenados na base de dados relacional para tirarem partido das vantagens destes novos paradigmas.

Finalmente, iremos efetuar uma análise crítica do nosso trabalho, indicando os aspetos mais e menos vantajosos de cada tipo de base de dados para obter uma visão mais aprofundada as situações onde devem ser utilizadas.

2 Contexto

Para analisar a base de dados relacional e adaptar os seus dados para outros paradigmas é estritamente necessário aplicar um contexto ao nosso projeto, pois a natureza das operações de manipulação das bases de dados variam de acordo com o objetivo em mente.

No nosso caso, imaginamos que as bases de dados serão utilizadas para obter informação que será usada para criação de estatísticas sobre o funcionamento do hospital(contas, medicamentos, marcação de consultas, utilização de determinados serviços, etc.). O objetivo final será então utilizar a informação para adaptar o hospital às necessidades dos pacientes, alocando o seu orçamento e funcionários de acordo com as *insights* obtidas.

Algumas das funções a implementar(triggers, proccedures) estarão mais profundamente interligadas com a administração e modificação da informação das bases de dados do que o contexto que lhes foram atribuídas devido à natureza das operações.

3 Exploração da Base de Dados

A partir do modelo lógico da base de dados, conseguimos perceber que existem 3 divisões que se interligam. Primeiramente, temos o lado do paciente, que contém os seus dados pessoais e o seu histórico no hospital. A seguir, temos o lado do hospital, que contém todas as hospitalizações, análises e consultas. No fim, temos a parte administrativa do hospital, com os dados de todos os

funcionários do hospital e a que departamento pertencem. Assim, conseguimos pesquisar, por exemplo, todos os dados de uma consulta: a quem foi dada e por quem.

Já no ficheiro SQL é possível encontrar a criação de todas as tabelas, dados, triggers, procedimentos e views. Só existe um trigger que atua após a atualização da data de alta num registo de hospitalização. A sua função é calcular os custos associados à hospitalização e gerar uma fatura. Primeiro, o trigger declara quatro variáveis para armazenar os custos do quarto, dos testes, de outros encargos e o custo total. Após calcular estes custos, ele insere uma nova fatura na tabela **bill** com os valores calculados. O estado do pagamento (inicialmente definido como **PENDING**) e a data de registo. Assim, o trigger garante que, ao registar a alta de um paciente, uma fatura correspondente seja gerada automaticamente, refletindo todos os custos associados à hospitalização. O único procedimento chamado **sp_update_bill_status** foi criado para atualizar o estado do pagamento de uma fatura com base no valor pago. Ele recebe dois parâmetros: o identificador da fatura (**p_bill_id**) e o valor pago (**p_paid_value**). Primeiro, o procedimento recupera o valor total da fatura e compara o valor pago com o valor total da fatura. Se o valor pago for menor que o valor total, o procedimento atualiza o estado da fatura para **FAILURE** e lança um erro a informar que o valor pago é inferior ao valor devido. Caso contrário, se o valor pago for igual ou maior que o valor total, o estado da fatura é atualizado para **PROCESSED**. Esse procedimento garante que o estado da fatura reflete corretamente se o pagamento foi suficiente ou não para cobrir o valor total devido. A view PatientAppointmentView é uma consulta que reúne informações sobre as consultas médicas de pacientes de várias tabelas numa única visualização. Ela mostra detalhes importantes, como a data e hora da consulta, informações do médico, o departamento em que o médico trabalha, e os dados do paciente.

4 Queries SQL

Neste tema iremos apresentar um conjunto de *queries* desenvolvidas para extrair informações da base de dados que visam responder certas questões para o gerenciamento do hospital. Neste contexto, a análise dos dados desempenha um papel na melhoria da qualidade e gestão de recursos.

- **Medicamentos Mais Prescritos**: Esta *query* ajuda a identificar que medicamentos são prescritos mais frequentemente, auxiliando na gestão de stock para garantir que os medicamentos mais regulares estejam disponíveis. Como resultado obtivemos a lista de todos os medicamentos por ordem decrescente de quantidade de vezes que são prescritos e obtivemos paracetamol em primeiro lugar, sendo prescrito 200 vezes, seguido de ibuprofen e amoxicillin. Por último obtivemos amlodipine com apenas 9 prescrições.

```sql
SELECT
    MEDICINE.M_NAME,
    COUNT(MEDICINE.IDMEDICINE)
    AS PRESCRIPTION_COUNT
FROM
    MEDICINE
JOIN
    PRESCRIPTION
ON
    PRESCRIPTION.IDMEDICINE = MEDICINE.IDMEDICINE
GROUP BY
    MEDICINE.M_NAME
ORDER BY
    PRESCRIPTION_COUNT DESC;
```

Figura 1: Query Medicamentos Mais Precistos

M_NAME	PRESCRIPTION_COUNT
Paracetamol	200
Ibuprofen	186
Amoxicillin	164
Ciprofloxacin	148
Lisinopril	129
Atorvastatin	107
Metformin	75
Levothyroxine	46
Simvastatin	24
Amlodipine	9

Figura 2: Resultado Query Medicamentos Mais Precistos

- **Quartos mais utilizados**: A importância desta *query* é monitorizar a utilização dos quartos para uma gestão mais eficiente dos mesmos. Obtivemos assim uma lista decrescente de todos os quartos pelo número de vezes utilizados. Os resultados mostram que os quartos com o *id* 37, 38, 39 e 40 apresentam o maior uso, sendo utilizados 7 vezes.

```sql
SELECT
    HOSPITALIZATION.ROOM_IDROOM,
    ROOM.ROOM_COST,
    ROOM.ROOM_TYPE,
    COUNT(HOSPITALIZATION.ROOM_IDROOM) AS TIMES_USED
FROM
    ROOM
JOIN
    HOSPITALIZATION
ON
    HOSPITALIZATION.ROOM_IDROOM = ROOM.IDROOM
GROUP BY
    HOSPITALIZATION.ROOM_IDROOM, ROOM.ROOM_COST, ROOM.ROOM_TYPE
ORDER BY
    TIMES USED DESC;
```

Figura 3: Query Quartos Mais Utilizados

ROOM_IDROOM	ROOM_COST	ROOM_TYPE	TIMES_USED
39	550	Penthouse	7
38	200	Family	7
37	80	Economy	7
40	420	Executive	7
41	100	Single	6
42	150	Double	5
10	400	Executive	3
9	500	Penthouse	3
8	180	Family	3
6	300	VIP	3
7	70	Economy	2

...

Figura 4: Resultado Query Quartos Mais Utilizados

- **Médicos com Mais Consultas**: Identifica os médicos mais ativos, ajudando na gestão de carga de trabalho. Como resultado obtivemos Emily Rowe com 7 compromissos, estando a cima dos restantes.

```
SELECT
    STAFF.EMP_FNAME || ' ' || STAFF.EMP_LNAME AS FULL_NAME,
    DOCTOR.QUALIFICATIONS,
    DEPARTMENT.DEPT_NAME,
    COUNT(APPOINTMENT.IDDOCTOR) AS APPOINTMENTS
FROM
    APPOINTMENT
JOIN
    DOCTOR
ON
    APPOINTMENT.IDDOCTOR = DOCTOR.EMP_ID
JOIN
    STAFF
ON
    DOCTOR.EMP_ID = STAFF.EMP_ID
JOIN
    DEPARTMENT
ON
    DEPARTMENT.IDDEPARTMENT = STAFF.IDDEPARTMENT
GROUP BY
    STAFF.EMP_FNAME || ' ' || STAFF.EMP_LNAME,
    DOCTOR.QUALIFICATIONS,
    DEPARTMENT.DEPT_NAME
ORDER BY
    APPOINTMENTS DESC;
```

Figura 5: Query Médicos com Mais Consultas

FULL_NAME	QUALIFICATIONS	DEPT_NAME	APPOINTMENTS
Emily Rowe	MD	Ophthalmology	7
Brittany Collins	PhD	Emergency_1	6
Sarah Boyd	MD	Rheumatology	6
Rebecca Reyes	MD	Pulmonology	6
Ashley Jones	MD	Urology	6
Kathryn Perez	MD	Emergency_2	6
Christina Dalton	MD	Radiology	5
Michael Peterson	PhD	Cardiology_4	5
Christopher Martinez	MD	Surgery	5
Noah Terry	MD	Cardiology_1	5
Lee Collins	PhD	Ophthalmology	5
...			

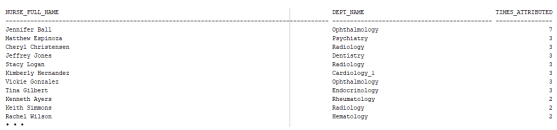
Figura 6: Resultado Query Médicos com Mais Consultas

- **Enfermeiras Mais Atribuídas**: Mostra a quantidade de pacientes que cada enfermeira está responsável de forma decrescente. Obtivemos a enfermeira Jennifer Ball responsável por 7 pacientes. É importante notar

que as enfermeiras com mais pacientes a seguir de Jennifer apenas são responsáveis por 3 pacientes.

```
SELECT
    STAFF.EMP_FNAME || ' ' || STAFF.EMP_LNAME AS NURSE_FULL_NAME,
    DEPARTMENT.DEPT_NAME,
    COUNT(HOSPITALIZATION.RESPONSIBLE_NURSE) AS TIMES_ATTRIBUTED
FROM
    HOSPITALIZATION
JOIN
    NURSE
ON
    HOSPITALIZATION.RESPONSIBLE_NURSE= NURSE.STAFF_EMP_ID
JOIN
    STAFF
ON
    NURSE.STAFF_EMP_ID = STAFF.EMP_ID
JOIN
    DEPARTMENT
ON
    DEPARTMENT.IDDEPARTMENT = STAFF.IDDEPARTMENT
GROUP BY
    STAFF.EMP_FNAME || ' ' || STAFF.EMP_LNAME,
    DEPARTMENT.DEPT_NAME
ORDER BY
    TIMES ATTRIBUTED DESC;
```

Figura 7: Query Enfermeiras Mais Atribuídas



NURSE_FULL_NAME	DEPT_NAME	TIMES_ATTRIBUTED
Jennifer Ball	Ophthalmology	7
Matthew Espinoza	Psychiatry	3
Cheryl Christensen	Radiology	3
Jeffrey Jones	Dentistry	3
Stacy Logan	Radiology	3
Kimberly Hernandez	Cardiology_1	3
Vickie Gonzalez	Ophthalmology	3
Tina Gilbert	Endocrinology	3
Kenneth Ayers	Rheumatology	2
Keith Simmons	Radiology	2
Rachel Wilson	Hematology	2
...		

Figura 8: Resultado Query Enfermeiras Mais Atribuídas

- **10 Pacientes com Mais Episódios**: Apresenta os pacientes com mais episódios ajudando a identificar quais precisam de maior atenção e recursos. Apenas Alexander Ho apresenta 2 episódios, sendo que os restantes pacientes apresentam apenas 1.

```
SELECT
    PATIENT.PATIENT_FNAME || ' ' || PATIENT.PATIENT_LNAME AS FULL_NAME,
    COUNT(EPISODE.IDEPISODE) AS EPISODES
FROM
    PATIENT
JOIN
    EPISODE
ON
    EPISODE.IDEPISODE=PATIENT.IDPATIENT
GROUP BY
    PATIENT.PATIENT_FNAME || ' ' || PATIENT.PATIENT_LNAME
ORDER BY
    EPISODES DESC
FETCH FIRST 10 ROWS ONLY;
```

Figura 9: Query 10 Pacientes com Mais Episódios

FULL_NAME	EPISODES
Alexander Ho	2
Michael Johnson	1
Emily Brown	1
Emma Perez	1
Sophia Garcia	1
James Lopez	1
Olivia Lee	1
Benjamin Gonzalez	1
John Doe	1
William Martinez	1

Figura 10: Resultado Query 10 Pacientes com Mais Episódios

- **20 Pacientes com Maiores Gastos**: Analisa os 20 pacientes com maiores custos no hospital. Os resultados apresentam que Noah Vo é o paciente com maiores despesas, sendo estas de 12020 euros onde 540 são de custos de hospitalização.

```
SELECT
    PATIENT.PATIENT_FNAME || ' ' || PATIENT.PATIENT_LNAME AS  FULL_NAME,
    COUNT(EPISODE.IDEPISODE) AS NUM_EPISODIOS,
    SUM(BILL.ROOM_COST) AS CUSTOS_HOSPITALIZACAO,
    SUM(BILL.TEST_COST) AS CUSTOS_ANALISES,
    SUM(BILL.OTHER_CHARGES) AS OUTROS_CUSTOS,
    SUM(BILL.TOTAL) AS CUSTOS_TOTAIS
FROM
    PATIENT
JOIN
    EPISODE
ON
    PATIENT.IDPATIENT = EPISODE.PATIENT_IDPATIENT
JOIN
    BILL
ON
    BILL.IDEPISODE = EPISODE.IDEPISODE
GROUP BY
    PATIENT.PATIENT_FNAME || ' ' || PATIENT.PATIENT_LNAME
ORDER BY
    CUSTOS_TOTAIS DESC
FETCH FIRST 20 ROWS ONLY;
```

Figura 11: Query 20 Pacientes com Maiores Gastos

FULL_NAME	NUM_EPISODIOS	CUSTOS_HOSPITALIZACAO	CUSTOS_ANALISES	OUTROS_CUSTOS	CUSTOS_TOTAIS
Noah Vo	2	540	0	11480	12020
Michael Dang	1	300	0	8975	9275
William Bui	1	200	0	9465	9665
James Kim	1	500	0	7070	7570
Benjamin Gonzalez	1	200	0	7645	7845
Isabella Huynh	1	80	0	7900	7980
William Martinez	1	250	0	7100	7350
Mason Pham	1	420	0	6005	6425
Jacob Rodriguez	1	300	0	6030	6330
Elijah Truong	1	100	0	4600	4700
Sophia Garcia	1	80	0	4490	4570
...	...	...	...	...	...

Figura 12: Resultado Query 20 Pacientes com Maiores Gastos

- **Diagnósticos Mais Comuns de Pessoas que Já Foram Hospitalizadas**: Identifica os diagnósticos mais comuns de forma decrescente ajudando a focar em prevenções e tratamentos para estes. Como resultado obtivemos ansiedade, gripe comum, bronquite e úlcera estomacal como condições mais comuns e enxaqueca e gripe como condições menos comuns.

```
SELECT
    MEDICAL_HISTORY.CONDITION AS DIAGONÓSTICO,
    COUNT(HOSPITALIZATION.IDEPISODE) AS NUM_HOSPITALIZACOES
FROM
    PATIENT
JOIN
    MEDICAL_HISTORY
ON
    MEDICAL_HISTORY.IDPATIENT = PATIENT.IDPATIENT
JOIN
    EPISODE
ON
    PATIENT.IDPATIENT = EPISODE.PATIENT_IDPATIENT
JOIN
    HOSPITALIZATION
ON
    HOSPITALIZATION.IDEPISODE = EPISODE.IDEPISODE
GROUP BY
    MEDICAL_HISTORY.CONDITION
ORDER BY
    NUM_HOSPITALIZACOES DESC;
```

Figura 13: Query Diagnósticos Mais Comuns de Pessoas que Já Foram Hospitalizadas

DIAGONÓSTICO	NUM_HOSPITALIZACOES
Anxiety	5
Common Cold	5
Bronchitis	5
Stomach Ulcer	5
Asthma	3
Broken Arm	3
Osteoporosis	3
Concussion	3
Headache	3
Rheumatoid Arthritis	3
Insomnia	3
• • •	

Figura 14: Resultado Query Diagnósticos Mais Comuns de Pessoas que Já Foram Hospitalizadas

- **Seguros Mais Utilizados**: Apresenta os seguros mais utilizados de maneira decrescente. Observamos que o seguro XYZ é utilizado por 17 pacientes, o YZA por apenas 1 paciente e todos os restantes são utilizados por 9 pacientes.

```sql
SELECT
    COUNT(PATIENT.IDPATIENT) AS CLIENTES,
    INSURANCE.PROVIDER AS SEGURO,
    INSURANCE.COVERAGE AS COBERTURA
FROM
    PATIENT
JOIN
    INSURANCE
ON
    INSURANCE.POLICY_NUMBER = PATIENT.POLICY_NUMBER
GROUP BY
    INSURANCE.PROVIDER,
    INSURANCE.COVERAGE
ORDER BY
    CLIENTES DESC;
```

Figura 15: Query Seguros Mais Utilizados

CLIENTES	SEGURO	COBERTURA
17	XYZ Insurance	Partial Coverage
9	DEF Insurance	Limited Coverage
9	GHI Insurance	Full Coverage
9	ABC Insurance	Full Coverage
9	JKL Insurance	Partial Coverage
9	MNO Insurance	Limited Coverage
9	PQR Insurance	Full Coverage
9	STU Insurance	Partial Coverage
9	VWX Insurance	Full Coverage
1	YZA Insurance	Limited Coverage

Figura 16: Resultado Query Seguros Mais Utilizados

5 MongoDB

5.1 Coleções

Neste tópico iremos apresentar as coleções criadas no MongoDB, detalhando o trabalho realizado e as estratégias de desenvolvimento. Foram desenvolvidas 5 coleções através de código *python*, com dados provenientes de uma base de dados Oracle. Para fazer a ligação entre ambas as bases de dados usamos as bibliotecas *oracledb* e *pymongo*, permitindo assim uma integração eficaz e otimizada entre as duas bases de dados.

1. **Coleção Pacientes**: esta coleção armazena informações sobre os pacientes, incluindo os seus dados pessoais, histórico médico, contactos de emergência e informações sobre o seguro. Tem um total de 90 pacientes diferentes.
    - Dados Pessoais: inclui nome, tipo de sangue, nº de telemóvel, email, género, política de seguro e data de nascimento.
    - Histórico Médico: inclui informações sobre condições médicas e datas de diagnóstico.

- Contactos de Emergência: inclui os contactos de emergência associados a cada paciente.
- Informações de Seguro: inclui detalhes sobre o seguro de saúde, como o provedor, plano, copagamento e coberturas adicionais.

2. **Coleção Funcionários**: A coleção *staff* tem informações sobre todos os funcionários do hospital, incluindo médicos, enfermeiros, técnicos e detalhes do respetivo departamento.
   - Dados Básicos: inclui ID, nome, data de ingresso, estado ativo ou inativo, email e cargo.
   - Informações Específicas: os médicos possuem as suas qualificações listadas, enquanto que os restantes apenas são identificados pelos respetivos cargos.
   - Informações Departamentais: cada membro da equipa é associado ao seu departamento.
3. **Coleção Episódios**: abrange dados de consultas, receitas, hospitalizações e faturas associadas a episódios de tratamento de pacientes.
   - Consultas: inclui registos de consultas com a data, hora e médico responsável.
   - Receitas: detalhes sobre medicamentos prescritos, dosagens e datas de prescrição.
   - Hospitalizações: inclui dados sobre datas de admissão de altas, quarto do paciente e enfermeiro responsável.
   - Faturas: inclui os custos de quarto, testes e outros serviços, bem como o estado do pagamento.
4. **Medicamentos**: esta coleção armazena dados sobre os medicamentos disponíveis, incluindo o ID, nome, quantidade e custo.
5. **Quartos**: a coleção *rooms* contém detalhes sobre os quartos do hospital, como o ID, tipo e custo diário.

Estas coleções criadas no MongoDB oferecem uma estrutura organizada e eficiente para armazenar e recuperar dados de um sistema de saúde. As estratégias de desenvolvimento adotadas garantiram a integridade e a consistência dos dados, permitindo uma fácil visualização e manipulação das informações críticas para a operação do hospital.

5.2 *Queries*

Iremos agora apresentar as operações executadas na base de dados orientada a documentos que criamos, analisando *queries* que consideramos mais relevantes em maior detalhe. A grande maioria das *queries* gera resultados que possuem

maior relevância para os funcionários do hospital, visto que a informação obtida pode ser utilizada para otimizar a quantidade de pessoal disponível durante certos períodos de tempo, e de adaptar a quantidade de funcionários em cada área. Decidimos não colocar todas as *prints* das queries porque algumas são bastante longas, no entanto podem ser todas visualizadas na pasta da entrega.

1. **Episódios por mês:** Para identificar o período onde existe mais atividade no hospital, esta *querie* revela os meses mais ocupados da instituição. Neste caso, os meses de Dezembro e Outubro revelam um maior nível de atividade no hospital, e as suas funções deverão ser adaptadas para tal.

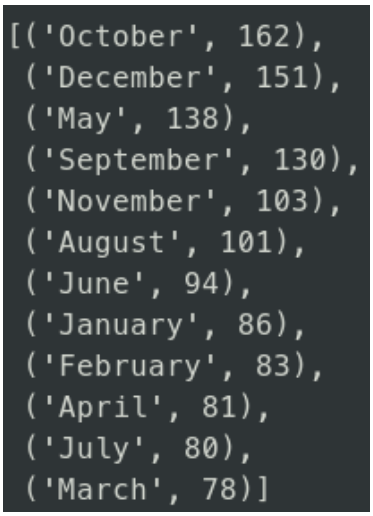


Figura 17: Resultado da query de episódios por mês

2. **Medicamentos mais comuns:** *Queries* disponíveis na base de dados SQL podem ser replicadas numa base de dados orientada a documentos com bastante facilidade. Os resultados de ambas são, como seria de esperar, iguais com a distinção de que esta versão apresenta os 3 medicamentos mais comuns. O hospital terá de ter maiores quantidades de *Paracetamol*, *Ibuprofen* e *Amoxicillin* pois estes são os mais receitados.

```python
def medicamento_mais_comum():
    db = setupConnection()
    result = db.episodes.aggregate([
        {"$unwind": "$prescriptions"},
        {
            "$group": {
                "_id": "$prescriptions.medicine_id",
                "count": { "$sum": 1 }
            }
        },
        { "$sort": { "count": -1 } },
        { "$limit": 3 },
        {
            "$lookup": {
                "from": "medicines",
                "localField": "_id",
                "foreignField": "id",
                "as": "medicine_details"
            }
        },
        {
            "$project": {
                "_id": 0,
                "medicine_id": "$_id",
                "count": 1,
                "medicine_details": { "$arrayElemAt": ["$medicine_details", 0] }
            }
        }
    ])
    return list(result)
```

Figura 18: Query Medicamentos mais comuns

```
[{'count': 200,
  'medicine_details': {'_id': ObjectId('666c66890ae95f7e2dcc96d1'),
                       'cost': 10.0,
                       'id': 1,
                       'name': 'Paracetamol',
                       'quantity': 50},
  'medicine_id': 1},
 {'count': 186,
  'medicine_details': {'_id': ObjectId('666c66890ae95f7e2dcc96d2'),
                       'cost': 15.0,
                       'id': 2,
                       'name': 'Ibuprofen',
                       'quantity': 30},
  'medicine_id': 2},
 {'count': 164,
  'medicine_details': {'_id': ObjectId('666c66890ae95f7e2dcc96d3'),
                       'cost': 20.0,
                       'id': 3,
                       'name': 'Amoxicillin',
                       'quantity': 20},
  'medicine_id': 3}]
Enter para continuar...
```

Figura 19: Resultado da query de medicamentos mais comuns

3. **Horas mais comuns para consultas:** De um ponto de vista logístico, faria mais sentido colocar mais funcionários(neste caso, doutores) a trabalhar durante horas de ponta, e é precisamente este o objetivo desta *querie*. As consultas são mais comuns no fim período da manhã e durante a tarde, entre as 11:00 e 12:00 horas, ou entre as 16:00 e as 18:00 horas.

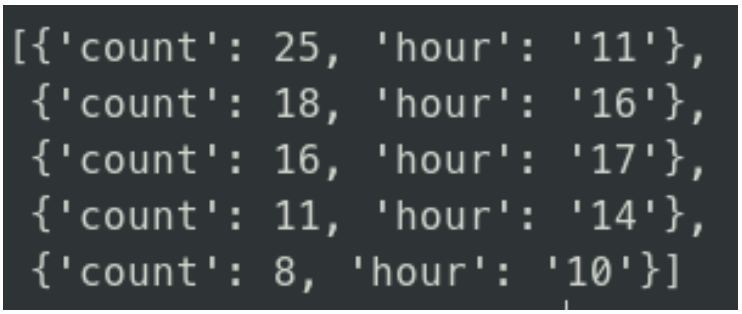


Figura 20: Resultado da query de horas mais comuns para consultas

4. **Condições mais comuns:** Num hospital as condições mais comuns devem ser verificadas antes de tentar diagnosticar algo mais específico. Logo descobrimos quais são os diagnósticos passados mais comuns para conseguir detetar o problema de saúde dos pacientes de maneira eficiente. Os diagnósticos mais comuns são Bronquite, Constipação e Artrite Reumatoide.

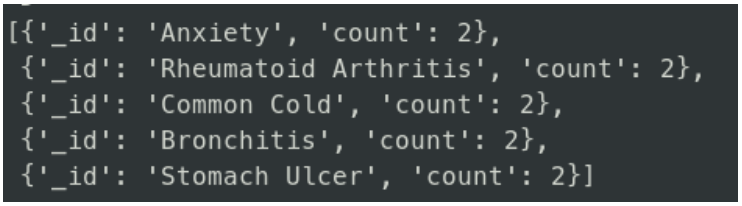


Figura 21: Resultado da query com os diagnósticos mais comuns

5. **Pacientes sem contas:** Uma das medidas para monitorizar o aspeto financeiro da instituição é verificar quais pacientes têm contas registadas em seu nome. À volta de 40 pacientes possuem contas pagas ou a pagar.

```
[{'name': 'Olivia Lee', 'patient_id': 8},
 {'name': 'Michael Duong', 'patient_id': 70},
 {'name': 'Avery Hoang', 'patient_id': 75},
 {'name': 'Oliver Truong', 'patient_id': 34},
 {'name': 'Olivia Gutierrez', 'patient_id': 24},
 {'name': 'Isabella Phan', 'patient_id': 32},
 {'name': 'Olivia Nguyen', 'patient_id': 51},
 {'name': 'Sophia Garcia', 'patient_id': 6},
 {'name': 'Natalie Trinh', 'patient_id': 49},
 {'name': 'Michael Johnson', 'patient_id': 3},
 {'name': 'Evelyn Ly', 'patient_id': 67},
 {'name': 'Isabella Hernandez', 'patient_id': 12},
 {'name': 'William Bui', 'patient_id': 76},
 {'name': 'Jacob Rodriguez', 'patient_id': 11},
 {'name': 'William Martinez', 'patient_id': 5},
 {'name': 'Lucas Ha', 'patient_id': 74},
 {'name': 'Emily Brown', 'patient_id': 4},
 {'name': 'Ethan Lopez', 'patient_id': 13},
 {'name': 'Amelia Huynh', 'patient_id': 30},
 {'name': 'Evelyn Bui', 'patient_id': 39},
 {'name': 'James Lopez', 'patient_id': 7},
 {'name': 'Jacob Lam', 'patient_id': 88},
 {'name': 'Harper Ho', 'patient_id': 63},
 {'name': 'Logan Trinh', 'patient_id': 78},
```


Figura 22: Resultado da query com os pacientes sem contas calculadas

6. **Pacientes com tipo de sangue específico:** Esta *query* lista os pacientes com um certo tipo de sangue, sendo uma informação vital em situações de emergência no caso de ser necessário doadores de sangue. O tipo de sangue a procurar é pedido pelo programa antes de executar a query.

```python
def pacientes_com_tipo_sangue(blood_type):
    db = setupConnection()
    result = db.patients.find({"blood_type": blood_type})
    return list(result)
```

Figura 23: Query Pacientes com tipo de sangue específico

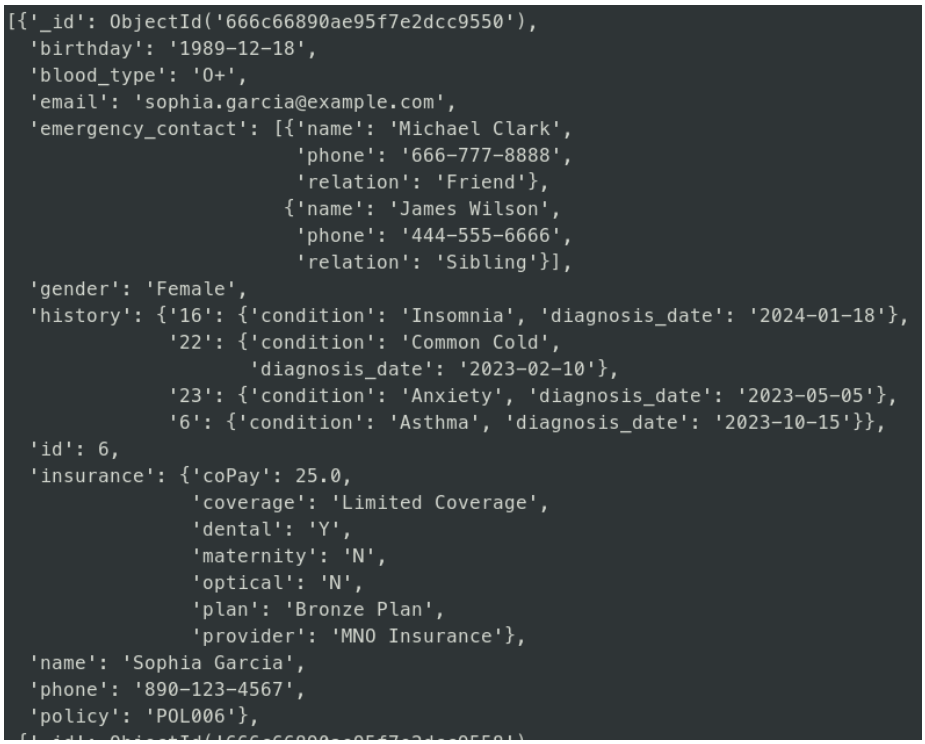


Figura 24: Resultado da query com os pacientes c/ um tipo de sangue específico, neste exemplo é O+

7. **Enfermeiras sem hospitalização:** Identifica quais enfermeiras não estão designadas a hospitalizações. Esta consulta ajuda a melhorar a distribuição de tarefas e a utilização adequada do *staff* de enfermeiras.

```python
def enfermeiras_sem_hospitalizacao():
    db = setupConnection()

    pipeline = [
        {
            "$unwind": "$hospitalizations"
        },
        {
            "$group": {
                "_id": None,
                "assigned_nurses": { "$addToSet": "$hospitalizations.nurse_id" }
            }
        }
        ]

    result = list(db.episodes.aggregate(pipeline))

    assigned_nurse_ids = result[0]['assigned_nurses'] if result else []
    unassigned_nurses = db.staff.find(
        { "id": { "$nin": assigned_nurse_ids } },
        { "id": 1, "first_name": 1, "last_name": 1, "_id": 0 }
        )

    return list(unassigned_nurses)
```

Figura 25: Query Enfermeiras sem hospitalização

```
[{'first_name': 'Christopher', 'last_name': 'Martinez', 'number': 82},
 {'first_name': 'Lee', 'last_name': 'Collins', 'number': 83},
 {'first_name': 'Ashley', 'last_name': 'Lucas', 'number': 85},
 {'first_name': 'Rebecca', 'last_name': 'Reyes', 'number': 89},
 {'first_name': 'Sarah', 'last_name': 'Boyd', 'number': 92},
 {'first_name': 'Robert', 'last_name': 'Perez', 'number': 96},
 {'first_name': 'James', 'last_name': 'Palmer', 'number': 97},
 {'first_name': 'Brittany', 'last_name': 'Collins', 'number': 99},
 {'first_name': 'Angela', 'last_name': 'Park', 'number': 100},
 {'first_name': 'Jillian', 'last_name': 'Gordon', 'number': 1},
 {'first_name': 'James', 'last_name': 'Williams', 'number': 2},
 {'first_name': 'Joshua', 'last_name': 'Carter', 'number': 3},
 {'first_name': 'Lisa', 'last_name': 'Hayes', 'number': 6},
 {'first_name': 'Dawn', 'last_name': 'Roberts', 'number': 8},
 {'first_name': 'Jessica', 'last_name': 'Jones', 'number': 9},
 {'first_name': 'Kimberly', 'last_name': 'Blankenship', 'number': 11},
 {'first_name': 'William', 'last_name': 'Grant', 'number': 13},
 {'first_name': 'Andrew', 'last_name': 'Deleon', 'number': 14},
 {'first_name': 'Emily', 'last_name': 'Rowe', 'number': 15},
 {'first_name': 'Daniel', 'last_name': 'Mills', 'number': 17},
 {'first_name': 'Scott', 'last_name': 'Holmes', 'number': 19},
 {'first_name': 'James', 'last_name': 'Carpenter', 'number': 24},
 {'first_name': 'Noah', 'last_name': 'Terry', 'number': 30},
 {'first_name': 'Dillon', 'last_name': 'Jones', 'number': 34},
```

Figura 26: Resultado da query que apresenta as enfermeiras que nunca foram alocadas a um paciente hospitalizado

8. **Quantidade de quartos mais comuns:** De modo a entender quais são

os quartos com mais procurados por pacientes, esta query foi criada de modo a dar uma ideia ao hospital do tipo de quarto em que devem investir.

```python
def room_mais_comum():
    db = setupConnection()
    result = db.episodes.aggregate([
        {"$unwind": "$hospitalizations"},
        {
            "$group": {
                "_id": "$hospitalizations.room_id",
                "count": { "$sum": 1 }
            }
        },
        {
            "$lookup": {
                "from": "rooms",
                "localField": "_id",
                "foreignField": "room_id",
                "as": "room_info"
            }
        },
        {"$unwind": "$room_info"},
        {
            "$group": {
                "_id": "$room_info.type",
                "totalCount": { "$sum": "$count" }
            }
        },
        { "$sort": { "totalCount": -1 } },
        { "$limit": 1 }
    ])

    result_list = list(result)
    if result_list:
        most_common_room = result_list[0]
        return most_common_room["_id"], most_common_room["totalCount"]
    else:
        return None
```

Figura 27: Query Quantidade de quartos mais comuns

```
[{'_id': 'Family', 'totalCount': 14},
 {'_id': 'Executive', 'totalCount': 14},
 {'_id': 'Penthouse', 'totalCount': 14},
 {'_id': 'Economy', 'totalCount': 13},
 {'_id': 'Single', 'totalCount': 10}]
```

Figura 28: Resultado da query que apresenta os quartos mais utilizados

9. **Custo médio por tipo de quarto:** Esta consulta revela o custo médio de cada tipo de quarto no hospital. Verifica-se que o custo médio de uma *penthouse* é de 528.57€, seguido pelo quarto *executive* com uma média de 411.43€. A opção mais barata é o quarto económico que fica em média por 76.15€.

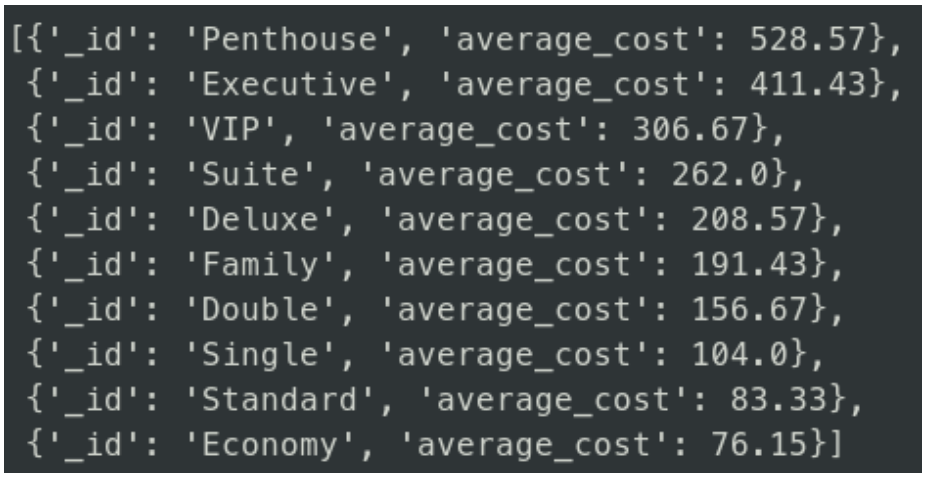


Figura 29: Resultado da query que apresenta o custo, em média, de cada tipo de quarto oferecido pelo hospital

10. **Número de pacientes por género:** A presente consulta determina o número de pacientes de cada género. Verifica-se que existem 46 pacientes de género masculino e 44 pacientes de género feminino, totalizando 100 pacientes no hospital.

```python
def total_patients_per_gender():
    db = setupConnection()
    result = db.patients.aggregate([
        {
            "$group": {
                "_id": "$gender",
                "count": {"$sum": 1}
            }
        },
        {"$sort": {"count": -1}},
        {
            "$project": {
                "_id": 0,
                "gender": "$_id",
                "count": 1
            }
        }
    ])
    return list(result)
```

Figura 30: Query Número de pacientes por género

```
[{'count': 46, 'gender': 'Male'}, {'count': 44, 'gender': 'Female'}]
```

Figura 31: Resultado da divisão de pacientes por género

11. **Pacientes com múltiplos contactos de emergência:** Identifica que pacientes possuem mais do que um contacto de emergência, auxiliando em situações de urgência onde é necessário que haja múltiplas formas de contactar os responsáveis pelos pacientes.

```
[{'emergency_contact_count': 2, 'name': 'John Doe'},
 {'emergency_contact_count': 2, 'name': 'Emily Brown'},
 {'emergency_contact_count': 2, 'name': 'Sophia Garcia'},
 {'emergency_contact_count': 3, 'name': 'James Lopez'}]
```

Figura 32: Ilustração dos pacientes com dois ou mais contactos de emergência

5.3 Triggers e Procedures

Devido ao facto do MongoDB dar prioridade a performance e flexibilidade, em conjunto com a falta de suporte para triggers até as versões mais recentes e a tendência para ser uma base de dados distribuída, dividida em *clusters*, não acreditamos que seria justificável adicionar mais do que um único *trigger* visto que estes não tiram proveito das vantagens principais desta base de dados. Contudo, adicionamos três procedures, tendo um deles a mesma função daquele incluído na base de dados relacional.

Os procedures novos que implementamos foram os seguintes:

1. **Adicionar stock a um medicamento:** com o nome completo do medicamento e a quantidade a acrescentar, incrementa o stock deste medicamento pela quantidade indicada.

2. **Remover stock a um medicamento:** funciona de maneira semelhante, contudo também é dado um custo e apenas retira unidades até á quantidade existente em stock, impedindo que o stock seja negativo. Caso o custo total seja menor do que o orçamento apresentado, isto também limita a quantidade de medicamentos obtidos.

O trigger implementado ocorre quando um documento é inserido na coleção com os episódios. Quando isto acontece, os pacientes que não possuem contas irão ser atualizados para incluir uma conta com os valores das prescrições, hospitalizações e análises que fizeram, e a conta ficará com estado "PENDING".

Caso o paciente já possua uma conta e esta não tenha mudado de valor, o seu estado irá manter-se igual. Caso o valor seja diferente, ou seja, eventualmente o paciente foi consultado por um especialista, fez análises ou foi hospitalizado. Nesse caso, o valor da conta é atualizado e o estado muda para "PENDING".

6 Neo4J

O Neo4j caracteriza-se por ser uma base de dados orientada a grafos e, por isso, organiza os seus dados através de nodos e relações entre eles.

6.1 Estrutura de Dados

Para definir a estrutura de dados em neo4j foi necessário abstraírmos das funções de cada tabela. Apesar da base de dados original pode ser dividida em três vertentes (Pacientes, Episódios, Staff), é possível assumir que um membro do staff é uma "Pessoa", assim como um paciente. Logo, decidimos criar um único nodo chamado **Pessoa** que poderia ser um paciente, um contacto de emergência ou um membro do staff. Sendo que um elemeto da tabela staff poderia ser um médico, enfermeiro ou técnico, juntamos os atributos de membro do staff aos seus dados, criando assim um nodo por membro do staff. Aplicando a mesma lógica mas desta vez para os episódios, criamos um nodo chamado **Episódio** que poderia ser uma sequência de análises, uma hospitalização ou uma

consulta. Para restantes elementos, nomeadamente Bill, Department, Insurance, Medical History, Medicine, Prescription e Room, foi criado um nodo para cada um.

Figura 33: Gráfico Neo4j

6.1.1 Nodos

Desta forma, foram criados diversos nodos para cada vertente, nomeadamente.:

- ***Person.* Contém**
  1. Patient — Nodo representante da tabela do paciente;
  2. Doctor — Nodo representante da tabela do doutor;
  3. Nurse — Nodo representante da tabela da enfermeira;
  4. Technician — Nodo representante da tabela do técnico;
  5. EmergencyContact — Nodo representante da tabela do contacto de emergência do paciente.
- ***Episode.* Contém:**
  1. Hospitalization — Nodo representante da tabela da hospitalização;
  2. Appointment — Nodo representante da tabela da consulta previamente marcada;
  3. LabScreening — Nodo representante da tabela dos exames do hospital;

- ***Prescription*** — Nodo representante da tabela da prescrições;
- ***Medicine*** — Nodo representante da tabela dos medicamentos;
- ***Department*** — Nodo representante da tabela do departamento;
- ***MedicalHistory*** — Nodo representante da tabela do histórico médico do paciente;
- ***Insurance*** — Nodo representante da tabela do seguro do paciente;
- ***Room*** — Nodo representante da tabela do episódio do quarto correspondente à hospitalização;
- ***Bill*** — Nodo representante da tabela da fatura do hospital;

6.1.2 Relação entre Nodos

Considerando os nodos descritos, foram criadas as seguintes relações.:

- **APPOINTED_DOCTOR** - Relaciona o episódio (consulta) de um paciente com o médico que o atendeu;
- **HAS_BILL** — Relaciona os pacientes e as suas faturas;
- **HAS_EMERGENCY_CONTACT** — Relaciona os pacientes aos seus contactos de emergência;
- **HAS_EPISODE** — Relaciona os pacientes aos seus episódios específicos de atendimento médico ou hospitalar;
- **HAS_INSURANCE** — Relaciona os pacientes com os seus planos de seguro de saúde;
- **HAS_LAB_SCREENING** — Relaciona o episódio de um paciente com os exames de laboratório realizados;
- **HAS_MEDICAL_HISTORY** — Relaciona os pacientes ao seu histórico médico;
- **HAS_PRESCRIPTION** — Relaciona os pacientes às suas prescrições médicas;
- **HAS_ROOM** - Relaciona o episódio de um paciente (hospitalização) com o quarto em que esteve internado;
- **PRESCRIBED_MEDICINE** - Relaciona a prescrição médica de um paciente com o medicamento prescrito;
- **RESPONSIBLE_NURSE** - Relaciona o episódio (hospitalização) de um paciente com o enfermeiro responsável;

- **TESTED_BY** - Relaciona o episódio (que contém análises) com o técnico que o realizou;
- **WORKS_IN** — Relaciona funcionários (como médicos, enfermeiros ou técnicos) ao departamento em que trabalham.

6.2 Queries

6.2.1 Top 5 médicos com mais consultas e os departamentos onde trabalham:

Esta consulta procura os cinco médicos com o maior número de consultas. Primeiramente, conta o número de consultas para cada médico e ordena-os por ordem decrescente e, posteriormente, encontra o departamento onde trabalham. Por fim, é retornado o ID do médico, o seu primeiro e último nome, o departamento onde trabalha e o número de consultas.

```
MATCH (d:PERSON)<-[:APPOINTED_DOCTOR]-(a:EPISODE)
WITH d, count(a) AS appointmentCount
ORDER BY appointmentCount DESC
LIMIT 5
MATCH (d)-[:WORKS_IN]->(dept:DEPARTMENT)
RETURN d.empFname AS doctorFirstName, d.empLname AS doctorLastName,
    appointmentCount, dept.deptName AS department
```

doctorFirstName	doctorLastName	appointmentCount	department
Emily	Rowe	7	Ophthalmology
Robert	Perez	6	Emergency_2
Ashley	Lucas	6	Urology
Brittany	Collins	6	Emergency_1
Rebecca	Reyes	6	Pulmonology

6.2.2 Pacientes sem seguro:

Esta consulta identifica os pacientes que não têm seguro de saúde. Por fim, é retornado o ID do paciente, primeiro nome e último nome para cada paciente que não possui uma relação com o seguro. O resultado indica que todos os pacientes têm seguro.

```
MATCH (p:PERSON)
WHERE NOT EXISTS {
    MATCH (p)-[:HAS_INSURANCE]->(:INSURANCE)
} AND p.idPatient IS NOT NULL
RETURN p.idPatient AS PatientID, p.patient_fname AS FirstName,
     p.patient_lname AS LastName;
```

6.2.3 Departamentos com mais médicos ativos:

Nesta consulta são listados os departamentos com o maior número de médicos ativos. Primeiramente, conta o número de médicos ativos em cada departamento. Por fim, retorna o nome e ID do departamento, juntamente com o número de médicos ativos, ordenados por quantidade de médicos ativos em ordem decrescente.

```
MATCH (d:DEPARTMENT)<-[:WORKS_IN]-(p:PERSON)
WHERE p.isActiveStatus = 'Y'
RETURN d.deptName AS department, count(p) AS activeDoctors
ORDER BY activeDoctors DESC
LIMIT 5
```

department	activeDoctors
Radiology	8
Dentistry	6
Endocrinology	5
Cardiology_1	5
Gynecology	5

6.2.4 Planos de seguro que cobrem condições específicas

Aqui, são listados os planos de seguro que cobrem uma condição específica, neste caso, Diabetes. A consulta conta o número de pacientes que têm histórico médico de diabetes e se estão associados a um plano de seguro. Por fim, é retornado o número da apólice do seguro, o provedor e a contagem de pacientes cobertos, ordenados por contagem decrescente.

```
MATCH (i:INSURANCE)<-[:HAS_INSURANCE]-(p:PERSON)
    -[:HAS_MEDICAL_HISTORY]->(mh:MEDICAL_HISTORY)
WHERE mh.condition = 'Diabetes'
RETURN i.policy_number AS PolicyNumber, i.provider AS Provider,
    count(p) AS PatientsCount
ORDER BY PatientsCount DESC;
```

PolicyNumber	Provider	PatientsCount
POL007	PQR Insurance	1

6.2.5 Técnico responsável pelo teste de laboratório mais caro:

Esta consulta identifica o técnico responsável pelo teste de laboratório mais caro. Primeiramente, é classificado os testes de laboratório por custo em ordem decrescente e retornado as informações do técnico responsável pelo teste de laboratório mais caro, incluindo o ID do técnico, primeiro nome, último nome, ID do laboratório e custo do teste.

```
MATCH (ls:LABSCREENING)-[:TESTED_BY]->(t:PERSON)
WITH t, ls, ls.testCost AS cost
ORDER BY cost DESC
LIMIT 1
RETURN t.empFname AS technicianFirstName,
t.empLname AS technicianLastName, cost
```

technicianFirstName	technicianLastName	cost
"Ashley"	"Stein"	198.34

6.2.6 Top 5 técnicos com mais testes de laboratório:

Aqui, são listados os cinco técnicos com o maior número de testes de laboratório atribuídos a eles. A consulta conta o número de testes de laboratório para cada técnico e os ordena em ordem decrescente. Por fim, é retornado o ID do técnico, primeiro nome, último nome e contagem de testes de laboratório para os cinco técnicos com mais testes atribuídos.

```
MATCH (ls:LABSCREENING)-[:TESTED_BY]->(t:PERSON)
WITH t, count(ls) AS testCount
ORDER BY testCount DESC
LIMIT 5
RETURN t.empFname AS technicianFirstName,
t.empLname AS technicianLastName, testCount
```

technicianFirstName	technicianLastName	testCount
"Deborah"	"Collins"	15
"Linda"	"White"	14
"Ashley"	"Stein"	14
"Allison"	"Miller"	13
"Emily"	"Ryan"	13

6.2.7 Top 5 medicamentos prescritos por médicos:

Nesta consulta, são listados os cinco medicamentos mais prescritos pelos médicos. Primeiramente, é contado o número de vezes que cada medicamento foi prescrito e ordenado de forma decrescente. Por fim, é retornado o ID do medicamento, nome do medicamento e contagem de prescrições para os cinco medicamentos mais prescritos.

```
MATCH (p:PRESCRIPTION)-[:PRESCRIBED_MEDICINE]->(m:MEDICINE)
WITH m, count(p) AS prescriptionCount
ORDER BY prescriptionCount DESC
LIMIT 5
RETURN m.name AS medicineName, prescriptionCount
```

medicineName	prescriptionCount
"Paracetamol"	200
"Ibuprofen"	186
"Amoxicillin"	164
"Ciprofloxacin"	148
"Lisinopril"	129

6.2.8 Pacientes com maior dívida pendente

Esta consulta identifica os pacientes que acumularam o maior valor em dívidas ainda não pagas. Primeiramente, são filtradas as faturas com status de pagamento pendente e calculada a soma dos valores devidos para cada paciente. Por fim, o resultado é ordenado em ordem decrescente pelo total devido, listando os pacientes que mais devem primeiro.

```
MATCH (p:PERSON)-[:HAS_EPISODE]->(e:EPISODE)-[:HAS_BILL]->(b:BILL)
WHERE b.paymentStatus = 'PENDING'
RETURN p.idPatient AS patientId, p.patientFname AS firstName,
    p.patientLname AS lastName, sum(b.total) AS totalDebt
ORDER BY totalDebt DESC
LIMIT 5
```

patientId	firstName	lastName	totalDebt
75	"Avery"	"Hoang"	14978.98
55	"Isabella"	"Huynh"	14813.35
28	"Sophia"	"Le"	13496.36
54	"Noah"	"Vo"	12020.0
68	"Alexander"	"Dang"	11359.58

6.2.9 Pacientes que mais visitam o hospital

Esta consulta identifica os pacientes que mais frequentam o hospital, ordenados por número de visitas em ordem decrescente. Inicialmente, conta o número de episódios associados a cada paciente, divididos em categorias de consulta, hospitalização e triagem laboratorial. Em seguida, soma esses números para calcular o total de visitas de cada paciente. Por fim, retorna o primeiro nome, sobrenome e o total de visitas de cada paciente, bem como o número de consultas, hospitalizações e triagens laboratoriais individuais, ordenando pelo total de visitas em ordem decrescente.

```
MATCH (p:PERSON)-[:HAS_EPISODE]->(e:EPISODE)
WITH p, count(e) AS visitCount
ORDER BY visitCount DESC
LIMIT 5
RETURN p.idPatient AS patientId, p.patientFname AS firstName,
    p.patientLname AS lastName, visitCount
```

patientId	firstName	lastName	visitCount
30	"Amelia"	"Huynh"	8
89	"Amelia"	"Tran"	4
11	"Jacob"	"Rodriguez"	4
14	"Mia"	"Gomez"	4
85	"Scarlett"	"Dang"	4

6.2.10 Quartos com maior número de hospitalizações

Esta consulta identifica os quartos de hospital mais utilizados, calculando a quantidade de hospitalizações para cada quarto. Por fim, é retornado o identificador do quarto, o tipo de quarto e a contagem de hospitalizações, ordenados do maior para o menor uso.

```
MATCH (e:EPISODE)-[:HAS_ROOM]->(r:ROOM)
WITH r, count(e) AS hospitalizationCount
ORDER BY hospitalizationCount DESC
LIMIT 5
RETURN r.idRoom AS roomId, r.roomType AS roomType, hospitalizationCount
```

roomId	roomType	hospitalizationCount
40	"Executive"	7
38	"Family"	7
37	"Economy"	7
39	"Penthouse"	7
41	"Single"	6

6.3 Triggers e Procedures

No contexto de SQL, os triggers são processos automáticos que são acionados em resposta a eventos como inserções, atualizações ou exclusões de dados. Esses gatilhos são frequentemente utilizados para impor restrições de integridade referencial, aplicar regras de negócios e automatizar tarefas repetitivas. Por outro lado, as procedures SQL são conjuntos de instruções armazenadas no banco de dados, projetadas para executar operações complexas e encapsular lógica de processamento de dados. Elas oferecem uma maneira de modularizar e reutilizar código SQL, promovendo a manutenção e a escalabilidade do sistema.

No contexto do Neo4j, o paradigma de triggers e procedures também desempenha um papel significativo, embora de uma forma ligeiramente diferente. Enquanto que os triggers nas bases de dados relacionais respondem a eventos em tabelas, em Neo4j, os gatilhos podem ser implementados para responder a mudanças na estrutura do grafo ou em propriedades de nós e relacionamentos.

Além disso, as procedures em Neo4j são utilizadas para executar consultas complexas, realizar operações de transformação de dados e implementar lógica de aplicação diretamente no banco de dados de grafos.

6.3.1 Procedures

Em Neo4j, usamos o procedure apoc.cypher.doIt com a seguinte estrutura:

Name	Type	Default
cypher	STRING?	null
params	MAP?	null

- **Procedure sp_update_bill_status**

  Este procedure, pelo nome indica, atualiza o estado de uma fatura. Ele obtém o valor total da fatura associada ao ID fornecido e compara esse valor com o valor pago: Se o valor pago for menor que o total da fatura, o estado de pagamento é atualizado para 'FAILURE' e um erro é enviado. Caso contrário, o estado de pagamento é atualizado para 'PROCESSED'.

```
CALL apoc.cypher.doIt("
    MATCH (b:BILL {idBill: $billId})
    SET b.payment_status = CASE
        WHEN $paidValue < b.total THEN 'FAILURE'
        ELSE 'PROCESSED'
    END
    RETURN CASE
        WHEN $paidValue < b.total
        THEN 'Paid value is inferior to the total value of the bill.'
        ELSE 'Payment processed successfully.'
    END AS result
", {billId: $billId, paidValue: $paidValue}) YIELD value
RETURN value.result;
```

  Assim, recebendo dois valores (billI e paidValue), este procedure faz o mesmo trabalho mas em Neo4j, ou seja, altera o estado da fatura. Inicialmente, tivemos que fazer alterações nas definições da base de dados para que a chamada de apoc.cypher.doIt seja feita com sucesso.

- **Procedure update_emergency_contact**

  Este procedimento atualiza as informações de contacto de emergência de um paciente., visto que retorna o ID do paciente e as novas informações de contacto, como nome, telefone e relação. Através destes argumentos,

ele encontra o contacto de emergência associado ao paciente e atualiza suas informações.

```
CALL apoc.cypher.doIt("
    MATCH (p:PERSON {idPatient: $idPatient})
    CREATE (ec:PERSON {
        contact_name: $contact_name, phone: $phone, relation: $relation
    })
    MERGE (p)-[:HAS_EMERGENCY_CONTACT]->(ec)
    RETURN 'Emergency contact updated successfully.' AS result
",
{idPatient: $idPatient, contact_name: $contact_name,
phone: $phone, relation: $relation}) YIELD value
RETURN value.result;
```

Assim, recebendo os valores (`idPatient`, `contact_name`, `phone`, `relation`), este procedure faz o mesmo trabalho, mas em Neo4j, ou seja, altera as informações de contato de emergência do paciente. Inicialmente, tivemos que fazer alterações nas definições da base de dados para que a chamada de `apoc.cypher.doIt` seja feita com sucesso.

6.3.2 Trigger trg_generate_bill

Este trigger é acionado quando alguém der alta a um paciente, calculando todos os custos (de análises, consultas, quartos e receitas médicas) e criando uma fatura com esse valor.

Em Neo4j, usamos o procedure apoc.trigger.install que passou a ser o principal procedure de inserção de triggers desde a versão 4.4 do APOC. Eis a sua estrutura:

Name	Type	Default
databaseName	STRING?	null
name	STRING?	null
statement	STRING?	null
selector	MAP?	null
config	MAP?	{}

Sendo assim, a nossa solução foi:

```
CALL apoc.trigger.install('neo4j','trg_generate_bill', '
    MATCH (episode: EPISODE)
    MATCH (episode)-[r:HAS_BILL]->(b:BILL)
    WHERE r IS NULL
    WITH episode
```

```
    OPTIONAL MATCH (b:BILL)
    WITH episode, max(b.idBill) as maxIdBill

    OPTIONAL MATCH (episode)-[:HAS_ROOM]->(room:ROOM)
    OPTIONAL MATCH (episode)-[:HAS_LAB_SCREENING]->
        (lab_screening:LABSCREENING)
    OPTIONAL MATCH (episode)-[:HAS_PRESCRIPTION]->
        (prescription:PRESCRIPTION)

    WITH episode, COALESCE(room.room_cost, 0) AS room_cost,
        COALESCE(lab_screening.test_cost, 0) AS test_cost,
        COALESCE(maxIdBill, 0) + 1 AS maxIdBill,
        (COALESCE(room.room_cost, 0) + COALESCE(lab_screening.test_cost, 0))
        AS total

    MERGE (bill:BILL {idBill: maxIdBill, idEpisode: episode.idEpisode,
        other_charges: 0, room_cost: room_cost, test_cost: test_cost,
        total: total, payment_status: "PENDING", registered_at: datetime()}
        )
    CREATE (episode)-[:HAS_BILL]->(bill)

RETURN bill',
{phase:'after'});
```

7 Análise Crítica

Iremos agora elaborar uma análise de cada um dos paradigmas que foram utilizados neste projeto, assim como identificar aspetos positivos e aspetos a melhorar do trabalho efetuado nas bases de dados não-relacionais.

7.1 Base de Dados Relacional

Como principais vantagens e desvantagens da base de dados que nos foi dada inicialmente, podemos anotar:

- **Pontos Fortes**
    1. **Backups:** É possível criar cópias de segurança, ou 'snapshots' da base de dados com relativa facilidade, o que permite a recuperabilidade de estados anteriores.
    2. **Robustez:** Sendo o paradigma mais antigo e utilizado para bases de dados, a robustez deste modelo é clara devido à sua obediência aos princípios *ACID*, o que leva a um alto grau de integridade dos dados.
- **Pontos Fracos**

1. **Esquema Definido:** Os dados seguem uma estrutura clara e obrigatória, o que constitui uma desvantagem quando existem bastantes atributos diferentes e nem todos têm valores.
2. Normalização dos Dados: A necessidade de normalizar os dados para obter e manter a integridade dos dados é uma desvantagem clara em determinadas situações.

Acreditamos que a base de dados foi explorada de maneira satisfatória, contudo a ausência de *queries* que envolvem manipulação de datas é um dos aspetos que poderia ser abordado com mais profundidade para obter mais *insights* sobre o histórico de funcionamento do hospital.

7.2 MongoDB

As características mais salientadas pela base de dados orientada a documentos utilizada neste projeto foram:

- **Pontos Fortes**
  1. **Performance:** Este paradigma está associado a um ritmo de leituras e escritas elevado(devido aos índices criados pela base de dados), sendo esta característica essencial para o contexto de análise estatística que escolhemos para o objetivo final do banco de dados.
  2. **Simplicidade:** Devido à natureza da base de dados orientada a documentos, percorrer a base de dados e modificar valores é um processo mais simples, o que agiliza a sua criação.
- **Pontos Fracos**
  1. **Falta de Apoio para Transações:** sem utilizar MongoDB Atlas(o serviço) e não apenas MongoDB, não é possível criar triggers para a base de dados.
  2. **Complexidade de Indexação e de Operações:** Em alguns casos, operações complexas ou a manutenção de vários índices num documento pode trazer problemas de performance(overhead exagerado).

O processo de criação de *procedures* e *queries* utilizando este paradigma revelou ser intuitivo e simples devido à estrutura de ficheiros (*JSON*) e ao número reduzido de coleções em comparação com o número de tabelas da base de dados relacional. Contudo, acreditamos que fizemos um erro ao criar a coleção de episódios pois separar a componente das contas seria mais apropriado. Isto porque assim conseguimos adicionar mais contas com bastante facilidade devido a sabermos o índice da última conta gerada. Como as contas estão integradas na coleção de episódios, isto iria complicar e reduzir consideravelmente o performance de um trigger que, por exemplo, calcula e adiciona novas contas a pacientes.

7.3 Neo4J

Os pontos fortes e fracos da base de dados orientada a grafos que observamos à medida da execução do projeto foram principalmente:

- **Pontos Fortes**

    1. **Modelagem Natural de Relacionamentos:** Neo4j permite a representação direta e intuitiva de dados complexos e inter-relacionados, o que facilita a modelagem dos dados;
    2. **Desempenho em Consultas Relacionais:** Consultas que envolvem múltiplos níveis de relacionamento são executadas de maneira eficiente em Neo4j, muitas vezes superando bancos de dados relacionais tradicionais em termos de velocidade e desempenho.
    3. **Escalabilidade e Desempenho:** Neo4j oferece várias opções para escalabilidade, incluindo sharding e clustering, o que permite a gestão de grandes volumes de dados e a distribuição de carga de trabalho.
    4. **Comunidade Ativa e Suporte:** Existe uma comunidade ativa de utilizadores, bem como uma abundância de recursos educacionais, documentação oficial e suporte empresarial.

- **Pontos Fracos**

    1. **Curva de Aprendizado Inicial:** Para aqueles não familiarizados com bancos de dados orientados a grafos ou a linguagem Cypher, a curva de aprendizado pode ser acentuada inicialmente.
    2. **Limitações de Transações Complexas:** Em alguns casos, operações de escrita complexas ou transações de longa duração podem ser menos eficientes, necessitando de otimizações cuidadosas.

Sendo assim, uma base de dados orientada a grafos, como o Neo4j, destaca-se pela modelagem natural de relacionamentos complexos e pelo desempenho eficiente em consultas relacionais, especialmente em ambientes que exigem escalabilidade e distribuição de carga de trabalho. Contudo, como pode enfrentar limitações em operações de escrita complexas ou transações de longa duração, exige otimizações cuidadosas. Apesar disso, a comunidade ativa e o suporte abrangente compensam algumas dessas desvantagens, oferecendo recursos educativos e ajuda profissional.

8 Conclusão

Em suma, acreditamos que os objetivos deste projeto foram, em grande parte, concluídos. É possível observar as diferenças, vantagens, desvantagens em utilizar cada paradigma, desde o nosso nível de familiaridade/facilidade de aprendizagem até à facilidade de escalonamento e performance de operações de leitura e escrita.

Verificamos também que cada tipo de base de dados tem as suas vantagens e desvantagens. A base de dados relacional, neste caso o Oracledb, mostra-se eficiente para transações completas e consultas estruturadas, no entanto não é a melhor escolha para dados não estruturados ou interligados. Já o MongoDB oferece flexibilidade e escalabilidade, sendo ideal para dados semi-estruturados e para aplicações que necessitam de iterações rápidas de desenvolvimento. Uma das desvantagens é que não se adequa para aplicações com transações complexas e os *triggers* podem ser uma limitação. O Neo4j é ideal para explorar relações complexas e analisar dados interligados, no entanto, a complexidade de modelação em casos onde existem grandes volumes de dados distribuídos pode ser um desafio.

Concluindo, com este projeto conseguimos desenvolver uma visão abrangente sobre diferentes tipos de abordagens para resolver problemas de gestão de dados através da migração e adaptação dos dados hospitalares para diferentes paradigmas de bases de dados.