Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Base de Dados I

Relatório Relativo ao Trabalho Prático

Realizado por: Guilherme Bento - 25193

Tiago Portugal -30816

Lucas Santos – 27440

César Cabral -6802

Gabriel Maciel- 24819

Nome e Apelido – Número Mecanográfico

Etapa #2

Nesta etapa vamos demostrar algumas consultas e objétos lógicos usados no projeto segundo requesitos e programa definido pela cadeira de BD1.

Consultas

Representamos aqui consultas que implementam várias operações de algebra relacional como projeções, junções (internas e externas), agrupamentos, condições, etc . . . categorizando-os em objétivos definidos para esta entrega.

Junção interna + subconsuta

| Selecionar Utilizadores inscritos no curso 'Desenvolvimento Web' :

```
SELECT idformando, formandos.nome, email FROM inscricao JOIN

( SELECT idformando,nome,email FROM formando
JOIN utilizadores ON utilizador=idutilizador )
AS formandos ON formando=idformando

JOIN curso ON curso=idcurso

WHERE curso.nome='Desenvolvimento Web';
```

junção externa

| Selecionar Utilizadores que não são administradores

```
SELECT idutilizador,nome,email
FROM utilizadores
LEFT OUTER JOIN admin ON utilizadores.idutilizador = admin.utilizador
WHERE (admin.idadmin IS NULL OR admin.ativo = false)
  AND utilizadores.ativo = true;
```

função de agregação + subquery + junção interna

| Obter horas totais lecionadas em cada curso sincrono

```
SELECT idcurso,nome,horaslecionadas  FROM cursosincrono JOIN

(
    SELECT cursosincrono, SUM(duracaohoras) AS horaslecionadas
    FROM sessao
    WHERE CURRENT_TIMESTAMP > datahora + (duracaohoras * INTERVAL '1 hour')
    GROUP BY cursosincrono
) AS t
ON idcursosincrono = cursosincrono
JOIN curso ON cursosincrono.curso = idcurso;
```

Objétos Lógicos

Dentro dos objétos lógicos como indicado representamos aqui, triggers, udfs e procedimentos.

Além disso achamos relevante inserir aqui outra estrutura que criamos `indexes` .

Triggers

Em postgresql para criar um trigger regularmente cria-se uma udf com retorno `trigger` para depois ser utilizada na sua defenição.

As udfs criadas para os triggers que vamos demostrar são as seguintes :

soft-delete de utilizador :

```
CREATE OR REPLACE FUNCTION desativar_utilizador()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.ativo THEN

    UPDATE utilizadores
        SET ativo = false
        WHERE idutilizador = OLD.idutilizador;

    END IF;

    UPDATE admin
        SET ativo = false
        WHERE utilizador = OLD.idutilizador;

    UPDATE formador
        SET ativo = false
        WHERE utilizador = OLD.idutilizador;

    UPDATE formando
        SET ativo = false
        WHERE utilizador = OLD.idutilizador;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Adicionar role (insert numa tabela dos papeis admin, formando ou formador) **:**

```
CREATE OR REPLACE FUNCTION adicionar_role()
RETURNS TRIGGER AS $$
DECLARE
    userActive BOOLEAN;
    inRole BOOLEAN;
BEGIN
    EXECUTE 'SELECT ativo FROM utilizadores WHERE idutilizador = $1'
    INTO userActive
    USING NEW.utilizador;

    IF NOT userActive THEN
        RAISE EXCEPTION 'Utilizador não está ativo';
        RETURN NULL;
    END IF;

    EXECUTE format(
        'SELECT EXISTS (SELECT 1 FROM %I WHERE utilizador = $1)',
        TG_TABLE_NAME
    )
    INTO inRole
    USING NEW.utilizador;

    IF inRole THEN
        EXECUTE format(
            'UPDATE %I SET ativo = true WHERE utilizador = $1',
            TG_TABLE_NAME
        )
        USING NEW.utilizador;

        RETURN NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Criação dos triggers :

Todos estes triggers são do tipo `Before Insert`, sendo o primeiro na tabela utilizador com a primeira udf acima e os restantes nas tabelas de papéis com a segunda udf.

```
CREATE OR REPLACE TRIGGER inserir_utilizador_trigger
BEFORE INSERT ON utilizadores
FOR EACH ROW
EXECUTE FUNCTION inserir_utilizador();

CREATE TRIGGER adicionar_admin
BEFORE INSERT ON admin
FOR EACH ROW
EXECUTE FUNCTION adicionar_role();


CREATE TRIGGER adicionar_formador
BEFORE INSERT ON formador
FOR EACH ROW
EXECUTE FUNCTION adicionar_role();


CREATE TRIGGER adicionar_formador
BEFORE INSERT ON formando
FOR EACH ROW
EXECUTE FUNCTION adicionar_role();
```

UDFs

Ignorando as udfs acima, criadas com o unico processo de ser utilizadas no trigger apresentamos aqui uma função que utilizamos no projeto para verificar se um utilizador (dado o seu id) é admin ou não :

```
CREATE OR REPLACE FUNCTION isadmin(p_id BIGINT)
RETURNS BOOLEAN AS $$
DECLARE
    v_ativo BOOLEAN;
BEGIN

    SELECT ativo INTO v_ativo
    FROM Admin
    WHERE utilizador = p_id
    LIMIT 1;

    IF FOUND THEN
        RETURN v_ativo;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Procedimentos

Quanto aos procedimentos mostramos aqui dois para os objetivos abaixo :

1. Atribuir papeis ao um utilizador dado uma flag por cada papel
2. Utilização de um cusor para listar a tabela dos utilizadores (não encontramos nada relvante para cumprir o requesito da utilização de um cursor)

1. `setStatus`

```
CREATE OR REPLACE PROCEDURE setStatus(
    p_admin BOOLEAN,
    p_formando BOOLEAN,
    p_formador BOOLEAN,
    p_id BIGINT
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF p_admin THEN
        INSERT INTO admin(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM admin WHERE utilizador = p_id;
    END IF;

    IF p_formando THEN
        INSERT INTO formando(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM formando WHERE utilizador = p_id;
    END IF;

    IF p_formador THEN
        INSERT INTO formador(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM formador WHERE utilizador = p_id;
    END IF;
END;
$$;
```

2. `list_utilizadores`

```sql
CREATE OR REPLACE PROCEDURE list_utilizadores()
LANGUAGE plpgsql
AS $$
DECLARE
    cur CURSOR FOR
        SELECT idUtilizador, nome, email, telefone, ativo
        FROM Utilizadores;

    v_id BIGINT;
    v_nome VARCHAR(60);
    v_email VARCHAR(60);
    v_telefone CHAR(9);
    v_ativo BOOLEAN;
BEGIN
    OPEN cur;

    LOOP
        FETCH cur INTO v_id, v_nome, v_email, v_telefone, v_ativo;

        EXIT WHEN NOT FOUND;

        RAISE NOTICE 'ID: %, Nome: %, Email: %, Telefone: %, Ativo: %',
            v_id, v_nome, v_email, v_telefone, v_ativo;
    END LOOP;

    CLOSE cur;
END;
$$;
```