Projeto de Bases de Dados
Parte 4

Grupo 16
Turno BD225179577L05 – 2ªf 14h-15h30
Prof. Duarte Alexandre Galvão

Número	Nome	Esforço (horas)	Esforço (%)
89476	João Fonseca	6	33
89544	Tiago Pires	6	33
89552	Tomás Lopes	6	33

Restrições de Integridade (triggers.sql)

```
DROP FUNCTION IF EXISTS update_anomalia_proc() CASCADE;
DROP FUNCTION IF EXISTS insert_anomalia_traducao_proc() CASCADE;
DROP FUNCTION IF EXISTS update_anomalia_traducao_proc() CASCADE;
DROP FUNCTION IF EXISTS insert_user_proc() CASCADE;
DROP FUNCTION IF EXISTS insert_reg_user_proc() CASCADE;
DROP FUNCTION IF EXISTS insert_qual_user_proc() CASCADE;
DROP FUNCTION IF EXISTS update_reg_user_proc() CASCADE;
DROP FUNCTION IF EXISTS update_qual_user_proc() CASCADE;
DROP FUNCTION IF EXISTS delete_reg_user_proc() CASCADE;
DROP FUNCTION IF EXISTS delete_qual_user_proc() CASCADE;

/* RI-1 */

CREATE OR REPLACE FUNCTION update_anomalia_proc() RETURNS TRIGGER AS
$$
DECLARE b BOX;
BEGIN
    SELECT zona2 INTO b FROM anomalia_traducao WHERE id=new.id;
    IF b IS NOT NULL AND b && new.zona THEN
        RAISE EXCEPTION 'Zona AND zona2 must not overlap';
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_anomalia BEFORE UPDATE ON anomalia
FOR EACH ROW EXECUTE PROCEDURE update_anomalia_proc();


CREATE OR REPLACE FUNCTION insert_anomalia_traducao_proc() RETURNS TRIGGER AS
$$
DECLARE b BOX;
BEGIN
    IF new.id NOT IN (SELECT id FROM anomalia) THEN
        RAISE EXCEPTION 'Please insert an anomalia with that id first';
    ELSIF new.id NOT IN (SELECT id FROM anomalia WHERE id=new.id AND tem_anomalia_redacao=false)
    THEN
        RAISE EXCEPTION 'The anomalia id introduced does not require zona2/lingua2 values';
    else
        SELECT zona INTO b FROM anomalia WHERE id=new.id;
        IF b && new.zona2 THEN
            RAISE EXCEPTION 'Zona AND zona2 must not overlap';
        END IF;
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_anomalia_traducao BEFORE INSERT ON anomalia_traducao
FOR EACH ROW EXECUTE PROCEDURE insert_anomalia_traducao_proc();


CREATE OR REPLACE FUNCTION update_anomalia_traducao_proc() RETURNS TRIGGER AS
$$
DECLARE b BOX;
BEGIN
    SELECT zona INTO b FROM anomalia WHERE id=new.id;
    IF b && new.zona2 THEN
        RAISE EXCEPTION 'Zona AND zona2 must not overlap';
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_anomalia_traducao BEFORE UPDATE ON anomalia_traducao
FOR EACH ROW EXECUTE PROCEDURE update_anomalia_traducao_proc();
```

```
/* RI-4, RI-5, RI-6 */

/*
A logica para o tratamento de utilizadores e:
- Inserir um novo utilizador (na tabela utilizador) coloca o mesmo na tabela utilizador_regular;
- Inserir um utilizador na tabela utilizador_qualificado retira-o da tabela utilizador_regular e
vice-versa: ou seja, esta e a forma de trocar o estatuto de um utilizador. Em alternativa,
tambem se pode trocar o estatuto de um utilizador regular apagando-o da tabela
utilizador_regular (e inserido automaticamente na tabela utilizador_qualificado) e vice-versa;
- Atualizar qualquer entrada da tabela utilizador_regular levanta uma excecao quando o novo
e-mail ja existe na tabela utilizador_qualificado e vice-versa: evita que um utilizador possa
estar nas tabelas utilizador_regular e utilizador_qualificado ao mesmo tempo);
- As atualizacoes e remocoes efetuadas na tabela utilizador nao tem regras especiais (as
alteracoes sao propagadas para as restantes tabelas por cascade, como definido no ficheiro
schema.sql da terceira entrega).
*/

CREATE OR REPLACE FUNCTION insert_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    INSERT INTO utilizador_regular VALUES (new.email);
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_user AFTER INSERT ON utilizador
FOR EACH ROW EXECUTE PROCEDURE insert_user_proc();


CREATE OR REPLACE FUNCTION insert_reg_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF new.email IN (SELECT email FROM utilizador_qualificado) THEN
        DELETE FROM utilizador_qualificado WHERE email=new.email;
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_reg_user AFTER INSERT ON utilizador_regular
FOR EACH ROW EXECUTE PROCEDURE insert_reg_user_proc();


CREATE OR REPLACE FUNCTION insert_qual_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF new.email IN (SELECT email FROM utilizador_regular) THEN
        DELETE FROM utilizador_regular WHERE email=new.email;
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_qual_user AFTER INSERT ON utilizador_qualificado
FOR EACH ROW EXECUTE PROCEDURE insert_qual_user_proc();


CREATE OR REPLACE FUNCTION update_reg_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF new.email IN (SELECT email FROM utilizador_qualificado) THEN
        RAISE EXCEPTION 'Please update the table utilizador to update user emails.';
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_reg_user BEFORE UPDATE ON utilizador_regular
FOR EACH ROW EXECUTE PROCEDURE update_reg_user_proc();
```

```
CREATE OR REPLACE FUNCTION update_qual_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF new.email IN (SELECT email FROM utilizador_regular) THEN
        RAISE EXCEPTION 'Please update the table utilizador to update user emails.';
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_qual_user BEFORE UPDATE ON utilizador_qualificado
FOR EACH ROW EXECUTE PROCEDURE update_qual_user_proc();


CREATE OR REPLACE FUNCTION delete_reg_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF (old.email NOT IN (SELECT email FROM utilizador_qualificado)
    AND old.email IN (SELECT email FROM utilizador)) THEN
        INSERT INTO utilizador_qualificado values (old.email);
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_reg_user AFTER DELETE ON utilizador_regular
FOR EACH ROW EXECUTE PROCEDURE delete_reg_user_proc();


CREATE OR REPLACE FUNCTION delete_qual_user_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF (old.email NOT IN (SELECT email FROM utilizador_regular)
    AND old.email IN (SELECT email FROM utilizador)) THEN
        INSERT INTO utilizador_regular values (old.email);
    END IF;
    RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_qual_user AFTER DELETE ON utilizador_qualificado
FOR EACH ROW EXECUTE PROCEDURE delete_qual_user_proc();
```

Índices (indexes.sql)

```
DROP INDEX IF EXISTS data_hora__proposta_de_correcao_idx;
DROP INDEX IF EXISTS anomalia_id__incidencia_idx;
DROP INDEX IF EXISTS ts_tem_anomalia_redacao_lingua__anomalia_idx;

/*
--1.1
E necessario analisar muitos registos e visto que a dimensao das tabelas ultrapassa em varias
ordens de grandeza a memoria disponivel, o acesso ao disco vai ser muito frequente, logo nao e
necessario criar um index.

--1.2
Como a query filtra com o operador between, cria-se um btree index.
*/

CREATE INDEX data_hora__proposta_de_correcao_idx ON proposta_de_correcao USING btree
(data_hora);

/*
--2
Ja existe um index btree para a coluna anomalia_id devido a esta ser chave primaria da
incidencia, mas quando possivel deve-se criar um hash index pois este e mais eficiente para
igualdades.
*/
```

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

/*
--3.1
E necessario analisar muitos registos e visto que a dimensao das tabelas ultrapassa em varias
ordens de grandeza a memoria disponivel, o acesso ao disco vai ser muito frequente. Alem disto
ja existe um index associado a 'anomalia_id' porque a coluna faz parte da chave primaria da
tabela 'correcao', logo nao e necessario criar um index.

--3.2
Como a coluna 'anomalia_id' faz parte da chave primaria da tabela 'correcao' existe um index
associado a esta coluna e como esta e o primeiro elemento da chave, este index pode ser
utilizado para a query em questao.

--4
A ordem de filtragem das colunas e 'tem_anomalia_redacao' depois 'lingua' e por fim 'ts'.
A flag 'tem_anomalia_redacao' esta em primeiro lugar porque so existem dois valores possiveis
(True e False), de seguida esta a coluna 'lingua' porque apenas existem cerca de 7000 linguas
faladas em todo o mundo, um numero bastante insignificante em termos de afetar a performance de
queries. Por fim filtra-se pelo intervalo de tempo fornecido pelo user, este pode ser muito ou
pouco abrangente.
*/

CREATE INDEX ts_tem_anomalia_redacao_lingua__anomalia_idx ON anomalia USING btree
(tem_anomalia_redacao,lingua,ts);
```

Modelo Multidimensional (star.sql)

```
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,
    email VARCHAR(100),
    tipo VARCHAR(20),
    CONSTRAINT pk_d_utilizador PRIMARY KEY(id_utilizador)
);

CREATE TABLE d_tempo (
    id_tempo SERIAL NOT NULL,
    dia INTEGER,
    dia_da_semana VARCHAR(20),
    semana INTEGER,
    mes INTEGER,
    trimestre INTEGER,
    ano INTEGER,
    CONSTRAINT pk_d_tempo PRIMARY KEY(id_tempo)
);

CREATE TABLE d_local (
    id_local SERIAL NOT NULL,
    latitude NUMERIC(8,6),
    longitude NUMERIC(9,6),
    nome VARCHAR(100),
    CONSTRAINT pk_d_local PRIMARY KEY(id_local)
);

CREATE TABLE d_lingua (
    id_lingua SERIAL NOT NULL,
    lingua VARCHAR(100),
    CONSTRAINT pk_d_lingua PRIMARY KEY(id_lingua)
);
```

```
CREATE TABLE f_anomalia (
    id_utilizador INTEGER,
    id_tempo INTEGER,
    id_local INTEGER,
    id_lingua INTEGER,
    tipo_anomalia VARCHAR(20),
    com_proposta BOOLEAN,
    CONSTRAINT pk_f_anomalia PRIMARY KEY(id_utilizador, id_tempo, id_local, id_lingua),
    CONSTRAINT fk_f_anomalia_d_utilizador FOREIGN KEY(id_utilizador)
        REFERENCES d_utilizador(id_utilizador) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_f_anomalia_d_tempo FOREIGN KEY(id_tempo)
        REFERENCES d_tempo(id_tempo) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_f_anomalia_d_local FOREIGN KEY(id_local)
        REFERENCES d_local(id_local) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_f_anomalia_d_lingua FOREIGN KEY(id_lingua)
        REFERENCES d_lingua(id_lingua) ON DELETE CASCADE ON UPDATE CASCADE
);


INSERT INTO d_utilizador (email, tipo)
SELECT DISTINCT email,
CASE
    WHEN EMAIL IN (SELECT email FROM utilizador_regular) THEN 'Regular'
    ELSE 'Qualificado'
END
FROM incidencia
ORDER BY email ASC;


INSERT INTO d_tempo (dia, dia_da_semana, semana, mes, trimestre, ano)
SELECT DISTINCT
    EXTRACT(DAY FROM ts) AS D,
    (SELECT CASE
        WHEN EXTRACT(DOW FROM ts)=1 THEN 'Segunda-Feira'
        WHEN EXTRACT(DOW FROM ts)=2 THEN 'Terca-Feira'
        WHEN EXTRACT(DOW FROM ts)=3 THEN 'Quarta-Feira'
        WHEN EXTRACT(DOW FROM ts)=4 THEN 'Quinta-Feira'
        WHEN EXTRACT(DOW FROM ts)=5 THEN 'Sexta-Feira'
        WHEN EXTRACT(DOW FROM ts)=6 THEN 'Sabado'
        WHEN EXTRACT(DOW FROM ts)=0 THEN 'Domingo'
    END),
    EXTRACT(WEEK FROM ts),
    EXTRACT(MONTH FROM ts) AS M,
    CEILING(EXTRACT(MONTH FROM ts)/3),
    EXTRACT(YEAR FROM ts) AS Y
FROM anomalia
ORDER BY Y ASC, M ASC, D ASC;

INSERT INTO d_local (latitude, longitude, nome)
SELECT DISTINCT latitude, longitude,
    (SELECT nome FROM local_publico
     WHERE item.latitude=local_publico.latitude AND
     item.longitude=local_publico.longitude) AS R
FROM item
ORDER BY R ASC;


INSERT INTO d_lingua (lingua)
SELECT DISTINCT lingua
FROM anomalia
ORDER BY lingua ASC;
```

```
INSERT INTO f_anomalia
SELECT
    (SELECT id_utilizador FROM d_utilizador
      WHERE incidencia.email=d_utilizador.email
    ) AS A,
    (SELECT id_tempo FROM d_tempo
     WHERE dia=EXTRACT(DAY FROM ts) AND mes=EXTRACT(MONTH FROM ts) AND ano=EXTRACT(YEAR FROM ts)
    ) AS B,
    (SELECT id_local FROM d_local
     WHERE item.latitude=d_local.latitude AND item.longitude=d_local.longitude
    ) AS C,
    (SELECT id_lingua FROM d_lingua
     WHERE d_lingua.lingua=anomalia.lingua
    ) AS D,
    (SELECT CASE
        WHEN tem_anomalia_redacao=true THEN 'Redacao'
        ELSE 'Traducao'
     END
     FROM anomalia
     WHERE anomalia_id=anomalia.id
    ) AS E,
    (SELECT CASE
        WHEN anomalia_id IN (SELECT anomalia_id FROM correcao) THEN true
        ELSE false
     END
     FROM incidencia
     WHERE anomalia_id=anomalia.id
    ) AS F
FROM anomalia,item,incidencia
WHERE anomalia_id=anomalia.id AND item.id=item_id
ORDER BY A ASC, B ASC, C ASC, D ASC, E ASC, F DESC;
```

Data Analytics (olap.sql)

```
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 tipo_anomalia ASC, lingua ASC, CASE
   WHEN dia_da_semana = 'Segunda-Feira' THEN 1
   WHEN dia_da_semana = 'Terca-Feira' THEN 2
   WHEN dia_da_semana = 'Quarta-Feira' THEN 3
   WHEN dia_da_semana = 'Quinta-Feira' THEN 4
   WHEN dia_da_semana = 'Sexta-Feira' THEN 5
   WHEN dia_da_semana = 'Sabado' THEN 6
   WHEN dia_da_semana = 'Domingo' THEN 7
   ELSE 8
END ASC;
```