**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;