Projeto de Bases de Dados - Parte 3

Construção de uma Base de Dados

Grupo 11: Esforço/Percentagem
78072 – António Torgal : 18 Horas/45%
84584 – Bruno Neves : 15 Horas/35%
84615 – Nuno Martins : 10 Horas/20%

Turno de 4ªFeira
16:30 – 18:00
Docente:
Paulo Carreira

1. Criação e Preenchimento da Base de Dados

1.1. Esquema de Tabelas

```
CREATE TABLE local_publico(
                latitude NUMERIC(9,6) NOT NULL,
                longitude NUMERIC(9,6) NOT NULL,
                nome varchar(100),
                PRIMARY KEY(latitude, longitude)
);

CREATE TABLE item(
   id INTEGER NOT NULL UNIQUE,
   descricao varchar(100),
   localizacao varchar(100),
   latitude NUMERIC(9,6),
   longitude NUMERIC(9,6),
   PRIMARY KEY(id),
   FOREIGN KEY(latitude, longitude) REFERENCES local_publico(latitude, longitude) ON DELETE CASCADE
);

CREATE TABLE anomalia(
   id INTEGER NOT NULL UNIQUE,
   zona varchar(100),
   imagem varchar(100),
   lingua varchar(100),
   ts timestamp,
   descricao varchar(100),
   tem_anomalia_redacao boolean,
   PRIMARY KEY(id)
);

CREATE TABLE anomalia_traducao(
   id INTEGER NOT NULL UNIQUE,
   zona2 varchar(100),
   lingua2 varchar(100),
   PRIMARY KEY (id),
   FOREIGN KEY(id) REFERENCES anomalia(id)
);


CREATE TABLE duplicado(
   item1 INTEGER NOT NULL,
   item2 INTEGER NOT NULL,
   PRIMARY KEY(item1, item2),
   FOREIGN KEY(item1) REFERENCES item(id),
   FOREIGN KEY(item2) REFERENCES item(id),
   CONSTRAINT check_duplicado CHECK (item2>item1)
);

CREATE TABLE utilizador(
   email varchar(100) NOT NULL UNIQUE,
   password varchar(100),
   PRIMARY KEY(email)
);

CREATE TABLE utilizador_qualificado(
   email varchar(100) NOT NULL UNIQUE,
   PRIMARY KEY(email),
   FOREIGN KEY(email) REFERENCES utilizador(email) ON DELETE CASCADE
);

CREATE TABLE utilizador_regular(
   email varchar(100) NOT NULL UNIQUE,
   PRIMARY KEY(email),
   FOREIGN KEY(email) REFERENCES utilizador(email) ON DELETE CASCADE
);

CREATE TABLE incidencia(
   anomalia_id INTEGER NOT NULL UNIQUE,
   item_id INTEGER NOT NULL,
   email varchar(100) NOT NULL,
   PRIMARY KEY(anomalia_id),
   FOREIGN KEY(anomalia_id) REFERENCES anomalia(id) ON DELETE CASCADE,
   FOREIGN KEY(email) REFERENCES utilizador(email) ON DELETE CASCADE,
   FOREIGN KEY(item_id) REFERENCES item(id) ON DELETE CASCADE
);

CREATE TABLE proposta_de_correcao(
   email varchar(100) NOT NULL,
   nro INTEGER NOT NULL,
   data_hora timestamp,
   texto varchar(1000),
   FOREIGN KEY(email) REFERENCES utilizador_qualificado(email) ON DELETE CASCADE,
   PRIMARY KEY(email, nro)
);

CREATE TABLE correcao(
   email varchar(100) NOT NULL,
   nro INTEGER NOT NULL,
   anomalia_id INTEGER NOT NULL ,
   PRIMARY KEY(email, nro, anomalia_id),
   FOREIGN KEY(email, nro) REFERENCES proposta_de_correcao(email, nro) ON DELETE CASCADE,
   FOREIGN KEY(anomalia_id) REFERENCES incidencia(anomalia_id) ON DELETE CASCADE);
```

1.2. Restrições de Integridade

```
-- RI-1

CREATE FUNCTION check_zona_anomalia ()
RETURNS TRIGGER AS
$$
  BEGIN
    IF NEW.zona2 LIKE (SELECT zona FROM anomalia WHERE id = NEW.id)
      THEN RAISE EXCEPTION 'As zonas sobrepoem se para a anomalia';
    END IF;
    RETURN NEW;
  END
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_zona_anomalia
BEFORE INSERT ON anomalia_traducao
FOR EACH ROW EXECUTE PROCEDURE          check_zona_anomalia();


-- RI-4

CREATE FUNCTION check_utilizador ()
RETURNS TRIGGER AS
$$
  BEGIN
    IF NEW.email LIKE (SELECT email FROM utilizador_qualificado WHERE email = NEW.email)
         OR (SELECT email FROM utilizador_regular WHERE email = NEW.email)
       THEN RETURN NEW;
    ELSE
      RAISE EXCEPTION 'O email não existe num utilizador qualificado ou regular';
    END IF;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_utilizador
AFTER INSERT ON utilizador
FOR EACH ROW EXECUTE PROCEDURE check_utilizador();

-- RI-5

CREATE FUNCTION check_utilizador_regular ()
RETURNS TRIGGER AS
$$
  BEGIN
    IF NEW.email LIKE (SELECT email FROM utilizador_regular WHERE email = NEW.email)
      THEN RAISE EXCEPTION 'O email existe num utilizador regular';
    END IF;
    RETURN NEW;
  END
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_utilizador_qualificado
BEFORE INSERT ON utilizador_qualificado
FOR EACH ROW EXECUTE PROCEDURE          check_utilizador_regular();


-- RI-6

CREATE FUNCTION check_utilizador_qualificado ()
RETURNS TRIGGER AS
$$
  BEGIN
    IF NEW.email LIKE (SELECT email FROM utilizador_regular WHERE email = NEW.email)
      THEN RAISE EXCEPTION 'O email existe num utilizador qualificado';
    END IF;
    RETURN NEW;
  END
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_utilizador_regular
BEFORE INSERT ON utilizador_regular
FOR EACH ROW EXECUTE PROCEDURE          check_utilizador_qualificado();
```

2. SQL

```
SELECT l.nome, l.latitude, l.longitude
FROM local_publico l
   INNER JOIN item i
   ON l.latitude = i.latitude AND l.longitude = i.longitude
   INNER JOIN incidencia ic
   ON i.id = ic.item_id
   INNER JOIN anomalia a
   ON ic.anomalia_id = a.id
GROUP BY l.nome, l.latitude, l.longitude
HAVING COUNT(*) <= ALL(
   SELECT COUNT(*)
   FROM local_publico
   GROUP BY nome
   );

SELECT l.nome, l.latitude, l.longitude
FROM local_publico l
   INNER JOIN item i
   ON l.latitude = i.latitude AND l.longitude = i.longitude
   INNER JOIN incidencia ic
   ON i.id = ic.item_id
   INNER JOIN anomalia a
   ON ic.anomalia_id = a.id
   INNER JOIN anomalia_traducao at
   ON a.id = at.id
WHERE DATE_PART('quarter', a.ts) = 1
GROUP BY l.nome, l.latitude, l.longitude
HAVING COUNT(*) > ALL(
   SELECT COUNT(*)
   FROM local_publico
   GROUP BY nome
   );


SELECT DISTINCT u.email, u.password
FROM utilizador u
   INNER JOIN proposta_de_correcao p
   ON u.email = p.email
   INNER JOIN correcao c
   ON p.email = c.email AND p.nro = c.nro
   INNER JOIN incidencia i
   ON c.anomalia_id = i.anomalia_id
   INNER JOIN anomalia a
   ON i.anomalia_id = a.id
   INNER JOIN item it
   ON i.item_id = it.id
WHERE DATE_PART('year', ts) = 2020 AND it.latitude < 39.336775;


SELECT DISTINCT u.email
FROM utilizador u
   INNER JOIN correcao1 c
   ON u.email = c.email
   INNER JOIN proposta_de_correcao p
   ON c.email = p.email AND c.nro = p.nro
WHERE NOT EXISTS (
   SELECT *
   FROM incidencia i
   WHERE c.anomalia_id = i.anomalia_id
   )
AND DATE_PART('year', data_hora) = 2020;
```

3. Aplicação PHP e HTML

3.1 Arquitetura da aplicação

A página inicial da aplicação é BD.html, nesta página é possível serem selecionadas 6 opções distintas:

1. Inserir e remover locais, itens e anomalias.
2. Inserir, editar e remover correções e propostas de correção.
3. Listar uma tabela com todos os utilizadores.
4. Registar incidências e duplicados.
5. Listar uma tabela com todas as anomalias entre dois locais públicos.
6. Listar uma tabela com todas as anomalias registadas nos últimos três meses.

Para as opções 1, 2, 4, 5 e 6 são necessárias a introdução de atributos para executar essas operações. É necessário submeter:

1. Os atributos de um local, item ou anomalia.
2. Os atributos de uma correção ou proposta de correção.
4. Os atributos de uma incidência e duplicado.
5. O nome de cada local público.
6. Uma latitude e longitude.

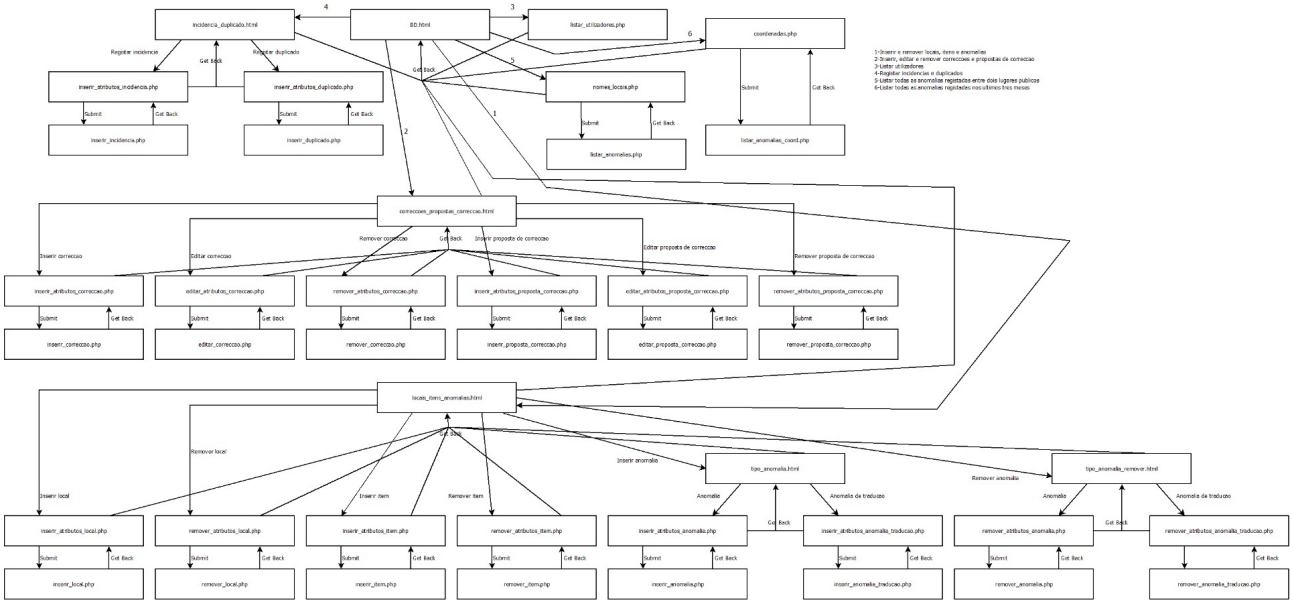

Imagem 1: Relação dos Ficheiros da Base de Dados HTML e PHP

4. Índices

4.1. Devolver o número de anomalias de um determinado item

Para acelerar esta pesquisa vão ser criados dois índices.
O indíce primário para a chave primária anomalia_id provém da tabela incidência para tornar a operação count(*) mais eficiente.
O PSQL já cria o índice primário automaticamente para cada tabela, logo não é necessário ser feito manualmente.
O segundo índice é um índice hash para a coluna item_id, porque estamos à procura de um valor específico e não de um intervalo.

```
CREATE INDEX item_idx
ON incidencia
USING HASH(item_id);
```

4.2 Listar todas as línguas e descrições entre dois momentos no tempo

Existem várias soluções para aumentar a velocidade de procura desta pesquisa.
Podemos criar só um índice individual para cada coluna, ts, tem_anomalia_redacao e língua da tabela anomalia, ou um complexo para todas as colunas.
Decidimos que o mais eficiente é criar um índice b-tree complexo para todas as colunas, porque hash não suporta índices com mais que uma coluna. Outra razão é por estarmos a pesquisar entre um intervalo de valores, por exemplo, no caso de ts.
A ordem de criação tem de ser ser: (tem_anomalia_redacao, lingua, ts), isto deve-se ao facto da tem_anomalia_redacao ser menos seletiva, apenas assumir dois valores que são *true or false.*
Também existem menos línguas que segundos num dia e as línguas tendem a repetir-se mais, logo o índice terá menos páginas do que no caso em que for feito por outra ordem.

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

5. Modelo Multidimensional

5.1. Tabelas de Dimensões

```
CREATE TABLE d_utilizador(
   id_utilizador SMALLSERIAL,
   email varchar(100) NOT NULL,
   tipo varchar(20) NOT NULL,
   PRIMARY KEY(id_utilizador)

);


CREATE TABLE d_tempo(
   id_tempo SMALLSERIAL,
   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 SMALLSERIAL,
   latitude NUMERIC(9,6) NOT NULL,
        longitude NUMERIC(9,6) NOT NULL,
        nome varchar(100) NOT NULL,
   PRIMARY KEY(id_local)
);


CREATE TABLE d_lingua(
   id_lingua SMALLSERIAL,
   lingua varchar(100) NOT NULL,
   PRIMARY KEY(id_lingua)
);
```

5.2. Tabelas de Factos

```
CREATE TABLE f_anomalia(
   id_utilizador INTEGER NOT NULL,
   id_tempo INTEGER NOT NULL,
   id_local INTEGER NOT NULL,
   id_lingua INTEGER NOT NULL,
   tipo_anomalia varchar(20) NOT NULL,
   com_proposta boolean NOT NULL,
   FOREIGN KEY(id_utilizador) REFERENCES d_utilizador,
   FOREIGN KEY(id_tempo) REFERENCES d_tempo,
   FOREIGN KEY(id_local) REFERENCES d_local,
   FOREIGN KEY(id_lingua) REFERENCES d_lingua,
   PRIMARY KEY(id_utilizador, id_tempo, id_local, id_lingua)
);
```

6. ETL e Data Analytics

```
INSERT INTO d_utilizador (email, tipo)
SELECT email, 'qualificado'
FROM utilizador_qualificado
UNION
SELECT email, 'regular'
FROM utilizador_regular;

INSERT INTO d_tempo(dia, dia_da_semana, semana, mes, trimestre, ano)
SELECT DISTINCT
DATE_PART('day', ts),
DATE_PART('dow', ts),
DATE_PART('week', ts),
DATE_PART('month', ts),
DATE_PART('quarter', ts),
DATE_PART('year', ts)
FROM anomalia;

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

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


INSERT INTO f_anomalia (id_utilizador, id_tempo, id_local, id_lingua, tipo_anomalia, com_proposta)
SELECT DISTINCT
  id_utilizador,
  id_tempo,
  id_local,
  id_lingua,
  CASE WHEN tem_anomalia_redacao = true THEN 'redacao' ELSE 'traducao' END,
  CASE WHEN c.anomalia_id= i.anomalia_id THEN true ELSE false END
FROM anomalia a
LEFT OUTER JOIN incidencia i
ON  a.id = i.anomalia_id
LEFT OUTER JOIN d_utilizador u
ON u.email = i.email
LEFT OUTER JOIN d_lingua dl
ON a.lingua = dl.lingua
LEFT OUTER JOIN item it
ON i.item_id = it.id
LEFT OUTER JOIN d_local dlo
ON it.latitude = dlo.latitude AND it.longitude = dlo.longitude
LEFT OUTER JOIN correcao c
ON i.anomalia_id = c.anomalia_id
LEFT OUTER JOIN d_tempo dt
ON a.ts::date = make_date(dt.ano, dt.mes, dt.dia);
```