Universidade de São Paulo
Escola de Artes, Ciências e Humanidades
Disciplina: Laboratório de Banco de Dados
Profª Dra. Fátima Nunes.

Administração de Condomínio

Parte II

Fernando K. G. de Amorim – 10387644
João Guilherme da Costa Seike – 9784634
Lucas Pereira Castelo Branco – 10258772
Victor Gomes de O. M. Nicola – 9844881

São Paulo
2019

Nesta segunda parte do trabalho para a matéria de Laboratório de Banco de Dados, precisamos enunciar algumas regras de negócio para fazer asserções, *triggers* e visões. Esta segunda parte foi subdividida em outras quatro partes segundo o enunciado.

LETRA A

Regras de negócios para a parte (a):

1. Um membro do corpo administrativo só pode ter dois mandatos seguidos para a mesma função, após isso ele deve ficar um ano sem poder ser nomeado para a mesma função. Só pode haver uma eleição por ano e não podem haver inserções para um ano anterior ao atual.

Enunciado textual

Seguindo regras condominiais, um membro do corpo administrativo só pode ser eleito duas vezes seguidas para a mesma função, podendo se candidatar novamente para a mesma função após um ano sem ser eleito para aquela função específica. Só ocorre uma eleição de um corpo administrativo, e não obstante, devem ser impedidas entradas de eleições de anos anteriores ao ano atual vigente, as eleições só podem ser feitas a partir do ano atual. Para verificar se a eleição está de acordo com as regras, são verificadas as datas e a quantidade de vezes que aquela pessoa foi eleita para os últimos dois anos.

Solução textual em SQL padrão

```
CREATE ASSERTION eleicao_corpo_adm CHECK (
NOT EXISTS ( SELECT * FROM corpo_administrativo WHERE
( IF ( NEW.data_eleicao (YEAR) >= NOW() (YEAR) ) ) AND
( IF NOT EXISTS ( SELECT * FROM corpo_administrativo c_adm
        WHERE c_adm.data_eleicao (YEAR) =
        NEW.data_eleicao (YEAR) ) ) AND
( IF ( SELECT COUNT(*) FROM edua WHERE edua.id_sindico = NEW.id_sindico) < 2) AND
( IF ( SELECT COUNT(*) FROM edua WHERE edua.id_subsindico = NEW.id_subsindico) < 2 ) AND
( IF ( SELECT COUNT(*) FROM edua WHERE edua.id_conselheiro_1 = NEW.id_conselheiro_1) < 2 ) AND
( IF ( SELECT COUNT(*) FROM edua WHERE edua.id_conselheiro_2 = NEW.id_conselheiro_2) < 2 ) AND
( IF ( SELECT COUNT(*) FROM edua WHERE edua.id_conselheiro_3 = NEW.id_conselheiro_3) < 2 ) );
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```sql
CREATE OR REPLACE FUNCTION fc_deleta_apartamentos()
RETURNS trigger AS $tr_dapartamentos$
BEGIN

    CREATE TABLE table_holder AS (SELECT fk_id_moradia FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio);


    DELETE FROM adm_condominio.Moradia_Pessoa WHERE fk_id_moradia IN
        (SELECT id_moradia from adm_condominio.Moradia WHERE id_moradia IN (SELECT * FROM table_holder));

    DELETE FROM adm_condominio.Condominio_Moradia WHERE  fk_id_moradia IN (SELECT * FROM table_holder);
    DELETE FROM adm_condominio.Apartamento WHERE  fk_id_moradia IN (SELECT * FROM table_holder);
    DELETE FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio;
    DELETE FROM adm_condominio.Moradia WHERE  id_moradia IN (SELECT * FROM table_holder);
    DROP TABLE table_holder;
RETURN NEW;
END;
$tr_dapartamentos$ LANGUAGE plpgsql;

CREATE TRIGGER tr_deleta_apartamentos
AFTER DELETE ON adm_condominio.Edificio
FOR EACH ROW
EXECUTE PROCEDURE fc_deleta_apartamentos();
```

```sql
CREATE OR REPLACE FUNCTION eleicao_corpo_adm() RETURNS TRIGGER AS $$
BEGIN
    DROP TABLE IF EXISTS elec_dois_ult_anos;
    CREATE TEMP TABLE elec_dois_ult_anos
    (
        id_corpo INT,
        id_sindico INT,
        id_subsindico INT,
        id_conselheiro_1 INT,
        id_conselheiro_2 INT,
        id_conselheiro_3 INT
    );

    INSERT INTO elec_dois_ult_anos
    SELECT id_corpo, id_sindico, id_subsindico, id_conselheiro_1, id_conselheiro_2, id_conselheiro_3
    FROM adm_condominio.corpo_administrativo adm_ca
    WHERE DATE_PART('year', adm_ca.data_eleicao) = DATE_PART('year', (NEW.data_eleicao - interval '1 year'))
    UNION
    SELECT id_corpo, id_sindico, id_subsindico, id_conselheiro_1, id_conselheiro_2, id_conselheiro_3
    FROM adm_condominio.corpo_administrativo adm_ca
    WHERE DATE_PART('year', adm_ca.data_eleicao) = DATE_PART('year', (NEW.data_eleicao - interval '2 years'));


    -- verificar ano de eleicao
    IF (DATE_PART('year', NEW.data_eleicao) >= DATE_PART('year',NOW())) THEN

    -- verifica se ja ocorreu uma eleicao naquele ano
    IF NOT EXISTS
    (
        SELECT *
        FROM adm_condominio.corpo_administrativo adm_ca
        WHERE DATE_PART('year', adm_ca.data_eleicao) = DATE_PART('year', NEW.data_eleicao)
    )
    THEN
```

```
-- verifica síndico ja foi eleito 2 vezes
IF
(
    SELECT COUNT(*) FROM elec_dois_ult_anos edua
    WHERE edua.id_sindico = NEW.id_sindico
) < 2 THEN

-- verifica sub-síndico ja foi eleito 2 vezes
IF
(
    SELECT COUNT(*) FROM elec_dois_ult_anos edua
    WHERE edua.id_subsindico = NEW.id_subsindico
) < 2 THEN

-- verifica conselheiro 1 ja foi eleito 2 vezes
IF
(
    SELECT COUNT(*) FROM elec_dois_ult_anos edua
    WHERE edua.id_conselheiro_1 = NEW.id_conselheiro_1
) < 2 THEN


-- verifica conselheiro 2 ja foi eleito 2 vezes
IF
(
    SELECT COUNT(*) FROM elec_dois_ult_anos edua
    WHERE edua.id_conselheiro_2 = NEW.id_conselheiro_2
) < 2 THEN


-- verifica conselheiro 3 ja foi eleito 2 vezes
IF
(
    SELECT COUNT(*) FROM elec_dois_ult_anos edua
    WHERE edua.id_conselheiro_3 = NEW.id_conselheiro_3
) < 2 THEN

    RETURN NEW;
```

```
    -- else conselheiro 3 ja eleito
    ELSE RAISE EXCEPTION 'O conselheiro 3 já foi eleito duas vezes seguidas nos últimos anos!';
    END IF;
    -- else conselheiro 2 ja eleito
    ELSE RAISE EXCEPTION 'O conselheiro 2 já foi eleito duas vezes seguidas nos últimos anos!';
    END IF;
    -- else conselheiro 1 ja eleito
    ELSE RAISE EXCEPTION 'O conselheiro 1 já foi eleito duas vezes seguidas nos últimos anos!';
    END IF;
    -- else subsindico ja eleito
    ELSE RAISE EXCEPTION 'O sub-síndico já foi eleito duas vezes seguidas nos últimos anos!';
    END IF;
    -- else sindico ja eleito
    ELSE RAISE EXCEPTION 'O síndico já foi eleito duas vezes seguidas nos últimos anos!';
    END IF;
    -- else ja ocorreu eleicao
    ELSE RAISE EXCEPTION 'Um corpo já foi eleito esse ano!';
    END IF;
    -- else ano anterior ao atual
    ELSE RAISE EXCEPTION 'Você não pode eleger um corpo para um ano anterior ao atual!';
    END IF;

    DROP TABLE IF EXISTS elec_dois_ult_anos;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER verifica_eleicao BEFORE INSERT ON adm_condominio.corpo_administrativo
    FOR EACH ROW EXECUTE PROCEDURE eleicao_corpo_adm();
```

Transcrição

A asserção precisa primeiro ter a declaração de uma tabela temporária com os dados do SELECT apresentado, que insere as tuplas dos dois últimos anos relacionados às eleições.
As verificações começam da seguinte forma:

1. É criada a tabela temporária e os dados são inseridas na mesma através de INSERT INTO com SELECT e um UNION, o só é possível encadear tuplas de anos diferentes com o UNION para este caso.
2. É verificado se o ano da eleição é igual ou superior ao ano atual.
3. É verificado com base na data da inserção a ser feita, se já foi eleito um corpo administrativo no ano inserido.
4. Começam então uma série de verificações para cada um dos cargos do corpo a ser inserido, caso um dos cargos tenha uma pessoa que esteja sendo eleita, e já possui dois mandatos seguidos dos dois últimos anos, uma exceção é gerada, aquela pessoa não pode ser elegida por cerca de um ano.
5. As exceções se aplicam a cada um dos cargos e caso as datas não passem nas verificações também.
6. Por fim a tabela temporária é excluída.

Está incluso também a criação do *trigger* para a asserção, o *trigger* é executado para cada tupla e antes de ser inserido.

Regras de negócios para a parte (a):

2. Um condômino só pode reservar até 3 espaços do condomínio no máximo por mês do ano. O condômino não pode reservar mais de um espaço por dia. O condomínio deve garantir que só

deve haver uma reserva de um dado espaço no dia, que deve se encerrar até as 22h e deve durar até no máximo 6h de duração.

Enunciado textual

Para que um condomínio tenha um controle das reservas dos espaços, são necessários também que algumas regras sejam seguidas como em qualquer outro local da sociedade. Respeitando a lei do silêncio, as reservas dos espaços devem se encerrar até as 22h do mesmo dia da reserva e cada reserva deve possuir duração máxima de 6 horas para não atrapalhar outros moradores. Claro que para isso, é necessário que não ocorram conflitos de reservas e datas. Uma pessoa que já alugou um espaço no dia, não pode alugar outro para dar chance aos outros moradores de poderem alugar. O espaço só pode ser reservado uma vez por dia, ou seja, caso o mesmo tenha sido alugado, não poderá ser alugado novamente no dia. E por fim, o condômino só pode alugar um máximo de 3 vezes por mês em seu condomínio, garantindo que não fique alugando espaços todos os dias.

Solução textual em SQL padrão

```
CREATE ASSERTION reserva_espaco_unico CHECK (
        NOT EXISTS ( SELECT * FROM reserva WHERE
        ( IF ( SELECT NEW.hora_final <= '22:00:00')) AND
        ( IF ( SELECT NEW.hora_final - NEW.hora_inicial <= '06:00:00' )) AND
        ( IF NOT EXISTS ( SELECT * FROM reserva WHERE fk_id_pessoa = NEW. fk_id_pessoa AND
                        data = NEW.data )) AND
        ( IF NOT EXISTS ( SELECT * FROM reserva WHERE fk_id_espaco = NEW.fk_id_espaco AND
                        data = NEW.data )) AND
        ( IF ( SELECT COUNT(*) FROM reserva
                WHERE fk_id_pessoa = NEW.fk_id_pessoa AND
                data (MONTH) = NEW.data (MONTH) AND
                data (YEAR) = NEW.data (YEAR)) < 3 ));
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```sql
CREATE OR REPLACE FUNCTION reserva_espaco_unico() RETURNS TRIGGER AS $$
BEGIN
    -- verifica se o horario de encerramento da reserva eh superior a 22h
    IF
    (SELECT NEW.hora_final) <= time '22:00:00' THEN

    -- verifica se a duracao da reserva eh maior que 6h
    IF
    (SELECT NEW.hora_final - NEW.hora_inicial) <= interval '06:00:00' THEN

    -- verifica se ele ja alugou um espaco no dia
    IF NOT EXISTS
    (
        SELECT * FROM adm_condominio.reserva adm_r
        WHERE adm_r.fk_id_pessoa = NEW.fk_id_pessoa AND
        adm_r.data = NEW.data
    ) THEN

    -- verifica se o espaco ja foi alugado naquele dia
    IF NOT EXISTS
    (
        SELECT * FROM adm_condominio.reserva adm_r
        WHERE adm_r.fk_id_espaco = NEW.fk_id_espaco AND
        adm_r.data = NEW.data
    ) THEN
```

```sql
    -- verifica se ele ja alugou 3 espaços no mes do mesmo ano
    IF
    (
        SELECT COUNT(*) FROM adm_condominio.reserva adm_r
        WHERE adm_r.fk_id_pessoa = NEW.fk_id_pessoa AND
        DATE_PART('month', adm_r.data) = DATE_PART('month', NEW.data) AND
        DATE_PART('year', adm_r.data) = DATE_PART('year', NEW.data)
    ) < 3 THEN
            RETURN NEW;

    -- else pessoa ja tem mais que 3 reservas
    ELSE RAISE EXCEPTION 'O condômino já tem mais que 3 reservas!';
    END IF;
    -- else espaco reservado no dia
    ELSE RAISE EXCEPTION 'Esse espaço já foi reservado neste dia!';
    END IF;
    -- else condomino ja reservou um espaco no dia
    ELSE RAISE EXCEPTION 'Este condômino já reservou um espaço hoje!';
    END IF;
    -- else duracao 6h
    ELSE RAISE EXCEPTION 'A duração da reserva ultrapassa 6 horas!';
    END IF;
    -- else hora final 22h
    ELSE RAISE EXCEPTION 'A hora final da reserva deve ser até 22h!';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER reserva_espacos BEFORE INSERT ON adm_condominio.reserva
    FOR EACH ROW EXECUTE PROCEDURE reserva_espaco_unico();
```

Transcrição

O código implementado funciona da seguinte maneira:

1. É feita uma verificação para se o horário final da reserva é menor ou igual a 22h, que é o horário do silêncio.
2. A segunda verificação feita é em relação ao horário de duração da reserva, o limite de duração para uma reserva é de 6h.
3. Como terceira verificação, é executado uma operação de SELECT para verificar se não existe nenhuma reserva daquele condômino na mesma data, caso exista, será impedido de fazê-la.
4. Na quarta verificação, um outro SELECT é executado para verificar se o espaço já foi alugado naquele dia por outra pessoa.
5. E por último, contabiliza quantas vezes o condômino fez solicitações de reserva, e caso passem de 3, ele não permite que ele faça outra reserva, sendo necessário esperar o próximo mês do ano.

Após isso, é criado o *trigger* com a asserção descrita na função acima, executando o *trigger* sempre antes da inserção e para cada tupla.

LETRA B

Regras de negócios para a parte (b):

3. Ao associar um edifício novo ao condomínio, todos os apartamentos devem ser cadastrados de maneira automática e ordenada de acordo com o número do andar e com o número final de cada apartamento.

Enunciado textual

Para evitar esforço desnecessário durante a inserção de novos edifícios, onde seria necessário também realizar a inserção de todos os apartamentos deste edifício, foi implementado um gatilho que ao receber uma inserção de um edifício, automaticamente insere todos os apartamentos deste edifício na tabela “Apartamento”.

Solução textual em SQL padrão

```
CREATE OR REPLACE TRIGGER tr_insere_apartamentos
AFTER INSERT OF adm_condominio.Edificio
FOR EACH ROW
DECLARE
   id_max integer;
BEGIN
   SELECT max(id_moradia) into id_m from adm_condominio.Moradia;
   FOR i in 1 .. new.andares
```

```
  LOOP
    FOR j in 1 .. new.qtd_finais
    LOOP
      id_m := id_m + 1;
      INSERT INTO adm_condominio.Moradia (tipo_moradia, id_moradia) VALUES
        ('a', (id_m));
      INSERT INTO adm_condominio.Moradia_Edificio (fk_id_edificio, fk_id_moradia) VALUES
        (new.id_edificio, (id_m));
      INSERT INTO adm_condominio.Condominio_Moradia (fk_id_condominio, fk_id_moradia) VALUES
        (new.fk_id_condominio, (id_m));
      INSERT INTO adm_condominio.Apartamento (fk_id_moradia,andar, final, numero_ap) VALUES
        (id_m, i, j, CAST(CONCAT(CAST(i as varchar(10)), CAST(j as varchar(10))) AS INT));
    END LOOP;
  END LOOP;
END;
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```sql
CREATE OR REPLACE FUNCTION fc_insere_apartamentos()
RETURNS trigger AS $tr_apartamentos$
DECLARE
    id_max integer;
BEGIN
    SELECT max(id_moradia) into id_m from adm_condominio.Moradia;

    FOR i in 1 .. new.andares
    LOOP
        FOR j in 1 .. new.qtd_finais
        LOOP

            id_m := id_m + 1;

            INSERT INTO adm_condominio.Moradia_Edificio (fk_id_edificio, fk_id_moradia) VALUES
                (new.id_edificio, (id_m));
            INSERT INTO adm_condominio.Condominio_Moradia (fk_id_condominio, fk_id_moradia) VALUES
                (new.fk_id_condominio, (id_m));

            INSERT INTO adm_condominio.Moradia (tipo_moradia, id_moradia) VALUES
                ('a', (id_m));

            INSERT INTO adm_condominio.Apartamento (fk_id_moradia,andar, final, numero_ap) VALUES
                (id_m, i, j, CAST(CONCAT(CAST(i as varchar(10)), CAST(j as varchar(10))) AS INT));
        END LOOP;
    END LOOP;

RETURN NEW;
END;
$teste_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER tr_insere_apartamentos
AFTER INSERT ON adm_condominio.Edificio
FOR EACH ROW
EXECUTE PROCEDURE tr_insere_apartamentos();
```

Regras de negócios para a parte (b):

O código implementado funciona da seguinte maneira:
1- É criado a função fc_insere_apartamentos que será executada pelo trigger.
2- A função armazena o maior id_moradia da tabela adm_condominio.Moradia na variável id_m
3-a função entrea em um loop que vai iterar de 1 até a quantidade de andares do edifício inserido e a cada iteração do andar, é feita uma iteração de 1 até o numero de qtd_finais.
4- Dentro do loop de qtd_finais, o valor de id_m é incrementado em seguida é inserido o apartamento no banco na seguinte ordem: insere na tabela Moradia, insere na tabela Edificio_Moradia, insere Condominio_Moradia (sempre passando o valor do id incrementado) e finalmente insere em Apartamento o id e o numero do apartamento, que é a concatenação dos valores (i,j) da iteração atual (andar atual + numero do apartamento atual) que resulta no número completo do apartamento.
5-Por fim, é criado um trigger chamado tr_insere que será ativado quando toda vez que acontecer um Insert na tabela adm_condominio.Edificio e executará a função descrita acima(APÓS a inserção).

Regras de negócios para a parte (b):

4. Ao deletar um edifício da tabela Edificío, além de deletar todos os apartamentos vinculados a esse edifício na tabela Apartamento, também devem ser deletados todas as pessoas que estão relacionadas com esses apartamentos da tabela Pessoas.

Enunciado textual

Caso um edifício deixe de ser propriedade do condomínio (em caso de venda ou demolição do edifício, por exemplo), deve haver uma deleção na tabela Edifício que ativará um gatilho que automaticamente deleta todos os apartamentos deste edifício e todas as pessoas que moravam nesse edifício do banco de dados do condomínio.

Solução textual em SQL padrão

```
CREATE TRIGGER tr_deleta_apartamentos
AFTER DELETE OF adm_condominio.Edificio
FOR EACH ROW
BEGIN
    CREATE TABLE table_holder AS (SELECT fk_id_moradia FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio);
   DELETE FROM adm_condominio.Moradia_Pessoa WHERE fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
      DELETE FROM adm_condominio.Condominio_Moradia WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
    DELETE FROM adm_condominio.Apartamento WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio;
    DELETE FROM adm_condominio.Moradia WHERE  id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DROP TABLE table_holder;
END;
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```
CREATE OR REPLACE FUNCTION fc_deleta_apartamentos()
RETURNS trigger AS $tr_dapartamentos$
BEGIN

  CREATE TABLE table_holder AS (SELECT fk_id_moradia FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio);


  DELETE FROM adm_condominio.Moradia_Pessoa WHERE fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);

  DELETE FROM adm_condominio.Condominio_Moradia WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Apartamento WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio;
  DELETE FROM adm_condominio.Moradia WHERE  id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DROP TABLE table_holder;
RETURN NEW;
END;
$tr_dapartamentos$ LANGUAGE plpgsql;

CREATE TRIGGER tr_deleta_apartamentos
before DELETE ON adm_condominio.Edificio
FOR EACH ROW
EXECUTE PROCEDURE fc_deleta_apartamentos();
```

Regras de negócios para a parte (b):

O código implementado funciona da seguinte maneira:

1- É criado a função fc_deleta_apartamentos que será executada pelo trigger.
2-Essa função cria uma tabela temporária e coloca todos os “id_moradia” da tabela Moradia_Edificio pois serão essas moradias que serão deletadas.
3-A tabela temporária é usada para deletar as informações referentes ao edifício deletado em outras tabelas, decidimos usar uma tabela temporária para armazenar os IDs das moradias deletados pois não era possível guardar todas essas informações em uma só variável.
4- Então é feito a deleção nas outras tabelas na seguinte ordem: Condominio_Moradia, Apartamento, Moradia_Edificio e Moradia.
5- Por fim é criado um gatilho que será ativado após acontecer um DELETE na tabela adm_condominio.Edificio e executará a função descrita acima(ANTES da deleção).

LETRA C

Especificação do Requisito

Para esse requisito, levantou-se a necessidade de ter-se a relação das pessoas e seus bens dentro do condomínio. É importante ressaltar a importância do sigilo desses dados entre os envolvidos, visto a questão da privacidade de dados tão presente. Para tal, podemos definir formalmente:

“O sistema deverá ser capaz de permitir a filiais interessadas que haja uma relação dos condôminos que estão cadastrados dentro de condomínios ligados a sua administração, correlacionando com todos os bens que estão sob sua custódia.”

Solução em SQL Padrão:

```
CREATE VIEW relatorio_pessoas_filial AS
        SELECT  pessoa.nome AS nome,
                        pessoa.cpf AS cpf,
                        (CASE WHEN moradia.tipo_moradia = 'c'
                                THEN 'Casa'
                                 WHEN moradia.tipo_moradia = 'a'
```

```
                                THEN 'Apartamento'
                                ELSE NULL
                                END
                        ) AS tipo_moradia,
                        (CASE WHEN tipo_moradia = 'Casa'
                                THEN casa.numero_casa
                                ELSE apartamento.numero_ap --Assumindo que toda moradia que
não é uma casa, é um apartamento
                                END
                        ) AS numero,
                        apartamento.andar,
                        apartamento.final,
                        veiculo.placa,
                        veiculo.marca,
                        veiculo.modelo--,
                        --COALESCE(condominio_moradia.fk_id_condominio,
edificio.fk_id_condominio)
                        --      AS final_fk_id_condominio --Auxiliar para nossa query abaixo

                        FROM adm_condominio.pessoa AS pessoa

                        --Dando JOIN na parte de moradia
                        JOIN adm_condominio.moradia_pessoa AS moradia_pessoa
                          ON moradia_pessoa.fk_id_pessoa = pessoa.id_pessoa
                        JOIN adm_condominio.moradia AS moradia
                          ON moradia.id_moradia = moradia_pessoa.fk_id_moradia
                         --Temos LEFT JOIN aqui, pois a pessoa só terá um deles
        LEFT JOIN adm_condominio.casa AS casa
                          ON casa.fk_id_moradia = moradia.id_moradia
        LEFT JOIN adm_condominio.apartamento AS apartamento
                          ON apartamento.fk_id_moradia = moradia.id_moradia

                          -- Bloco de JOINs para conseguirmos testar o condominio e a filial na qual ele
pertence
                        JOIN adm_condominio.condominio_moradia AS condominio_moradia
                          ON condominio_moradia.fk_id_moradia = moradia.id_moradia
                        JOIN adm_condominio.condominio AS condominio
                          ON condominio.id_condominio  = condominio_moradia.fk_id_condominio
                        JOIN adm_condominio.condominio_filial AS condominio_filial
                          ON condominio_filial.fk_id_condominio = condominio.id_condominio

                         -- Temos LEFT JOIN porque a pessoa pode não ter veiculo
        LEFT JOIN adm_condominio.veiculo_moradia AS veiculo_moradia
                          ON veiculo_moradia.fk_id_moradia = moradia.id_moradia
        LEFT JOIN adm_condominio.veiculo AS veiculo
                         ON veiculo.id_veiculo = veiculo_moradia.fk_id_veiculo

                        -- LIMITADOR DE SEGURANÇA: vemos apenas a lista de pessoas ligada a essa
filial
                        WHERE condominio_filial.fk_id_filial = 5
```

Solução em código implementada

```sql
CREATE VIEW relatorio_pessoas_filial AS
    SELECT  pessoa.nome AS nome,
            pessoa.cpf AS cpf,
            (CASE WHEN moradia.tipo_moradia = 'c'
                THEN 'Casa'
                  WHEN moradia.tipo_moradia = 'a'
                THEN 'Apartamento'
                ELSE NULL
                END
             ) AS tipo_moradia,
            (CASE WHEN tipo_moradia = 'Casa'
                THEN casa.numero_casa
                ELSE apartamento.numero_ap --Assumindo que toda moradia que não é uma casa, é um apartamento
                END
            ) AS numero,
            apartamento.andar,
            apartamento.final,
            veiculo.placa,
            veiculo.marca,
            veiculo.modelo--,
            --COALESCE(condominio_moradia.fk_id_condominio, edificio.fk_id_condominio)
            --  AS final_fk_id_condominio --Auxiliar para nossa query abaixo

            FROM adm_condominio.pessoa AS pessoa

            --Dando JOIN na parte de moradia
            JOIN adm_condominio.moradia_pessoa AS moradia_pessoa
              ON moradia_pessoa.fk_id_pessoa = pessoa.id_pessoa
            JOIN adm_condominio.moradia AS moradia
              ON moradia.id_moradia = moradia_pessoa.fk_id_moradia
             --Temos LEFT JOIN aqui, pois a pessoa só terá um deles
        LEFT JOIN adm_condominio.casa AS casa
              ON casa.fk_id_moradia = moradia.id_moradia
        LEFT JOIN adm_condominio.apartamento AS apartamento
              ON apartamento.fk_id_moradia = moradia.id_moradia

              -- Bloco de JOINs para conseguirmos testar o condominio e a filial na qual ele pertence
            JOIN adm_condominio.condominio_moradia AS condominio_moradia
              ON condominio_moradia.fk_id_moradia = moradia.id_moradia
            JOIN adm_condominio.condominio AS condominio
              ON condominio.id_condominio  = condominio_moradia.fk_id_condominio
            JOIN adm_condominio.condominio_filial AS condominio_filial
              ON condominio_filial.fk_id_condominio = condominio.id_condominio

             -- Temos LEFT JOIN porque a pessoa pode não ter veiculo
        LEFT JOIN adm_condominio.veiculo_moradia AS veiculo_moradia
              ON veiculo_moradia.fk_id_moradia = moradia.id_moradia
        LEFT JOIN adm_condominio.veiculo AS veiculo
             ON veiculo.id_veiculo = veiculo_moradia.fk_id_veiculo

            -- LIMITADOR DE SEGURANÇA: vemos apenas a lista de pessoas ligada a essa filial
            WHERE condominio_filial.fk_id_filial = 5
```

Transcrição do código

A ideia é que, a partir dessa lista, consigamos fazer toda a lista dos itens que estão cadastrados no sistema que estão ligados a pessoa, ou seja, a sua moradia e seu veículo.
Para que isso fosse possível, as três tabelas básicas usadas foram ‘pessoa’, ‘moradia’ e ‘veículo’. Porém, para a restrição, foi utilizado também a tabela ‘condomínio’.

Durante a etapa de selecionar os campos, fazemos um tratamento simples para que o campo se torne mais legível ao usuário final. Podemos observar este caso em ‘tipo_moradia”, onde fazemos uma leve conversão de acordo com o caractere registrado. Outro tratamento realizado para garantir que haja uma tabela mais legível é a existência de um campo flexível: em ‘número”, realizamos um teste justamente com o tipo de moradia para permitir que o campo tenha significado de acordo com o contexto da linha.

Já na parte de agrupar as tabelas, podemos dividir em quatro blocos, como explicado no primeiro parágrafo. Na primeira parte, pegamos como base a tabela 'pessoa'. A partir dele, a segunda parte realiza um INNER JOIN com 'moradia', pois apenas teremos registros da pessoa se necessariamente ele possuir uma moradia. Porém, no conceito de 'casa' e 'apartamento', pode-se dizer que uma pessoa muito possivelmente terá apenas um deles, sendo assim necessário um LEFT JOIN para garantir a funcionalidade.
Por fim, no bloco usado para a filtragem, usamos de JOIN para permitir uma ligação entre a moradia e o 'condomínio' na qual está presente.

Para essa VIEW em específico, assumiu-se que a Filial de número 5 no sistema fora a solicitadora principal dessa possibilidade, e portanto, o WHERE se baseia nele. Para fins práticos, assumiu-se que apenas ela será consultada.

Custo-Benefício e Atualização de Dados

O ganho em custo-benefício é evidente. A complexidade de montar uma query nesse formato mostra quão prático pode ser termos uma tabela pronta, que pode ser acionada em uma operação SELECT simples. Quando contamos com a presença de um modelo que possui sentido prático para o usuário final dentro de seu caso de uso, o fato de permitirmos uma visão pré-estabelecida facilita sua construção a longo prazo. Em questão de funcionalidade, também permite com que tenhamos variações dessa view para outros, já que sua função é garantir que cada filial tenha visão apenas de seus próprios condôminos.
A atualização de dados não é tão constante quanto à consulta, mas sua importância é inegável: a filial deve manter uma relação de seus condôminos, tanto por questões financeiras (garantindo que todos estão pagos) como para questões legais (que podem ir desde a questão de ter-se os dados das pessoas em casos de crimes até verificação de quebras de contrato).

LETRA D

Especificação do Requisito

Uma das consultas mais comuns em bancos de dados como esse é justamente o gerenciamento do fluxo de entrada e saída aos condomínios. Por isso, é interessante haver alguma forma de já termos, condensado em uma tabela, alguma forma de visualizarmos isso de forma mais abrangente, de forma a permitir sua manipulação quando necessário.
Para isso, contamos com três tabelas: 'entrada_saida' é a principal, responsável por identificar o momento da ação e seu tipo; e o que chamaremos de ator, que pode ser uma pessoa ou um veículo.
O requisito, formalizado, será:
"O sistema deve ser capaz de encontrar, com uma latência de menos de um segundo, os registros de fluxo de condôminos, de forma a permitir não apenas identificá-los, como também verificar quais as últimas ocorrências na qual o sistema tem ciência."

Solução em SQL Padrão:

```
CREATE VIEW es_ator AS
        SELECT   entrada_saida.id_es AS id_es,
                        entrada_saida.data_hora AS data_hora,
                        (CASE WHEN entrada_saida.acao = 'e'
                                THEN 'Entrada'
                                WHEN entrada_saida.acao = 's'
```

```
                THEN 'Saída'
                ELSE NULL
                END
        ) AS acao,
        COALESCE(pessoa.id_pessoa, veiculo.id_veiculo) AS fk_id_ator,
        COALESCE(pessoa.cpf, veiculo.placa) AS registro_ator
        FROM adm_condominio.entrada_saida AS entrada_saida
        LEFT JOIN adm_condominio.es_pessoa AS es_pessoa
                ON es_pessoa.fk_id_es = entrada_saida.id_es
        LEFT JOIN adm_condominio.pessoa AS pessoa
                ON pessoa.id_pessoa = es_pessoa.fk_id_pessoa
        LEFT JOIN adm_condominio.es_veiculo AS es_veiculo
                ON es_veiculo.fk_id_es = entrada_saida.id_es
        LEFT JOIN adm_condominio.veiculo AS veiculo
                ON veiculo.id_veiculo = es_veiculo.fk_id_veiculo

        ORDER BY 2 DESC
```

Solução em código implementada

Para tal caso, temos a seguinte visão:

```sql
CREATE VIEW es_ator AS
    SELECT  entrada_saida.id_es AS id_es,
            entrada_saida.data_hora AS data_hora,
            (CASE WHEN entrada_saida.acao = 'e'
                  THEN 'Entrada'
                  WHEN entrada_saida.acao = 's'
                  THEN 'Saída'
                  ELSE NULL
                  END
             ) AS acao,
            COALESCE(pessoa.id_pessoa, veiculo.id_veiculo) AS fk_id_ator,
            COALESCE(pessoa.cpf, veiculo.placa) AS registro_ator
            FROM adm_condominio.entrada_saida AS entrada_saida
            LEFT JOIN adm_condominio.es_pessoa AS es_pessoa
                  ON es_pessoa.fk_id_es = entrada_saida.id_es
            LEFT JOIN adm_condominio.pessoa AS pessoa
                  ON pessoa.id_pessoa = es_pessoa.fk_id_pessoa
            LEFT JOIN adm_condominio.es_veiculo AS es_veiculo
                  ON es_veiculo.fk_id_es = entrada_saida.id_es
            LEFT JOIN adm_condominio.veiculo AS veiculo
                  ON veiculo.id_veiculo = es_veiculo.fk_id_veiculo

            ORDER BY 2 DESC
```

Transcrição do código

Neste código, optamos por duas características: a presença de identificadores únicos ao sistema (visto sua utilidade posterior em consultas); e a aglutinação de informações relevantes ao usuário, de forma a permitir uma maior facilidade tanto para o mantenedor do banco quanto para o usuário final.

Para isso, criamos uma ‘entidade’, que é chamada ‘ator’, que representa quem de fato está ligado a aquela ação de entrada/saída, podendo ser uma pessoa ou um veículo. Dessa forma, usamos da função COALESCE, que irá obter a primeira informação não-nula do sistema, e a partir disso, exibir a informação relevante a aquele registro.

Custo-Benefício e Atualização de Dados

O custo-benefício é simples: a partir dessa visão, já temos a junção de informações simples, porém essenciais para a segurança do condomínio. Além disso, se quisermos eventualmente obter mais informações sobre a pessoa, ou sobre a moradia, por exemplo, podemos fazê-lo usando do ‘id_ator’ para fazermos os JOINs necessários e completar a consulta. Dessa forma, conseguimos atender de antemão ao anseio básico do administrador como também auxiliamos a construção de consultas mais complexas ao banco, em caso de necessidade.

Alguns exemplos de queries que podemos utilizar nesse caso são:

- *Query que realiza a listagem de todas as entradas e saídas e seus respectivos atores, considerando apenas aquelas que ocorreram antes do período de 2 dias atrás.*

```
SELECT *
FROM es_ator
    WHERE data_hora < NOW() - interval '2 days'
```

Para essa query, usamos do recurso NOW(), que registra o horário atual do sistema na qual o SGBD está inserido. A partir de ‘interval’ permite com que façamos operações com ela.

- *Conta o número de registros de entrada e saída por tipo de moradia.*

```
SELECT   moradia.tipo_moradia,
                COUNT(es_ator.id_es)
FROM es_ator
LEFT JOIN adm_condominio.veiculo_moradia AS veiculo_moradia
          ON es_ator.fk_id_ator = veiculo_moradia.fk_id_veiculo
LEFT JOIN adm_condominio.moradia_pessoa AS moradia_pessoa
          ON es_ator.fk_id_ator = moradia_pessoa.fk_id_pessoa
JOIN adm_condominio.moradia AS moradia
          ON moradia.id_moradia IN (veiculo_moradia.fk_id_moradia, moradia_pessoa.fk_id_moradia)
        GROUP BY 1
```

Nessa query, aproveitamos do fato de ‘ator’ poder representar duas entidades para fazer uma contagem total ligada a outra entidade, ‘moradia’, que possui relacionamento tanto com ‘pessoa’ quanto com ‘veiculo’.

A questão da atualização de dados é latente. A tabela de entrada e saída tende a ser a mais modificada do banco, ao se tratar de um registro de atividades dos condomínios. Portanto, a presença de uma visão como essa privilegia que haja um acompanhamento *real-time* dessa atualização e seus pares pertinentes (no caso, o envolvido na ação em questão). Por isso, não é uma boa ideia torná-la uma view persistente no banco, visto sua volatilidade constante.