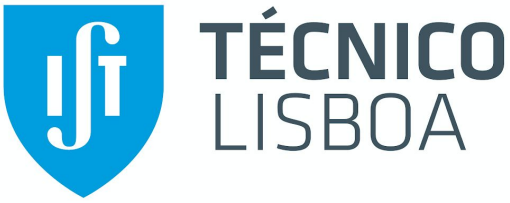

Projeto de Bases de Dados

Parte 4

Grupo 38

4ª Feira - 11:00H

Docente André Pereira

Henrique Dias - 189455 : 33.3(3)%
Isabel Soares - 189466 : 33.3(3)%
Rodrigo Sousa - 189535 : 33.3(3)%

1. **Restrições de Integridade**

RI-1:
```
CREATE OR REPLACE FUNCTION
caixa1_nao_sobrepoe_procedure ()
RETURNS TRIGGER
AS $$
BEGIN
       IF (EXISTS (
      SELECT *
      FROM anomalia_traducao
      WHERE id = new.id AND zona2 && new.zona
    )) THEN
      RAISE EXCEPTION 'A zona 1 sobrepõe-se a outra zona.';
    END IF;
    RETURN new;
END;
$$ LANGUAGE PLPGSQL;

CREATE OR REPLACE FUNCTION
caixa2_nao_sobrepoe_procedure ()
RETURNS TRIGGER
AS $$
BEGIN
       IF (EXISTS (
      SELECT *
      FROM anomalia
      WHERE id = new.id AND zona && new.zona2
    )) THEN
      RAISE EXCEPTION 'A zona 2 sobrepõe-se a outra zona.';
    END IF;
    RETURN new;
END;
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS caixa_nao_sobrepoe_zona1 ON anomalia;
DROP TRIGGER IF EXISTS caixa_nao_sobrepoe_zona2 ON anomalia_traducao;

CREATE TRIGGER caixa_nao_sobrepoe_zona1 BEFORE UPDATE ON anomalia
FOR EACH ROW EXECUTE PROCEDURE caixa1_nao_sobrepoe_procedure();
CREATE TRIGGER caixa_nao_sobrepoe_zona2 BEFORE INSERT OR UPDATE ON anomalia_traducao
FOR EACH ROW EXECUTE PROCEDURE caixa2_nao_sobrepoe_procedure();
```

RI-4:
```
CREATE OR REPLACE FUNCTION
check_utilizador_procedure ()
RETURNS TRIGGER
AS $$
BEGIN
       IF (NOT EXISTS (
      SELECT *
      FROM (
        SELECT email FROM utilizador_qualificado
          UNION ALL
        SELECT email FROM utilizador_regular
      ) s
      WHERE email = new.email
    )) THEN
      RAISE EXCEPTION 'O utilizador % tem que ser qualificado ou regular.', new.email;
    END IF;
    RETURN new;
END;
```

```
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS check_utilizador ON utilizador;

CREATE CONSTRAINT TRIGGER check_utilizador AFTER INSERT OR UPDATE OR DELETE ON utilizador
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE check_utilizador_procedure();
```

RI-5:
```
CREATE OR REPLACE FUNCTION
check_utilizador_qualificado_procedure ()
RETURNS TRIGGER
AS $$
BEGIN
       IF (EXISTS (
      SELECT email
      FROM utilizador_regular
      WHERE email = new.email
    )) THEN
      RAISE EXCEPTION 'O utilizador % já é um utilizador regular.', new.email;
    END IF;
    RETURN new;
END;
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS check_utilizador_qualificado ON utilizador_qualificado;

CREATE TRIGGER check_utilizador_qualificado BEFORE INSERT OR UPDATE ON utilizador_qualificado
FOR EACH ROW EXECUTE PROCEDURE check_utilizador_qualificado_procedure();
```

RI-6:
```
CREATE OR REPLACE FUNCTION
check_utilizador_regular_procedure ()
RETURNS TRIGGER
AS $$
BEGIN
       IF (EXISTS (
      SELECT email
      FROM utilizador_qualificado
      WHERE email = new.email
    )) THEN
      RAISE EXCEPTION 'O utilizador % já é um utilizador qualificado.', new.email;
    END IF;
    RETURN new;
END;
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS check_utilizador_regular ON utilizador_regular;

CREATE TRIGGER check_utilizador_regular BEFORE INSERT OR UPDATE ON utilizador_regular
FOR EACH ROW EXECUTE PROCEDURE check_utilizador_regular_procedure();
```

2. **<u>Índices</u>**

1.1 Não se verifica a necessidade de criar um index, visto que do ponto de perspetiva teórico iremos carregar mais de 10% da tabela, percorrendo-a praticamente na totalidade não tirando proveito de uma estrutura auxiliar, como a btree.

1.2 Criamos uma btree para a coluna *data_hora* de modo a tornar a procura por uma gama de valores de *timestamps* mais eficiente.

```
CREATE INDEX data_hora_index ON proposta_de_correcao
    USING BTREE(data_hora);
```

2. Criamos uma Hash Table para a coluna *anomalia_id* de forma a tornar a procura por anomalias com um determinado ID mais eficientes.

```
CREATE INDEX email_incidencia_index ON incidencia
    USING HASH(anomalia_id);
```

3.1 Não se verifica a necessidade de criar um index, visto que do ponto de perspetiva teórico iremos carregar mais de 10% da tabela, percorrendo-a praticamente na totalidade não tirando proveito de uma estrutura auxiliar, como a btree.

3.2 Criamos uma btree para a coluna *anomalia_id* de modo a tornar a procura por uma gama de valores de IDs de anomalias mais eficiente.

```
CREATE INDEX email_correcao_index ON correcao
    USING BTREE(anomalia_id);
```

4. Criamos duas btrees, uma para a coluna *ts* de modo a tornar a procura por uma gama de valores de *timestamps* mais eficiente e outra para a coluna *lingua* de modo a tornar a procura por uma lingua que tenha correspondência com o padrão fornecido. No último caso, funciona visto que o padrão da língua encontrar-se-á sempre no início, caso contrário não surtiria efeito. Em ambas, restringimos os índices através da cláusula *where* uma vez que só são estas as anomalias que serão acedidas em todos os cenários.

```
CREATE INDEX anomalia_timestamp ON anomalia USING BTREE(ts, lingua)
    WHERE tem_anomalia_redacao = True;
```

3. **Modelo Multidimensional**

```
DROP TABLE IF EXISTS d_utilizador CASCADE;
DROP TABLE IF EXISTS d_tempo CASCADE;
DROP TABLE IF EXISTS d_local CASCADE;
DROP TABLE IF EXISTS d_lingua CASCADE;
DROP TABLE IF EXISTS f_anomalia CASCADE;

CREATE TABLE d_utilizador (
  id_utilizador SERIAL NOT NULL PRIMARY KEY,
  email VARCHAR(80) NOT NULL,
  tipo VARCHAR(80) NOT NULL
);

CREATE TABLE d_tempo (
  id_tempo SERIAL NOT NULL PRIMARY KEY,
  dia INT NOT NULL,
  dia_da_semana INT NOT NULL,
  semana INT NOT NULL,
  mes INT NOT NULL,
  trimestre INT NOT NULL,
  ano INT NOT NULL
);

CREATE TABLE d_local (
  id_local SERIAL NOT NULL PRIMARY KEY,
  latitude INT NOT NULL,
  longitude INT NOT NULL,
  nome VARCHAR(80) NOT NULL
);
```

```
CREATE TABLE d_lingua (
  id_lingua SERIAL NOT NULL PRIMARY KEY,
  lingua VARCHAR(80) NOT NULL
);

CREATE TABLE f_anomalia (
  id_utilizador INT NOT NULL,
  id_tempo INT NOT NULL,
  id_local INT NOT NULL,
  id_lingua INT NOT NULL,
  tipo_anomalia VARCHAR(80) NOT NULL,
  com_proposta BOOLEAN NOT NULL,
  FOREIGN KEY (id_utilizador) REFERENCES d_utilizador(id_utilizador) ON DELETE CASCADE ON
UPDATE CASCADE,
  FOREIGN KEY (id_tempo) REFERENCES d_tempo(id_tempo) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (id_local) REFERENCES d_local(id_local) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (id_lingua) REFERENCES d_lingua(id_lingua) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY(id_utilizador, id_tempo, id_local, id_lingua)
);

INSERT INTO d_utilizador(email, tipo)
  SELECT email, 'regular' as tipo FROM utilizador_regular;

INSERT INTO d_utilizador(email, tipo)
  SELECT email, 'qualificado' as tipo FROM utilizador_qualificado;

INSERT INTO d_tempo(dia, dia_da_semana, semana, mes, trimestre, ano)
  SELECT EXTRACT(day FROM ts) AS dia,
    EXTRACT(dow FROM ts) AS dia_da_semana,
    EXTRACT(week FROM ts) AS semana,
    EXTRACT(month FROM ts) AS mes,
    FLOOR((EXTRACT(month FROM ts) - 1) / 4) + 1 AS trimestre,
    EXTRACT(year FROM ts) AS ano
  FROM anomalia
  ORDER BY dia, dia_da_semana, semana, mes, trimestre, ano;

INSERT INTO d_local(latitude, longitude, nome)
  SELECT latitude, longitude, nome FROM local_publico
  ORDER BY nome;

INSERT INTO d_lingua(lingua)
  SELECT lingua FROM anomalia
  UNION
  SELECT lingua2 AS lingua FROM anomalia_traducao
  ORDER BY lingua;

INSERT INTO f_anomalia(id_utilizador, id_tempo, id_local, id_lingua, tipo_anomalia,
com_proposta)
  SELECT id_utilizador, id_tempo, id_local, id_lingua, tipo_anomalia, com_proposta
  FROM (SELECT id AS anomalia_id,
        lingua,
        EXTRACT(day FROM ts) AS dia,
      EXTRACT(dow FROM ts) AS dia_da_semana,
      EXTRACT(week FROM ts) AS semana,
      EXTRACT(month FROM ts) AS mes,
      FLOOR((EXTRACT(month FROM ts) - 1) / 4) + 1 AS trimestre,
      EXTRACT(year FROM ts) AS ano,
```

```
    CASE
            WHEN tem_anomalia_redacao
              THEN 'redacao'
              ELSE 'traducao'
        END AS tipo_anomalia,
    CASE
      WHEN EXISTS (SELECT *
            from correcao c
            where c.anomalia_id = anomalia.id)
      THEN True
      ELSE False
    END AS com_proposta
    FROM anomalia) AS anomalias NATURAL JOIN
    incidencia NATURAL JOIN
    (SELECT id AS item_id,
      latitude, longitude
    FROM item) AS items NATURAL JOIN
    d_utilizador NATURAL JOIN
    d_lingua NATURAL JOIN
    d_local NATURAL JOIN
    d_tempo
ORDER BY anomalia_id;
```

4. **Data Analytics**

```
SELECT tipo_anomalia, lingua, dia_da_semana, COUNT(*)
FROM f_anomalia NATURAL JOIN d_lingua NATURAL JOIN d_tempo
GROUP BY CUBE(tipo_anomalia, lingua, dia_da_semana)
ORDER BY count DESC;
```