Universidade Federal Fluminense
TCC00288 – Banco de Dados II, Turma A1/2021.2
P2 – 26/01/2022

Aluno: Pedro Alves Valentim
Matrícula: 217031096

1) [1,0 ponto] Defina o que é uma transação.

Uma transação é uma unidade lógica de processamento no banco de dados. Ou seja, um conjunto de uma ou mais operações que compõem uma única tarefa ou unidade lógica de trabalho a ser executada. As transações podem ser de qualquer uma das quatro operações básicas de armazenamento persistente (CRUD): criação (create), leitura (read), atualização (update) e exclusão (delete).

2) [1,0 ponto] Cite e descreva todas as propriedades que um SGBD deve garantir para suas transações.

São 4 as propriedades que um SGBD deve garantir:

Preservação de consistência: Uma transação deve deixar o banco de dados em um estado consistente após sua realização.

Atomicidade: Uma transação não pode ser realizada parcialmente. Ou é realizada por completo, ou não é realizada.

Permanência: As mudanças feitas por uma transação não podem ser perdidas por alguma falha.

Isolamento: Uma transação não deve interferir ou ser interferida pela execução de quaisquer outras transações que estejam acontecendo simultaneamente.

3) [1,0 ponto] Defina “plano de execução concorrente (schedule)” de um conjunto de transações.

Quando mais de uma transação deseja acessar a mesma área de dados e existe concorrência, o que poderia burlar a consistência do SGBD, é necessário criar um plano de execução. O plano de execução é a definição de uma ordem sobre quais instruções executar em qual momento, a fim de evitar a alteração indevida dos dados e gerar inconsistência. Apesar de alterar a execução de diferentes transações, mantém a ordem de execução das operações de

cada transação individual.

4) [1,0 ponto] Considerando as transações a seguir escreva um plano de execução serializável utilizando a técnica de bloqueio em duas fases básico (não é conservador, estrito nem rigoroso).

T_1: {R(Y), R(X), W(X)}
T_2: {R(X), R(Y), W(Y)}

T1	T2
read_lock(Y)	
read_item(Y)	
write_lock(X)	
	read_lock(X)
unlock(Y)	
read_item(X)	
write_item(X)	
unlock(X)	
	read_item(X)
	write_lock(Y)
	unlock(X)
	read_item(Y)
	write_item(Y)
	unlock(Y)

5) [1,0 ponto] Descreva uma técnica alternativa para a geração planos de execução serializáveis que não seja a técnica de bloqueios em duas fases.

Uma técnica alternativa para criação dos planos de execução é a técnica de ordenação por rótulo de tempo. Esta técnica utiliza a ordem de chegada das transações para gerenciar a organização das operações no schedule. Para cada transação iniciada, é associado um timestamp fixo exclusivo. Ou seja, antes que uma transação tenha início, o sistema de banco de dados fornecerá um rótulo de tempo (um identificador exclusivo criado pelo SGBD para identificar uma transação). Desse modo, transações que acessam o item de dados depois têm seu rótulo de tempo comparado ao rótulo marcado no item do banco. A

depender dos valores, a operação pode acessar o item ou ser abortada e reenviada ao sistema (gerando um novo rótulo).

6) [5,0 pontos] Considere o esquema de banco de dados sobre apresentações artísticas como definido a seguir. Implemente trigger(s) em PL/pgSQL para garantir que tanto as arenas quanto os artistas não estejam ocupados simultaneamente com mais de um concerto. Crie também outro(s) trigger(s) em PL/pgSQL para não se permitir excluir todos os artistas associados a uma atividade, isto é, após a associação de um artista a uma atividade essa atividade não poderá deixar de estar associada a pelo menos um artista.

Atividade (ID, nome)
Artista (ID, nome, rua, cidade, estado, cep, atividade)
Arena (ID, nome, cidade, capacidade)
Concerto (ID, artista, arena, inicio, fim, preco)

```
DO $$ BEGIN
    PERFORM drop_functions();
    PERFORM drop_tables();
END $$;

CREATE TABLE Atividade(
    id INT,
    nome VARCHAR);

CREATE TABLE Artista(
    id INT,
    nome VARCHAR,
    atividade INT);

CREATE TABLE Arena(
    id INT,
    nome VARCHAR);

CREATE TABLE Concerto(
```

```
    id INT,
    artista INT,
    arena INT,
    inicio TIMESTAMP,
    fim TIMESTAMP);

INSERT INTO Atividade values (1, 'Show de Rock');
INSERT INTO Atividade values (2, 'Show de Bossa Nova');
INSERT INTO Atividade values (3, 'Apresentação circense');
INSERT INTO Artista values (1, 'Led Zeppelin', 1);
INSERT INTO Artista values (2, 'João Gilberto', 2);
INSERT INTO Artista values (3, 'Cirque du Soleil', 2);
INSERT INTO Arena values (1, 'Maracanã');
INSERT INTO Arena values (2, 'Canecão');
INSERT INTO Arena values (3, 'Vivo Rio');

CREATE OR REPLACE FUNCTION checkUnique() RETURNS TRIGGER AS $$
declare
begin

    IF EXISTS (SELECT * FROM Concerto
                WHERE concerto.id != new.id AND
                (concerto.arena = new.arena OR concerto.artista =
new.artista) AND
                (concerto.inicio BETWEEN new.inicio AND new.fim or
concerto.fim BETWEEN new.inicio and new.fim)) THEN
        raise exception 'Horários informados incompatíveis';
    END IF;

    return NEW;
end;
$$ language plpgsql;

CREATE TRIGGER checkUnique
```

```
AFTER INSERT OR UPDATE ON Concerto FOR EACH ROW
EXECUTE PROCEDURE checkUnique();

CREATE OR REPLACE FUNCTION auxFunction() RETURNS TRIGGER AS $$
declare
begin
    create temp table activityAux(id int) on commit drop;
    return null;
end;
$$ language plpgsql;

CREATE TRIGGER auxTrigger
BEFORE UPDATE OR DELETE ON Artista FOR EACH STATEMENT
EXECUTE PROCEDURE auxFunction();

CREATE OR REPLACE FUNCTION registerArtist() RETURNS TRIGGER AS $$
declare
begin
    INSERT INTO activityAux values(old.atividade);
    return null;
end;
$$ language plpgsql;

CREATE TRIGGER registerArtist
AFTER UPDATE OR DELETE ON Artista FOR EACH ROW
EXECUTE PROCEDURE registerArtist();

CREATE OR REPLACE FUNCTION checkActivity() RETURNS TRIGGER AS $$
declare
    counter int;
    atvd record;
begin

    FOR atvd in SELECT DISTINCT * FROM activityAux LOOP
```

```
        Select count(*) from Artista WHERE atividade = atvd.id INTO
counter;
        IF counter = 0 THEN
            raise exception 'Atividade informada sem artista';
        END IF;

    END LOOP;

    return NULL;
end;
$$ language plpgsql;

CREATE TRIGGER checkActivity
AFTER UPDATE OR DELETE ON Artista FOR EACH STATEMENT
EXECUTE PROCEDURE checkActivity();

-- Problema da atividade vazia:
DELETE FROM Artista WHERE id = 1;

-- Problema da atividade vazia:
UPDATE Artista set atividade = 2 WHERE id = 1;

-- Arenas
INSERT INTO Concerto values (1, 1, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:10');
INSERT INTO Concerto values (2, 1, 2, '2022-11-10 00:00:00',
'2022-11-10 00:00:05');
INSERT INTO Concerto values (3, 1, 3, '2022-11-10 00:00:00',
'2022-11-10 00:00:30');

-- Artistas
INSERT INTO Concerto values (1, 1, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:10');
```

```
INSERT INTO Concerto values (2, 2, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:05');
INSERT INTO Concerto values (3, 3, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:30');

-- Horários
INSERT INTO Concerto values (1, 1, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:10');
INSERT INTO Concerto values (2, 2, 1, '2022-10-10 00:00:00',
'2022-10-10 00:00:05');
INSERT INTO Concerto values (3, 3, 1, '2022-11-10 00:00:00',
'2022-11-10 00:00:30');
```