**Banco de Dados 1 - Atividade Individual 2**
**Mirela Mei - nº USP: 11208392**

1 - a) Relação G(C, S, L, P)
Sendo PC → S, CL → P, e P → C, então P→ S.
PC = S, P, C
CL = P, C, L, S
P = S,C,P
Sendo chaves candidatas aquelas capazes de identificar uma tupla de forma única, as candidatas para essa relação são (C, L), uma vez que todos os atributos são dependentes dessa combinação.

b) G está na primeira forma normal pois todos os atributos são atômicos e não há tabelas aninhadas.
1FN = G(C, S, L, P)
2FN = G1(C,L,P) e G2(P,S)
3FN = G1(C,L,P) e G2(P,S)
Em G1 CL é a chave primária e P depende apenas dela, e em G2 C é a chave primária e S depende apenas dela.

2 - CONTRATOS (IdCont, IdForn, IdProj, IdDepto, IdMat, Quant, Valor)
{IdProj, IdMat} -> IdCont
{IdForn, IdDepto} -> IdMat

a) As chaves de CONTRATOS são IdProj, IdForn e IdDepto.

b) Não havendo tabelas aninhadas e sendo os atributos atômicos, a relação atende à primeira forma normal. Sendo a chave primária IdCont dependente de IdProj e IdMat, e sendo IdMat dependente de IdForn e IdDepto entende-se que a relação possui a chave composta (IdProj, IdForn, IdDepto). Assim, a relação contratos não atende à segunda forma normal.

c) Passando para a segunda forma normal, tem-se:
(IdCont, Quant, Valor)
(IdProj, IdMat, IdCont)
(IdProj, IdDepto, IdMat)

3- a) (select a.anome from Aula as a where sala = ‘R128’) UNION (select a.nome from Aula as a, Matriculado as m where count (m.est_id) >= 5 and a.anome = m.anome);

b) select p.profnome from Professor as p, Aula as a where p.profid = a.profid IN (select count (est_id) from Matriculado as m, Aula as a where count (m.est_id) < 5 and a.anome = m.anome);

c) select e.estnome from Estudante as e, Matriculado as m where count(m.estid) = (select count (a.anome) from Aula as a);

4 - Restrição selecionada: Triggers em PostgreSQL

a)
```
CREATE OR REPLACE FUNCTION AtualizaSalario()
        RETURNS TRIGGER AS $$
BEGIN
        UPDATE Emp
                SET salario = 20000
                WHERE salario = 20000;
        RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER ControleSalarioMaximo
BEFORE INSERT OR UPDATE ON Emp
FOR EACH STATEMENT
EXECUTE PROCEDURE AtualizaSalario();
```

b)
```
CREATE OR REPLACE FUNCTION AtualizaPorcentagem()
        RETURNS TRIGGER AS $$
BEGIN
        UPDATE Trabalha
                SET pct_tempo = 100
                WHERE pct = 100;
        RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER ControlePorcentagemTrabalho
BEFORE INSERT OR UPDATE ON Trabalha
FOR EACH STATEMENT
EXECUTE PROCEDURE AtualizaPorcentagem();
```

c)
```
CREATE OR REPLACE FUNCTION AlteraSalarioGerente()
        RETURNS TRIGGER AS $$
BEGIN
        IF New.salario > (SELECT E.salario from Dpt as D, Emp as E
                                WHERE D.deptid = New.deptid AND
                                D.gerenteid = E.empid)
                UPDATE Emp as E, Dpt as D
                        SET E.salario = New.salario
                        WHERE D.deptid = New.deptid AND
                        D.gerenteid = E.empid
                RETURN NEW;
        END IF;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION ControlaOrcamento()
```

```
        RETURNS TRIGGER AS $$
BEGIN
        UPDATE Dept as D, Emp as E
                SET D.orcamento = (SELECT SUM(salario) FROM New.E)
                WHERE D.deptid = New.deptid
        RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER AlteracaoSalarioFuncionario
BEFORE INSERT OR UPDATE OF salario ON FUNCIONARIO
FOR EACH ROW
WHEN (NEW.Salario > OLD.Salario)
        EXECUTE PROCEDURE ControlaOrcamento();
        EXECUTE PROCEDURE AlteraSalarioGerente();
```