Instituto Superior Técnico

Base de Dados

Professor: Paulo Carreira

Projeto – Parte 4

Grupo 10 – BD8179577L02

Nome	Número	Esforço (Horas)	Contribuição (%)
Sara Machado	86923	10	33.33%
Rafael Figueiredo	90770	10	33.33%
Ricardo Grade	90774	10	33.33%

Restrições de Integridade:

RI - 1:

```
CREATE OR REPLACE FUNCTION check_box_overlaps_proc()
RETURNS TRIGGER AS $$
  BEGIN
    IF EXISTS (
      SELECT zona
      FROM anomalia
      WHERE anomalia_id = NEW.anomalia_id AND
          zona && NEW.zona2
    ) THEN
      RAISE EXCEPTION 'Zona2 Invalida: %', NEW.zona2
      USING HINT = 'Verifique os limites da Zona';
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;


CREATE TRIGGER check_box_overlaps BEFORE INSERT OR
        UPDATE ON anomalia_traducao
FOR EACH ROW EXECUTE PROCEDURE
check_box_overlaps_proc();
```

RI - 4:

```
CREATE OR REPLACE FUNCTION check_user_email_proc()
RETURNS TRIGGER AS $$
  BEGIN
    IF NOT EXISTS (
      SELECT email
      FROM utilizador_qualificado
      WHERE email = NEW.email
      UNION
      SELECT email
      FROM utilizador_regular
      WHERE email = NEW.email
    ) THEN
      RAISE EXCEPTION 'Email Invalido: %', NEW.email
      USING HINT = 'Verifique o Email';
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;


CREATE TRIGGER check_user_email BEFORE INSERT OR
        UPDATE ON utilizador
FOR EACH ROW EXECUTE PROCEDURE
check_user_email_proc();
```

RI - 5:

```
CREATE OR REPLACE FUNCTION
check_qual_user_email_proc()

RETURNS TRIGGER AS $$

  BEGIN

    IF EXISTS (

      SELECT email

      FROM utilizador_regular

      WHERE email = NEW.email

    ) THEN

      RAISE EXCEPTION 'Email Invalido: %', NEW.email

      USING HINT = 'Utilizador so pode ser de um tipo';

    END IF;

    RETURN NEW;

  END;

  $$ LANGUAGE plpgsql;


CREATE TRIGGER check_qual_user_email BEFORE INSERT
OR UPDATE ON utilizador_qualificado

FOR EACH ROW EXECUTE PROCEDURE
check_qual_user_email_proc();
```

RI - 6:

```
CREATE OR REPLACE FUNCTION
check_reg_user_email_proc()

RETURNS TRIGGER AS $$

  BEGIN

    IF EXISTS (

      SELECT email

      FROM utilizador_qualificado

      WHERE email = NEW.email

    ) THEN

      RAISE EXCEPTION 'Email Invalido: %', NEW.email

      USING HINT = 'Utilizador so pode ser de um tipo';

    END IF;

    RETURN NEW;

  END;

  $$ LANGUAGE plpgsql;


CREATE TRIGGER check_reg_user_email BEFORE INSERT OR
UPDATE ON utilizador_regular

FOR EACH ROW EXECUTE PROCEDURE
check_reg_user_email_proc();
```

RI – (4, 5, 6):

De modo a que a inserção de um utilizador seja possível, foi criada uma função para tal "*create_user*". No início desta, desativa-se, temporariamente (até à função retornar), a verificação da *foreign key(email)* presente no utilizador qualificado e regular, para tal, foi necessário dar um nome a esta constraint "*fk_user*" e defini-la como *DEFERRABLE INITIALLY IMMEDIATE*, para que o seu estado possa ser alterado para *DEFERRED*.

Função:

```
CREATE OR REPLACE FUNCTION create_user(IN mail varchar(254), IN pass varchar(40), IN isQualified BOOLEAN)

RETURNS VOID AS $$

  BEGIN

    SET CONSTRAINTS fk_user DEFERRED;

    IF isQualified THEN

      INSERT INTO utilizador_qualificado VALUES(mail);

    ELSE

      INSERT INTO utilizador_regular VALUES(mail);

    END IF;

    INSERT INTO utilizador VALUES(mail, pass);

  END;

  $$ LANGUAGE plpgsql;
```

Modificação na tabela "*utilizador qualificado*" e "*utilizador regular*":

CONSTRAINT fk_user foreign key(email)
references utilizador(email) ON UPDATE CASCADE ON DELETE CASCADE DEFERRABLE INITIALLY IMMEDIATE

Índices:

1.1.

Tendo como base o facto de apenas existir dados unclustered em PSQL, e em 80% das invocações a query devolver > 10% do total de registos da tabela, decidimos que não havia necessidade de criar um index, visto que o número de I/O's continuara a ser bastante elevado.

Pois com > 10% dos registos da tabela a serem devolvidos, existe uma grande probabilidade de carregar o mesmo número de páginas, criando ou não um index.

1.2.

Tendo como base o facto de apenas existir dados unclustered em PSQL, e em 80% das invocações a query devolver < 0.001% (< 10%) do total de registos da tabela, decidimos utilizar uma B+ Tree como estrutura de indexação.

Pois, ao contrário da alínea anterior, tendo em conta a % de registos da tabela a serem devolvidos, com a utilização de um index, o número de I/O's será bastante mais reduzido.

```
CREATE INDEX idx_data_hora ON proposta_de_correcao
    USING btree (data_hora);
```

2.

Apesar da pesquisa efetuada ser feita através de uma primary key (anomalia_id), e sabendo que essa pesquisa, em PSQL, é feita numa B+ Tree, e como estamos a fazer uma pesquisa por igualdade, esta estrutura de indexação não nos parece ser a mais eficiente, pelo que decidimos que utilizar uma Hash seria o mais apropriado, dai a criação deste index.

```
CREATE INDEX idx_anomalia_id ON incidencia
    USING hash (anomalia_id);
```

3.1. e 3.2.

Ao termos uma primary key composta na tabela correção, do qual a anomalia_id faz parte, isto permite-nos que simplesmente alterando a ordem pela qual os campos da primary key são declarados (email, nro, anomalia_id) -> (anomalia_id, email, nro) otimiza a procura em B+ Tree pela anomalia_id, e tendo em conta que a query seleciona apenas o email, e este faz parte dos campos da primary key, então nem e necessário fazer um acesso a tabela, tornando assim a query o mais eficiente, com o menor número de I/O's possível.

4.

Como a query seleciona apenas as linhas que tem_anomalia_redacao = TRUE, podemos criar um parcial composite index com o ts e a lingua, sendo que os comparamos num certo range (*BETWEEN* ou *LIKE*), daí usarmos uma estrutura de indexação B+ Tree.

```
CREATE INDEX idx_ts ON anomalia
    USING btree (ts, lingua)
    WHERE tem_anomalia_redacao IS TRUE;
```

Modelo Multidimensional:

```
DROP TABLE f_anomalia;
DROP TABLE d_utilizador;
DROP TABLE d_tempo;
DROP TABLE d_local;
DROP TABLE d_lingua;

CREATE TABLE d_utilizador (
    id_utilizador serial not null,
    email varchar(254) not null,
    tipo varchar(11) not null,
    primary key(id_utilizador)
);

CREATE TABLE d_tempo (
    id_tempo serial not null,
    dia integer not null,
    dia_da_semana integer not null,
    semana integer not null,
    mes integer not null,
    trimestre integer not null,
    ano integer not null,
    primary key(id_tempo)
);

CREATE TABLE d_local (
    id_local serial not null,
    latitude decimal(8, 6) not null,
    longitude decimal(9, 6) not null,
    nome varchar(200) not null,
    primary key(id_local)
);

CREATE TABLE d_lingua (
    id_lingua serial not null,
    lingua char(3) not null,
    primary key(id_lingua)
);

CREATE TABLE f_anomalia (
    id_utilizador serial not null,
    id_tempo serial not null,
    id_local serial not null,
    id_lingua serial not null,
    tipo_anomalia varchar(8) not null,
    com_proposta boolean not null,
    primary key (id_utilizador, id_tempo, id_local, id_lingua),
    foreign key(id_tempo) references d_tempo(id_tempo),
    foreign key(id_local) references d_local(id_local),
    foreign key(id_lingua) references d_lingua(id_lingua)
);
```

Populate d_tempo:

```
CREATE OR REPLACE FUNCTION insert_d_tempo()
RETURNS VOID AS $$
    DECLARE ts timestamp;
    BEGIN
        ts = '2000-01-01 00:00:00';
        WHILE ts < '2025-01-01 00:00:00' LOOP
            INSERT INTO d_tempo(dia, dia_da_semana, semana,
                                mes, trimestre, ano)
            VALUES (
                date_part('day', ts),
                date_part('dow', ts) + 1,
                date_part('week', ts),
                date_part('month', ts),
                date_part('month', ts) / 3 + 1,
                date_part('year', ts)
                );
            ts = ts + INTERVAL '1 day';
        END LOOP;
    END;
    $$ LANGUAGE plpgsql;

SELECT insert_d_tempo();
```

Populate d_utilizador:

```
INSERT INTO d_utilizador(email, tipo)
  SELECT email, 'Qualificado' AS tipo
  FROM utilizador_qualificado
  UNION
  SELECT email, 'Regular' AS tipo
  FROM utilizador_regular;
```

Populate d_local:

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

Populate d_lingua:

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

Populate f_anomalia:

```
CREATE OR REPLACE FUNCTION check_corr(
        IN mail varchar(254))
RETURNS BOOLEAN AS $$
  BEGIN
    IF EXISTS (
      SELECT email
      FROM proposta_de_correcao
      WHERE email = mail
    ) THEN RETURN TRUE;
    END IF;
    RETURN FALSE;
  END;
  $$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION anomalia_type(
        IN is_redacao BOOLEAN)
RETURNS VARCHAR(8) AS $$
  BEGIN
    IF is_redacao THEN
      RETURN 'Redacao';
    END IF;
    RETURN 'Traducao';
  END;
  $$ LANGUAGE plpgsql;
```

```
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 email,
      date_part('day', ts) AS dia,
      date_part('month', ts) AS mes,
      date_part('year', ts) AS ano,
      latitude,
      longitude,
      lingua,
      anomalia_type(tem_anomalia_redacao) AS
        tipo_anomalia,
      check_corr(email) AS com_proposta
    FROM utilizador
      NATURAL JOIN incidencia
      NATURAL JOIN item
      NATURAL JOIN anomalia
    ) AS data_table
    NATURAL JOIN d_utilizador
    NATURAL JOIN d_tempo
    NATURAL JOIN d_local
    NATURAL JOIN d_lingua;
```

Data Analytics:

```
SELECT COALESCE(CAST(tipo_anomalia AS varchar), 'TODOS OS TIPOS') AS tipo_anomalia,
        COALESCE(CAST(lingua AS varchar), 'TODAS AS LINGUAS') AS lingua,
        COALESCE(CAST(dia_da_semana AS varchar), 'TODOS OS DIAS') AS dia_da_semana,
        count AS total
FROM (
  SELECT tipo_anomalia, lingua, dia_da_semana, COUNT(*)
  FROM f_anomalia
    NATURAL JOIN d_lingua
    NATURAL JOIN d_tempo
  GROUP BY tipo_anomalia, lingua, dia_da_semana
  UNION
  SELECT tipo_anomalia, lingua, NULL, COUNT(*)
  FROM f_anomalia
    NATURAL JOIN d_lingua
    NATURAL JOIN d_tempo
  GROUP BY tipo_anomalia, lingua
  UNION
  SELECT tipo_anomalia, NULL, NULL, COUNT(*)
  FROM f_anomalia
    NATURAL JOIN d_lingua
    NATURAL JOIN d_tempo
  GROUP BY tipo_anomalia
  UNION
  SELECT NULL, NULL, NULL, COUNT(*)
  FROM f_anomalia
    NATURAL JOIN d_lingua
    NATURAL JOIN d_tempo
  ORDER BY tipo_anomalia, lingua, dia_da_semana
  ) AS T;
```