Bases de Dados

2023/2024

Projecto - Entrega 2

0. Carregamento da Base de Dados

Crie a base de dados “Saude” no PostgreSQL e execute os comandos para criação das tabelas desta base de dados apresentados no **Anexo A**

1. Restrições de Integridade

Implemente na base de dados “Saude” as seguintes restrições de integridade, podendo recorrer a extensões procedimentais (*Triggers* e *Stored Procedures*) caso estritamente necessário:

RI-1 Os horários das consultas são à hora exata ou meia-hora no horário 8-13h e 14-19h

RI-2 Um médico não se pode consultar a si próprio, embora possa ser paciente de outros médicos no sistema

RI-3 Um médico só pode dar consultas na clínica em que trabalha no dia da semana correspondente à data da consulta

Os mecanismos **ON DELETE CASCADE** e **ON UPDATE CASCADE** não são permitidos.

2. Preenchimento da Base de Dados

Preencha todas as tabelas da base de dados de forma consistente (após execução do ponto anterior) com os seguintes requisitos adicionais de cobertura:

- 5 clínicas, de pelo menos 3 localidades diferentes do distrito de Lisboa
- 5-6 enfermeiros por clínica
- 20 médicos de especialidade ‘clínica geral’ e 40 outros distribuídos como entender por até 5 outras especialidades médicas (incluindo pelo menos, ‘ortopedia’ e ‘cardiologia’). Cada médico deve trabalhar em pelo menos duas clínicas, e em cada clínica a cada dia da semana (incluindo fins de semana), devem estar pelo menos 8 médicos
- Cerca de 5.000 pacientes
- Um número mínimo de consultas em 2023 e 2024 tais que cada paciente tem pelo menos uma consulta, e em cada dia há pelo menos 20 consultas por clínica, e pelo menos 2 consultas por médico
- ~80% das consultas tem receita médica associada, e as receitas têm 1 a 6 medicamentos em quantidades entre 1 e 3

- Todas as consultas têm 1 a 5 observações de sintomas (com parâmetro mas sem valor) e 0 a 3 observações métricas (com parâmetro e valor). Deve haver ~50 parâmetros diferentes para os sintomas (sem valor) e ~20 parâmetros diferentes para as observações métricas (com valor) e os dois conjuntos devem ser disjuntos.
- Todas as moradas são nacionais e seguem o formato Português, terminando com código postal: XXXX-XXX e de seguida a localidade.

Deve ainda garantir que todas as consultas necessárias para a realização dos pontos seguintes do projeto produzem um **resultado não vazio**.

Pode realizar o preenchimento da base de dados por meio de um ficheiro "populate.sql" com comandos INSERT, executando na sua base de dados, ou alternativamente por meio de ficheiros tabulares de texto (um por tabela) utilizando comandos COPY na base de dados para popular as tabelas. Em qualquer dos casos, todos os comandos e ficheiros usados devem ser incluídos na entrega do projeto.

Pode utilizar ferramentas de **IA generativo** (e.g. chatGPT) para gerar os comandos INSERT ou os ficheiros tabulares de texto, desde que o resultado cumpra com os requisitos apresentados.

3. Desenvolvimento de Aplicação

Crie um protótipo de RESTful *web service* para gestão de consultas por acesso programático à base de dados 'Saude' através de uma API que devolve respostas em JSON, implementando os seguintes *endpoints REST*:

Endpoint	Descrição
`/`	Lista todas as clínicas (nome e morada).
`/c/<clinica>/`	Lista todas as especialidades oferecidas na `<clinica>`.
`/c/<clinica>/<especialidade>/`	Lista todos os médicos (nome) da `<especialidade>` que trabalham na `<clínica>` e os primeiros três horários disponíveis para consulta[1] de cada um deles (data e hora).
`/a/<clinica>/registar/`	Registra uma marcação de consulta na `<clinica>` na base de dados (populando a respectiva tabela). Recebe como argumentos um paciente, um médico, e uma data e hora (posteriores ao momento de agendamento).
`/a/<clinica>/cancelar/`	Cancela uma marcação de consulta que ainda não se realizou na `<clinica>` (o seu horário é posterior ao momento do cancelamento), removendo a entrada da respectiva tabela na base de dados. Recebe como argumentos um paciente, um médico, e uma data e hora.

[1] Um horário está disponível para consulta se é posterior ao momento da interrogação, cumpre as restrições horárias da tabela consulta (RI-1 do ponto 1), e o médico não tem ainda entrada na tabela consulta nesse horário

A solução deve prezar pela segurança, prevenindo ataques por **SQL *injection***, e deve garantir a atomicidade das operações sobre a base de dados com recurso a **transações**.

Para facilitar a marcação de consultas, pode construir uma tabela acessória com todos os horários de 2024 que cumprem as restrições horárias da tabela consulta. Se o fizer, deve incluir o respectivo código juntamente com o código da aplicação.

Os endpoints de marcação e cancelamento de consultas devem devolver mensagens explícitas ou confirmando que dados foram inseridos/removidos ou indicando porque motivo não foi possível realizar a operação.

Todo o código da aplicação deve ser incluído na entrega do projeto. A aplicação deve ainda estar disponível *online*, no ambiente de desenvolvimento docker providenciado para a disciplina, para demonstração durante a discussão.

4. Vistas

Crie uma **vista materializada** que detalhe as informações mais importantes sobre as consultas dos pacientes, combinando a informação de várias tabelas da base de dados. A vista deve ter o seguinte esquema:

historial_paciente(*id*, *ssn*, *nif*, *nome*, *data*, *ano*, *mes*, *dia_do_mes*, *localidade*, *especialidade, tipo*, *chave*, *valor*)

em que:

- *id*, *ssn*, *nif*, *nome* e *data*: correspondem ao atributos homónimos da tabela *consulta*
- *ano, mes* e *dia_do_mes*: são derivados do atributo *data* da tabela *consulta*[2]
- *localidade*: é derivado do atributo *morada* da tabela *clinica*[3]
- *especialidade*: corresponde ao atributo homónimo da tabela medico
- *tipo*: toma os valores 'observacao' ou 'receita' consoante o preenchimento dos campos seguintes
- *chave*: corresponde ao atributo *parametro* da tabela *observacao* ou ao atributo *medicamento* da tabela *receita*
- *valor*: corresponde ao atributo *valor* da tabela *observacao* ou ao atributo *quantidade* da tabela *receita*

5. Análise de Dados (SQL e OLAP)

Usando a vista desenvolvida no ponto anterior, complementada com outras tabelas da base de dados 'Saude' quando necessário, apresente a consulta SQL mais sucinta para cada um dos seguintes objetivos

[2] Pode utilizar a função EXTRACT() para obter partes de datas ou timestamps.
[3] Pode utilizar a função SUBSTRING() especificando um padrão POSIX para extrair o nome da localidade após o código postal da morada.

analíticos. Pode usar as instruções ROLLUP, CUBE, GROUPING SETS ou as cláusulas UNION of GROUP BY para os objetivos em que lhe parecer adequado.

1. Determinar que paciente(s) tiveram menos progresso no tratamento das suas doenças do foro ortopédico para atribuição de uma consulta gratuita. Considera-se que o indicador de falta de progresso é o intervalo temporal máximo entre duas observações do mesmo sintoma (i.e. registos de *tipo* 'observacao' com a mesma *chave* e com *valor* NULL) em consultas de ortopedia.
2. Determinar que medicamentos estão a ser usados para tratar doenças crónicas do foro cardiológico. Considera-se que qualificam quaisquer medicamentos receitados ao mesmo paciente (qualquer que ele seja) pelo menos uma vez por mês durante os últimos doze meses, em consultas de cardiologia.
3. Explorar as quantidades totais receitadas de cada medicamento em 2023, globalmente, e com drill down nas dimensões espaço (localidade > clinica), tempo (mes > dia_do_mes), e médico (especialidade > nome [do médico]), separadamente.
4. Determinar se há enviesamento na medição de algum parâmetros entre clínicas, especialidades médicas ou médicos, sendo para isso necessário listar o valor médio e desvio padrão de todos os parâmetros de observações métricas (i.e. com *valor* não NULL) com drill down na dimensão médico (globalmente > especialidade > nome [do médico]) e drill down adicional (sobre o anterior) por clínica.

6. Índices

Apresente as instruções SQL para criação de índices para melhorar os tempos de cada uma das consultas listadas abaixo sobre a base de dados 'Saude'. **Justifique a sua escolha** de tabela(s), atributo(s) e tipo(s) de índice, explicando que operações seriam otimizadas e como. Considere que não existam índices nas tabelas, além daqueles implícitos ao declarar chaves primárias e estrangeiras, e para efeitos deste exercício, suponha que o tamanho das tabelas excede a memória disponível em várias ordens de magnitude.

6.1

```
SELECT nome
FROM paciente
JOIN consulta USING (ssn)
JOIN observacao USING (id)
WHERE parametro = 'pressão diastólica'
AND valor >= 9;
```

6.2

```
SELECT especialidade, SUM(quantidade) AS qtd
FROM medico
JOIN consulta USING (nif)
JOIN receita USING (codigo_sns)
```

```
WHERE data BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY especialidade
ORDER BY qtd;
```

Entrega

A avaliação do projeto será efetuada com base no relatório apresentado pelos alunos contendo as respostas aos itens solicitados acima, bem como da subsequente discussão oral. A tabela seguinte mostra a cotação de cada item.

Item	**Pontos**
Restrições de Integridade	3
Preenchimento da Base de Dados	1
Aplicação	6
Vistas	2
Análise de dados	6
Índices	2

A submissão deve ser feita na forma de um ficheiro **`entrega-bd-02-GG.zip`**[4], onde **GG** é o número do grupo, estruturado da seguinte forma:

`E2-report-GG.ipynb` (onde GG é o número do grupo)	**Um ficheiro Jupyter Notebook correspondendo ao preenchimento do template "E2-report.ipynb" disponibilizado na página da disciplina, com o nome modificado para incluir o número do grupo** Deverá preencher a informação da "folha de rosto" com a indicação do **número do grupo**, o **número e nome** dos alunos que o constituem, tal como **a percentagem relativa de contribuição** de cada aluno com o respectivo **esforço (horas)**, o **turno** a que o grupo pertence e o **nome do docente de laboratório** responsável. Deverá preencher cada uma das secções subsequentes: 1. Implementação de **Restrições de Integridade** em SQL

[4] ⚠ O formato do arquivo deve ser exclusivamente ZIP ou GZ. Outros formatos de arquivo não serão aceites.

	2. Código para **Preenchimento da Base de Dados** (executando um ficheiro 'populate.sql' ou com comandos COPY para importar ficheiros .txt incluídos na pasta data/ listada abaixo) 3. Explicação da **Aplicação** desenvolvida, listando e descrevendo todos os ficheiros incluídos na pasta web/ (listada abaixo) 4. Código para criação de **Vistas** 5. Código para **Análise de Dados (SQL & OLAP)** 6. Respostas aos **Índices** O ficheiro **"E2-report.ipynb"** pode ser importado para o ambiente de trabalho disponibilizado para as aulas de laboratório[5] (basta colocar na pasta work/), que serve de ambiente de teste para as partes em SQL. Deve popular a base de dados de forma a assegurar que o resultado das suas queries é não-vazio. Deve ainda certificar-se que todo o código SQL é executável no ambiente de trabalho.
`data/`	Pasta com o(s) arquivo(s) .sql ou .txt para população da base de dados
`app/`	Pasta com os arquivos da aplicação web

IMPORTANTE: Serão aplicadas penalizações aos grupos que não cumprirem o formato de submissão. Não serão aceites submissões fora do prazo.

[5] https://github.com/bdist/db-workspace

Anexo A - Esquema SQL

```
DROP TABLE IF EXISTS clinica CASCADE;
DROP TABLE IF EXISTS enfermeiro CASCADE;
DROP TABLE IF EXISTS medico CASCADE;
DROP TABLE IF EXISTS trabalha CASCADE;
DROP TABLE IF EXISTS paciente CASCADE;
DROP TABLE IF EXISTS receita CASCADE;
DROP TABLE IF EXISTS consulta CASCADE;
DROP TABLE IF EXISTS observacao CASCADE;

CREATE TABLE clinica(
	nome VARCHAR(80) PRIMARY KEY,
	telefone VARCHAR(15) UNIQUE NOT NULL CHECK (telefone ~ '^[0-9]+$'),
	morada VARCHAR(255) UNIQUE NOT NULL
);

CREATE TABLE enfermeiro(
	nif CHAR(9) PRIMARY KEY CHECK (nif ~ '^[0-9]+$'),
	nome VARCHAR(80) UNIQUE NOT NULL,
	telefone VARCHAR(15) NOT NULL CHECK (telefone ~ '^[0-9]+$'),
	morada VARCHAR(255) NOT NULL,
	nome_clinica VARCHAR(80) NOT NULL REFERENCES clinica (nome)
);

CREATE TABLE medico(
	nif CHAR(9) PRIMARY KEY CHECK (nif ~ '^[0-9]+$'),
	nome VARCHAR(80) UNIQUE NOT NULL,
	telefone VARCHAR(15) NOT NULL CHECK (telefone ~ '^[0-9]+$'),
	morada VARCHAR(255) NOT NULL,
	especialidade VARCHAR(80) NOT NULL
);

CREATE TABLE trabalha(
	nif CHAR(9) NOT NULL REFERENCES medico,
	nome VARCHAR(80) NOT NULL REFERENCES clinica,
	dia_da_semana SMALLINT,
	PRIMARY KEY (nif, dia_da_semana)
);

CREATE TABLE paciente(
	ssn CHAR(11) PRIMARY KEY CHECK (ssn ~ '^[0-9]+$'),
	nif CHAR(9) UNIQUE NOT NULL CHECK (nif ~ '^[0-9]+$'),
	nome VARCHAR(80) NOT NULL,
	telefone VARCHAR(15) NOT NULL CHECK (telefone ~ '^[0-9]+$'),
	morada VARCHAR(255) NOT NULL,
	data_nasc DATE NOT NULL
);
```

```
CREATE TABLE consulta(
	id SERIAL PRIMARY KEY,
	ssn CHAR(11) NOT NULL REFERENCES paciente,
	nif CHAR(9) NOT NULL REFERENCES medico,
	nome VARCHAR(80) NOT NULL REFERENCES clinica,
	data DATE NOT NULL,
	hora TIME NOT NULL,
	codigo_sns CHAR(12) UNIQUE CHECK (codigo_sns ~ '^[0-9]+$'),
	UNIQUE(ssn, data, hora),
	UNIQUE(nif, data, hora)
);

CREATE TABLE receita(
	codigo_sns VARCHAR(12) NOT NULL REFERENCES consulta (codigo_sns),
	medicamento VARCHAR(155) NOT NULL,
	quantidade SMALLINT NOT NULL CHECK (quantidade > 0),
	PRIMARY KEY (codigo_sns, medicamento)
);

CREATE TABLE observacao(
	id INTEGER NOT NULL REFERENCES consulta,
	parametro VARCHAR(155) NOT NULL,
	valor FLOAT,
	PRIMARY KEY (id, parametro)
);
```