Bases de Dados 2024/2025

Projecto - Entrega 2

0. Carregamento da Base de Dados

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

1. Restrições de Integridade [3 valores]

Implemente na base de dados “Aviacao” as seguintes restrições de integridade, podendo recorrer a *Triggers* caso necessário:

RI-1 Aquando do check-in (i.e. quando se define o assento em bilhete) a classe do bilhete tem de corresponder à classe do assento e o aviao do assento tem de corresponder ao aviao do voo

RI-2 O número de bilhetes de cada classe vendidos para cada voo não pode exceder a capacidade (i.e., número de assentos) do avião para essa classe

RI-3 A hora da venda tem de ser anterior à hora de partida de todos os voos para os quais foram comprados bilhetes na venda

Critérios de Avaliação

- Coerência da solução
- Simplicidade da solução

2. Preenchimento da Base de Dados [2 valores]

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

- ≥10 aeroportos internacionais (reais) localizados na Europa, com pelo menos 2 cidades tendo 2 aeroportos
- ≥10 aviões de ≥3 modelos distintos (reais), com um número de assentos realista; assuma que as primeiras ~10% filas são de 1a classe
- ≥5 voos por dia entre 1 de Janeiro e 31 de Julho de 2025, cobrindo todos os aeroportos e todos os aviões; garanta que para cada voo entre dois aeroportos se segue um voo no sentido oposto; garanta ainda que cada avião tem partida no aeroporto da sua chegada anterior
- ≥30.000 bilhetes vendidos até à data presente, correspondendo a ≥10.000 vendas, com todo os bilhetes de voos já realizados tendo feito check-in, e com todos os voos tendo bilhetes de primeira e segunda classe vendidos

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, ou alternativamente por meio de ficheiros tabulares de texto (um por tabela) utilizando comandos COPY. Todos os comandos e ficheiros usados devem ser incluídos na entrega do projeto.

Pode utilizar ferramentas de **IA generativo** (e.g. chatGPT), scripts ou qualquer outro meio para gerar os comandos INSERT ou ficheiros tabulares de texto.

Critérios de Avaliação

- Cumprimento dos requisitos de cobertura
- Resultado não vazio em todas as consultas do ponto 5

3. Desenvolvimento de Aplicação [5 valores]

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

Endpoint	Descrição
`/`	Lista todos os aeroportos (nome e cidade).
`/voos/<partida>/`	Lista todos os voos (número de série do avião, hora de partida e aeroporto de chegada) que partem do aeroporto de `<partida>` até 12h após o momento da consulta.
`/voos/<partida>/<chegada>/`	Lista os próximos três voos (número de série do avião e hora de partida) entre o aeroporto de `<partida>` e o aeroporto de `<chegada>` para os quais ainda há bilhetes disponíveis.
`/compra/<voo>/`	Faz uma compra de um ou mais bilhetes para o `<voo>`, populando as tabelas `<venda>` e `<bilhete>`. Recebe como argumentos o nif do cliente, e uma lista de pares (nome de passageiro, classe de bilhete) especificando os bilhetes a comprar.
`/checkin/<bilhete>/`	Faz o check-in de um bilhete, atribuindo-lhe automaticamente um assento da classe correspondente.

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

Os endpoints de compra e checkin devem devolver mensagens explícitas ou confirmando o sucesso da operação ou indicando porque motivo não foi possível realizá-la.

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.

Critérios de Avaliação

- Coerência dos *endpoints*
- Uso correto de transações
- Prevenção de injeção de SQL
- Uso correto de métodos e códigos de erro HTTP
- JSON bem estruturado
- Código da aplicação bem estruturado

4. Vistas [2 valores]

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

estatisticas_voos(*no_serie*, *hora_partida*, *cidade_partida*, *pais_partida*, *cidade_chegada*, *pais_chegada*, *ano*, *mes*, *dia_do_mes*, *dia_da_semana*, *passageiros_1c*, *passageiros_2c*, *assentos_1c*, *assentos_2c*, *vendas_1c, vendas_2c*)

em que:

- *no_serie*, *hora_partida*: correspondem aos atributos homónimos da tabela *voo*
- *cidade_partida*, *pais_partida*, *cidade_chegada*, *pais_chegada*: correspondem aos atributos *cidade* e *pais* da tabela *aeroporto*, para o aeroporto de *partida* e *chegada* do *voo*
- *ano, mes* e *dia_do_mes*, *dia_da_semana*: são derivados da *hora_partida*[1]
- *passageiros_1c*, *passageiros_2c*: correspondem ao número total de bilhetes vendidos para o voo, de primeira e segunda classe respectivamente
- *assentos_1c*, *assentos_2c*: correspondem ao número de assentos de primeira e segunda classe no avião que realiza o voo
- *vendas_1c*, *vendas_2c*: correspondem ao somatório total dos preços dos bilhetes vendidos para o voo, de primeira e segunda classe respectivamente

Critérios de Avaliação

- Coerência da vista
- Simplicidade do código

[1] Pode utilizar a função EXTRACT() para obter partes de timestamps.

5. Análise de Dados SQL e OLAP [5 valores]

Usando **apenas a vista *estatisticas_voos*** desenvolvida no ponto anterior, e **sem recurso ao operador LIMIT e com recurso ao operador WITH apenas se estritamente necessário**, apresente a consulta SQL mais sucinta para cada um dos seguintes objetivos analíticos da empresa. Pode usar agregações OLAP para os objetivos em que lhe parecer adequado.

1. Determinar a(s) rota(s) que tem/têm a maior procura para efeitos de aumentar a frequência de voos dessa(s) rota(s). Entende-se por rota um trajeto aéreo entre quaisquer duas cidades, independentemente do sentido (e.g., voos Lisboa-Paris e Paris-Lisboa contam para a mesma rota). Considera-se como indicador da procura de uma rota o preenchimento médio dos aviões (i.e., o rácio entre o número total de passageiros e a capacidade total do avião) no último ano.
2. Determinar as rotas pelas quais nos últimos 3 meses passaram todos os aviões da empresa, para efeitos de melhorar a gestão da frota.
3. Explorar a rentabilidade da empresa (vendas globais e por classe) nas dimensões espaço (global > pais > cidade, para a partida e chegada em simultâneo) e tempo (global > ano > mes > dia_do_mes), como apoio a um relatório executivo.
4. Descobrir se há algum padrão ao longo da semana no rácio entre passageiros de primeira e segunda classe, com drill down na dimensão espaço (global > pais > cidade), que justifique uma abordagem mais flexível à divisão das classes.

Critérios de Avaliação

- Coerência das soluções
- Simplicidade das soluções

6. Índices [3 valores]

É expectável que seja necessário executar consultas semelhantes ao **colectivo das consultas do ponto anterior** diversas vezes ao longo do tempo, e pretendemos otimizar o desempenho da **vista *estatisticas_voos*** para esse efeito. Crie sobre a vista o(s) índice(s) que achar mais indicados para fazer essa otimização, justificando a sua escolha com **argumentos teóricos** e com **demonstração prática** do ganho em eficiência do índice por meio do comando EXPLAIN ANALYSE. Deve procurar uma otimização coletiva das consultas, evitando criar índices excessivos, particularmente se estes trazem apenas ganhos incrementais a uma das consultas.

Critérios de Avaliação

- Utilidade dos índices
- Qualidade da justificação

Entrega

A submissão do projeto deve ser feita na forma de um arquivo zip de nome **`entrega-bd-02-GG.zip`**[2], 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 primeira célula com o **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)**. Deverá ainda preencher no Jupyter Notebook as respostas às perguntas das seguintes secções : 1. **Restrições de Integridade** 4. **Vistas** 5. **Análise de Dados SQL & OLAP** 6. **Índices** O ficheiro **"E2-report.ipynb"** pode ser importado para o ambiente de trabalho disponibilizado para as aulas de laboratório[3], que serve de ambiente de teste para as partes em SQL. Deve certificar-se que todo o código SQL é executável no ambiente de trabalho.
`data/`	Pasta com o(s) arquivo(s) .sql ou .txt correspondendo à secção 2. **Preenchimento da Base de Dados**
`app/`	Pasta com os ficheiros que compõem a aplicação web correspondendo à secção 3. **Desenvolvimento de Aplicação**

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

[2] ⚠ O formato do arquivo deve ser exclusivamente ZIP ou GZ. Outros formatos de arquivo não serão aceites.
[3] https://github.com/bdist/db-workspace

Anexo A - Esquema SQL

```
DROP TABLE IF EXISTS aeroporto CASCADE;
DROP TABLE IF EXISTS aviao CASCADE;
DROP TABLE IF EXISTS assento CASCADE;
DROP TABLE IF EXISTS voo CASCADE;
DROP TABLE IF EXISTS venda CASCADE;
DROP TABLE IF EXISTS bilhete CASCADE;

CREATE TABLE aeroporto(
	codigo CHAR(3) PRIMARY KEY CHECK (codigo ~ '^[A-Z]{3}$'),
	nome VARCHAR(80) NOT NULL,
	cidade VARCHAR(255) NOT NULL,
	pais VARCHAR(255) NOT NULL,
	UNIQUE (nome, cidade)
);

CREATE TABLE aviao(
	no_serie VARCHAR(80) PRIMARY KEY,
	modelo VARCHAR(80) NOT NULL
);

CREATE TABLE assento (
	lugar VARCHAR(3) CHECK (lugar ~ '^[0-9]{1,2}[A-Z]$'),
	no_serie VARCHAR(80) REFERENCES aviao,
	prim_classe BOOLEAN NOT NULL DEFAULT FALSE,
	PRIMARY KEY (lugar, no_serie)
);

CREATE TABLE voo (
	id SERIAL PRIMARY KEY,
	no_serie VARCHAR(80) REFERENCES aviao,
	hora_partida TIMESTAMP,
	hora_chegada TIMESTAMP,
	partida CHAR(3) REFERENCES aeroporto(codigo),
	chegada CHAR(3) REFERENCES aeroporto(codigo),
	UNIQUE (no_serie, hora_partida),
	UNIQUE (no_serie, hora_chegada),
	UNIQUE (hora_partida, partida, chegada),
	UNIQUE (hora_chegada, partida, chegada),
	CHECK (partida!=chegada),
	CHECK (hora_partida<=hora_chegada)
);

CREATE TABLE venda (
	codigo_reserva SERIAL PRIMARY KEY,
	nif_cliente CHAR(9) NOT NULL,
	balcao CHAR(3) REFERENCES aeroporto(codigo),
	hora TIMESTAMP
```

```
);

CREATE TABLE bilhete (
	id SERIAL PRIMARY KEY,
	voo_id INTEGER REFERENCES voo,
	codigo_reserva INTEGER REFERENCES venda,
	nome_passegeiro VARCHAR(80),
	preco NUMERIC(7,2) NOT NULL,
	prim_classe BOOLEAN NOT NULL DEFAULT FALSE,
	lugar VARCHAR(3),
	no_serie VARCHAR(80),
	UNIQUE (voo_id, codigo_reserva, nome_passegeiro),
	FOREIGN KEY (lugar, no_serie) REFERENCES assento
);
```