Projeto de Bases de Dados – Parte 4

Ana Cláudia Madeira David, 90702

João Diogo Guerra Veloso, 90733

João Maria Martins Catarino Galamba de Oliveira, 90735

	Percentagem relativa	Esforço (em horas)
Ana David	33,3%	8
João Veloso	33,3%	8
João Galamba	33,3%	8

Turno BD8179577L02, Grupo 3

Prof. Paulo Carreira

Restrições de Integridade

RI-1 A zona da anomalia_tradução não se pode sobrepor à zona da anomalia correspondente.

```
create or replace function check_zona() returns trigger as $$
	begin
		if (NEW.zona2 && (
			SELECT zona
			FROM Anomalia
			WHERE idAnomalia = NEW.idAnomaliaTraducao)
		) then
			raise exception ‘zona2 de Anomalia de Traducao nao se pode sobrepor
a zona da Anomalia correspondente’
		end if;
		return NEW;
	end
$$ language plpgsql;

create trigger check_zona_constraint before insert or update AnomaliaTraducao for each row
execute procedure check_zona();
```

RI-4 email de utilizador tem de figurar em utilizador_qualificado ou utilizador_regular

```
create or replace function check_email_existance(check boolean) returns trigger as $$
	begin
		if check then
			if exists(
				SELECT email
				FROM UtilizadorQualificado
				WHERE email = NEW.email
			) then
				return NEW;
			end if;
		else
			if exists(
				SELECT email
				FROM UtilizadorRegular
				WHERE email = NEW.email
			) then
				return NEW;
			end if;
		end if;
		raise exception ‘email de Utilizador tem de figurar em UtilizadorQualificado ou
UtilizadorRegular’
	end
$$ language plpgsql;
```

```
create trigger check_email_existance_constraint before insert or update Utilizador for each
row execute procedure check_email_existance();
```

RI-5 email não pode figurar em utilizador_regular

```
create or replace function check_emailregular() returns trigger as $$
	begin
		if exists(
			SELECT email
			FROM UtilizadorRegular NATURAL JOIN UtilizadorQualificado
		) then
			raise exception 'Se email ja existe em Utilizador Qualificado, tambem
nao pode existir em Utilizador Regular.';
		end if;
	end
$$ language plpgsql;

create trigger check_emailregular_constraint before insert or update UtilizadorQualificado for
each row execute procedure check_emailregular();
```

RI-6 email não pode figurar em utilizador_qualificado

```
create or replace function check_emailqualificado() returns trigger as $$
	begin
		if exists(
			SELECT email
			FROM UtilizadorQualificado NATURAL JOIN UtilizadorRegular
		) then
			raise exception 'Se email ja existe em Utilizador Regular, tambem nao
pode existir em Utilizador Qualificado.';
		end if;
	end
$$ language plpgsql;

create trigger check_emailqualificado_constraint before insert or update UtilizadorRegular for
each row execute procedure check_emailqualificado();
```

Índices

1.1 Para esta *query*, não achamos necessária a criação de um *index*. Tiramos esta conclusão devido a duas razões.

Como em 80% das invocações, a *query* devolve mais do que 10% do total de registos da tabela Proposta de Correção, estamos numa situação em que a criação do *index* não será eficiente.

Tendo em conta os dados fornecidos sobre a memória disponível e o total de registos originados pela *query*, retiramos que o resultado desta é superior à memória, deste modo, não existe vantagem em ter um *index* pois não irá melhorar significativamente o tempo de acesso.

1.2 CREATE INDEX data_hora_idx ON proposta_de_correcao USING btree(data_hora);

Este *index* seria útil pois *indexs* que usem *b-tree methods* são ótimos para otimização de *queries* com operadores do tipo “>”, “<”, “<=”, “>=”, “BETWEEN” ou “IN”.

2. CREATE INDEX anom_id_idx ON incidência USING hash(anomalia_id);

Este *index* otimizará a *query* por o hash ser indicado para otimizar *queries* que recorram ao operador “=”, como é o caso.

3.1 Para esta *query*, não achamos necessária a criação de um *index*. Para otimização desta, basta uma alteração no *schema.sql* de modo a que a chave primária seja (idAnomalia, email, nro) em vez de (email, nro, idAnomalia). Assim, a ordenação estará ela própria a otimizar a *query*, sem necessidade de se recorrer a um *index*.

3.2 CREATE INDEX anom_id_idx ON correcao USING btree(anomalid_id);

Este *index* provar-se-á útil devido à *query* recorrer ao operador “>”, estando na mesma situação do *index* criado na alínea 1.2 .

4. CREATE INDEX tem_anom_red_idx ON anomalia USING btree(ts, language) WHERE tem_anomalia_redacao IS True;

Devido à complexidade desta *query*, recorre-se a um *Partial Index*. Neste, conseguimos restringir os operadores a otimizar pelo *index* aos mesmos apresentados na alínea 1.2, criando o *index* só para entradas que tenham a coluna tem_anomalia_redacao a True. Na ordenação, devido ao *timestamp* ser mais específico (visto língua poder aparecer mais vezes repetido), ordena-se por *timestamp* primeiro.

Modelo Multidimensional

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

CREATE TABLE d_utilizador(
        id_utilizador SERIAL NOT NULL,
        email VARCHAR(254),
        tipo BIT NOT NULL,
        CONSTRAINT pk_d_utilizador 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,
	CONSTRAINT pk_d_tempo 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,
	CONSTRAINT pk_d_local PRIMARY KEY(id_local)
);

CREATE TABLE d_lingua(
	id_lingua SERIAL NOT NULL,
	lingua VARCHAR(3) NOT NULL,
	CONSTRAINT pk_d_lingua PRIMARY KEY(id_lingua)
);

CREATE TABLE f_anomalia(
	id_utilizador SERIAL,
	id_tempo SERIAL,
	id_local SERIAL,
	id_lingua SERIAL,
	tipo_anomalia BIT NOT NULL,
	com_proposta BIT NOT NULL,
	CONSTRAINT pk_f_anomalia PRIMARY KEY(id_utilizador, id_tempo, id_local, id_lingua),
	CONSTRAINT fk_anomalia_utilizador FOREIGN KEY(id_utilizador) REFERENCES
d_utilizador(id_utilizador) ON DELETE CASCADE,
	CONSTRAINT fk_anomalia_tempo FOREIGN KEY(id_tempo) REFERENCES
d_tempo(id_tempo) ON DELETE CASCADE,
	CONSTRAINT fk_anomalia_local FOREIGN KEY(id_local) REFERENCES d_local(id_local)
ON DELETE CASCADE,
	CONSTRAINT fk_anomalia_lingua FOREIGN KEY(id_lingua) REFERENCES
d_lingua(id_lingua) ON DELETE CASCADE
);

CREATE OR REPLACE FUNCTION get_time() RETURNS VOID AS $$
	DECLARE
		mind DATE;
		maxd DATE;
	BEGIN
		SELECT min(ts)::timestamp::date INTO mind
		FROM Anomalia;
		SELECT max(ts)::timestamp::date INTO maxd
		FROM Anomalia;

		WHILE mind <= maxd LOOP
```

```
                INSERT INTO d_tempo(dia, dia_da_semana, semana, mes, trimestre, ano)
VALUES (EXTRACT(day from mind), EXTRACT(DOW from mind), EXTRACT(week from mind),
EXTRACT(month from mind), EXTRACT(quarter from mind), EXTRACT(year from mind));
                mind := mind + interval '1 day';
                END LOOP;
        END;
$$ language plpgsql;

INSERT INTO d_utilizador(email, tipo)
SELECT email, B'0'
FROM UtilizadorRegular;

INSERT INTO d_utilizador(email, tipo)
SELECT email, B'1'
FROM UtilizadorQualificado;

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

INSERT INTO d_lingua(lingua)
SELECT lingua
FROM Anomalia;

IINSERT INTO d_lingua(lingua)
SELECT lingua2
FROM AnomaliaTraducao;

SELECT get_time();

INSERT INTO f_anomalia (id_utlizador, id_tempo, id_local, id_lingua, tipo_anomalia,
com_proposta)

SELECT (
        SELECT id_utilizador
        FROM d_utilizador NATURAL JOIN Incidencia
        WHERE Incidencia.email = d_utilizador.email;
) AS id_utlizador, (
        SELECT id_tempo
        FROM d_tempo NATURAL JOIN Anomalia
        WHERE EXTRACT(DAY FROM Anomalia.ts) = d_tempo.dia AND EXTRACT(MONTH FROM
Anomalia.ts) = d_tempo.mes AND EXTRACT(YEAR FROM Anomalia.ts) = d_tempo.ano;
) AS id_tempo, (
        SELECT id_local
        FROM d_local NATURAL JOIN Item
        WHERE d_local.latitude = Item.latitude AND d_local.longitude = Item.longitude;
) AS id_local, (
        SELECT id_lingua
        FROM d_lingua NATURAL JOIN Anomalia
        WHERE d_lingua.lingua = Anomalia.lingua;
) AS id_lingua, (
        SELECT temAnomaliaRedacao
```

```
        FROM Anomalia
) AS tipo_anomalia, (
        SELECT CASE (SELECT idAnomalia AS ids FROM Correcao;)
                WHEN Anomalia.idAnomalia IN ids THEN B'1'
                ELSE B'0'
        END;
) AS com_proposta
FROM Anomalia NATURAL JOIN Incidencia NATURAL JOIN Item NATURAL JOIN Correcao;
```

Data Analytics

```
SELECT tipo, língua, dia_da_semana, count()
FROM f_anomalia
        NATURAL JOIN d_utilizador
        NATURAL JOIN d_lingua
        NATURAL JOIN d_tempo
GROUP BY tipo, lingua, dia_da_semana

UNION

SELECT tipo, língua, NULL, count()
FROM f_anomalia
        NATURAL JOIN d_utilizador
        NATURAL JOIN d_lingua
        NATURAL JOIN d_tempo
GROUP BY tipo, lingua, dia_da_semana

UNION

SELECT tipo, NULL, NULL, count()
FROM f_anomalia
        NATURAL JOIN d_utilizador
        NATURAL JOIN d_lingua
        NATURAL JOIN d_tempo
GROUP BY tipo, lingua, dia_da_semana;
```