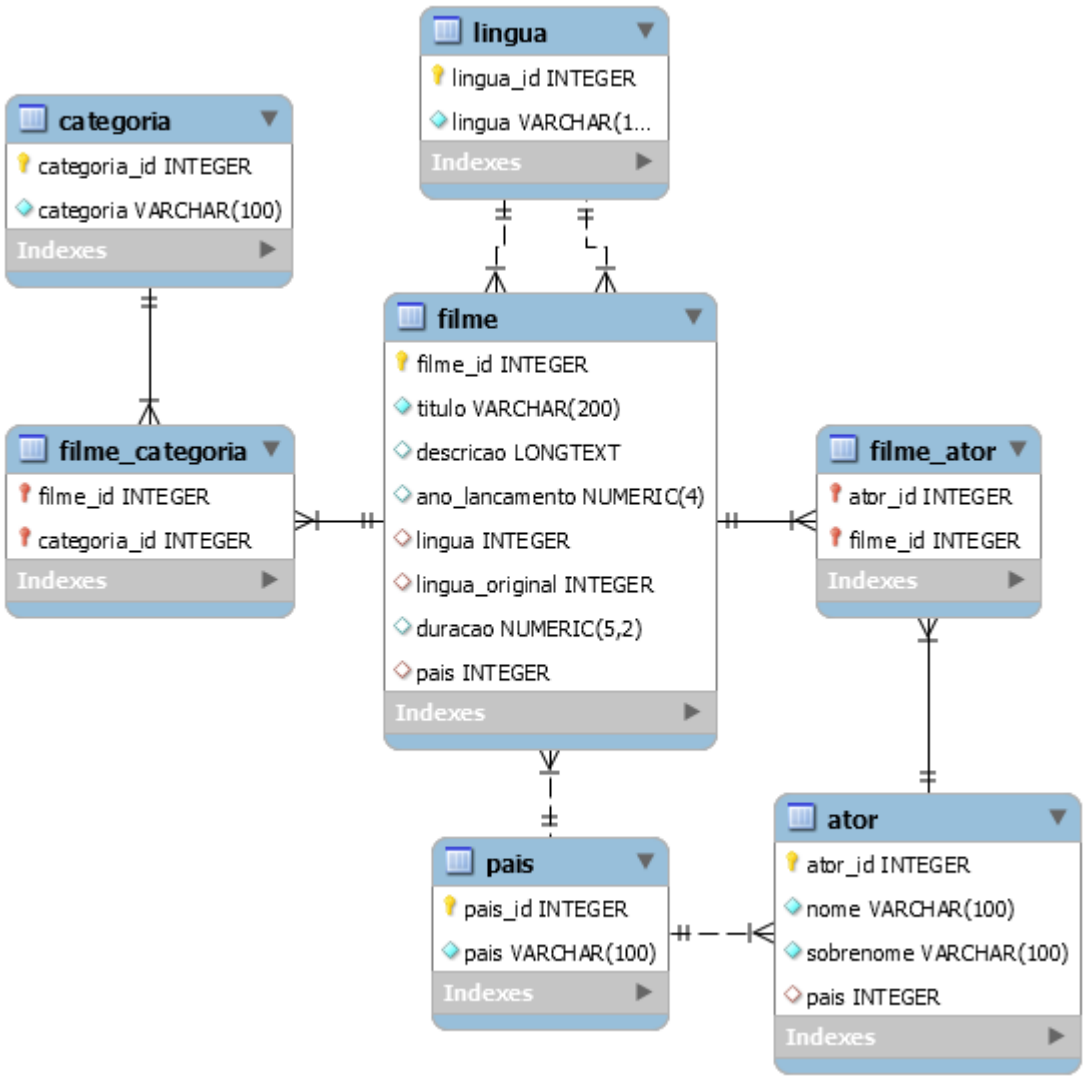
UNIVATES
BANCO DE DADOS 2015B - RESPOSTAS DA PROVA 2-1
PROF. JULIANO DERTZBACHER

ALUNO: _____

Identifique-se na folha da prova. Esta prova é individual e não permite a consulta a qualquer material que seja de domínio público, somente os arquivos pessoais autorizados. A prova consiste em 10 pontos distribuídos nas diversas questões, o peso de cada questão está descrito ao lado do respectivo enunciado. Somente serão aceitos comentários sobre o conteúdo da prova nos primeiros 30 minutos, as questões devem ser expostas em voz alta para que todos possam ouvir. A interpretação das questões faz parte da avaliação.

Para resolver as questões 1, 2, 3 e 4, utilize como base o modelo relacional abaixo. A implementação deste modelo no PostgreSQL pode ser recuperada a partir dos arquivos "prova2.backup" ou "prova2.sql", disponíveis no Ambiente Virtual (Aula16). As respostas para estas 3 questões devem ser submetidas na tarefa "Prova2", com o nome de arquivo "P2_NomeSobrenome.sql".



Questão 1 (1 ponto) – Utilize os recursos das subconsultas para listar os nomes dos filmes americanos que possuam uma duração superior à duração de todos os filmes italianos.

```
SELECT a.titulo
FROM filme a
WHERE a.pais = 2
AND a.duracao > ALL (
        SELECT i.duracao
        FROM filme i
        WHERE i.pais = 8);
```

Questão 2 (1 ponto) – Desenvolva uma visão (view) para retornar o nome completo dos atores (nome e sobrenome), suas respectivas nacionalidades (nome do país) e o código dos filmes nos quais atuaram.

```
CREATE OR REPLACE VIEW vw AS (
        SELECT a.nome || ' ' || a.sobrenome AS "Nome", p.pais AS "Nacionalidade", fa.filme_id
        FROM filme_ator fa, ator a, pais p
        WHERE fa.ator_id = a.ator_id
        AND a.pais = p.pais_id);

SELECT * FROM vw;
```

Questão 3 (2,5 pontos) – Desenvolva uma função (stored procedure) que recebe como parâmetro o código de um filme e retorna o conteúdo filtrado da visão (view) criada na questão anterior, utilizando um registro (record). Elabore também a chamada para esta stored procedure.

```
REATE OR REPLACE FUNCTION sp (f INTEGER)
        RETURNS SETOF vw
        AS $$
        DECLARE
                sel_vw RECORD;
        BEGIN
                FOR sel_vw IN (
                        SELECT *
                        FROM vw
                        WHERE filme_id = f) LOOP
                        RETURN NEXT sel_vw;
                END LOOP;
        END;
        $$ LANGUAGE plpgsql;

SELECT * from sp (4)
```

Questão 4 (2,5 pontos) – Desenvolva um gatilho (trigger) que realiza uma inserção automática na tabela "filme_categoria" quando um novo filme é inserido na tabela "filme", atribuindo a este registro automático o código do filme inserido e a categoria de código 8 (Outra). Teste a funcionalidade desta trigger efetuando uma inserção na tabela "filme" a fim de verificar o impacto na tabela "filme_categoria".

```
CREATE OR REPLACE FUNCTION ft ()
        RETURNS TRIGGER
        AS $$
        BEGIN
                INSERT INTO filme_categoria VALUES
                (NEW.filme_id, 8);
                RETURN NEW;
        END;
        $$ LANGUAGE plpgsql;

CREATE TRIGGER t
        AFTER INSERT ON filme
        FOR EACH ROW
        EXECUTE PROCEDURE ft ();

INSERT INTO filme (filme_id, titulo, descricao, ano_lancamento, lingua, lingua_original, duracao, pais)
VALUES
        (16, 'O Exterminador do Futuro: Gênesis', NULL, 2015, 2, 2, 126, 2);
```

Questão 5 (1 ponto) – Considere duas tabelas, criadas pelos comandos SQL a seguir, contendo os dados dos empregados de uma empresa e seus respectivos dependentes.

```
CREATE TABLE empregado (
        cpf CHAR(11) NOT NULL,
        nome VARCHAR(100) NOT NULL,
        salario DECIMAL(10,2) NOT NULL,
        sexo CHAR NOT NULL,
        CONSTRAINT pk_empregado PRIMARY KEY (cpf)
);
CREATE TABLE dependente (
        emp_cpf CHAR(11) NOT NULL,
        num_seq INTEGER NOT NULL,
        nome VARCHAR(100) NOT NULL,
        sexo CHAR NOT NULL,
        CONSTRAINT pk_dependente PRIMARY KEY (emp_cpf, num_seq),
        CONSTRAINT fk_dependente_empregado FOREIGN KEY (emp_cpf) REFERENCES empregado (cpf)
);
```

Com referência à seguinte consulta SQL, marque a alternativa correta.

```
SELECT *
FROM empregado e
WHERE e.cpf IN (
        SELECT d.emp_cpf
        FROM dependente d
        WHERE e.nome = d.nome
        AND e.sexo = d.sexo);
```

a) Recupera todos os dados de cada empregado que possui pelo menos um dependente com o mesmo nome e mesmo sexo do empregado.
b) Recupera todos os dados de cada empregado que possui algum dependente com o mesmo nome ou mesmo sexo do empregado.
c) Recupera todos os dados de cada empregado e seus respectivos dependentes, desde que o sexo e o nome dos empregados e seus dependentes sejam iguais.
d) A consulta não funcionará, pois existe erro de sintaxe.

Questão 6 (1 ponto) – Sobre o gerenciamento de funções (stored procedures) no SGBD PostgreSQL, assinale a sentença cujo comando remova a função chamada "minha_funcao", sem que o SGBD apresente uma mensagem de erro caso ela não exista:

a) DELETE FUNCTION minha_funcao CASCADE;
b) DELETE FUNCTION IF EXISTS minha_funcao;
c) DROP FUNCTION minha_funcao IF NOT EXISTS;
d) DROP FUNCTION IF EXISTS minha_funcao;

Questão 7 (1 ponto) – Através da instrução CREATE TRIGGER na linguagem SQL, podemos criar gatilhos (triggers), os quais correspondem a ações a serem tomadas em um banco de dados quando certos eventos ocorrem e quando certas condições são satisfeitas. A respeito de triggers, assinale a afirmação correta:

a) A cláusula BEFORE, quando corretamente empregada na especificação de um gatilho, determina que o evento que disparou o gatilho deve ser concluído antes que as ações do gatilho sejam executadas.
b) A palavra-chave NEW é usada para se referir a uma tupla recém-inserida ou recém-atualizada, enquanto a palavra-chave OLD é usada para se referir a uma tupla recém-excluída ou a uma tupla antes que ela seja atualizada.
c) É possível executar um gatilho com a instrução CALL.
d) Dentre os eventos que podem disparar a execução de triggers, podemos citar o INSERT, DELETE, UPDATE e SELECT.

BOA PROVA!