Puc-Rio

# BANCO DE DADOS
# ADMINISTRADORA DE IMÓVEIS

**Lucas Hardman**
**Pedro Augusto**
**Pedro Gomes**
**Stephanie Fay**

2/07/2015

# CONTEÚDO

# I. INTRODUÇÃO

## Administração de Imóveis

Nosso projeto teve em mente uma Administradora de Imóveis e seus funcionários como usuários. Assim criamos um modo para que possam administrar, organizar, arquivar e procurar qualquer informação importante sobre seus negócios atuais ou antigos.

Começamos fazendo um modelo lógico amigável que representa toda a estrutura do projeto, que se resume em várias relações da entidade Pessoa com a entidade Imóvel. Então passamos para a parte lógica, nos certificando que todas relações e entidades estivessem condizentes com nosso objetivo.

Este Banco de Dados se baseia em quatro simples relações: “É dono”, “Aluga”, “Mora em” e “Administra”. As duas primeiras relacionam a entidade Pessoa com a entidade Imóvel, pois ambos tipos de Pessoa podem ser donas e alugar um imóvel. Já “Mora em” ficou responsável pelo relacionamento de Física e Imóvel, pois não é possível que existam moradores jurídicos. Assim a especialização Jurídica que representa as empresas responsáveis pela administração do condomínio de um imóvel, caso haja um, ficou com a relação “Administra”.

# II. REQUISITOS PARA A MODELAGEM CONCEITUAL

O primeiro passo foi a entrevista com um membro de uma Administradora de Imóveis. Com as informações obtidas desenvolvemos um modelo conceitual do projeto. O modelo retrata o esquema de relação entre uma **Pessoa**, jurídica ou física, e um **Imóvel**, seja ele uma casa, apartamento, galpão, etc.

Primeiramente observamos que era necessário o cadastro de uma **Pessoa**. Concluímos que Nome Completo, Endereço, Dados Bancários, Email, Telefone e um Id na Administradora eram comuns as pessoas jurídicas e físicas. Então as dividimos de acordo com suas necessidades especiais.

- Uma Pessoa Física possui um CPF e uma Identificação como por exemplo o RG mais a UF. Documentos importantes para identificação em qualquer tipo de contrato de negócios.
- Uma Pessoa Jurídica possui um CNPJ, um responsável e uma URL. O responsável é aquele que assina o contrato feito com a Administradora.

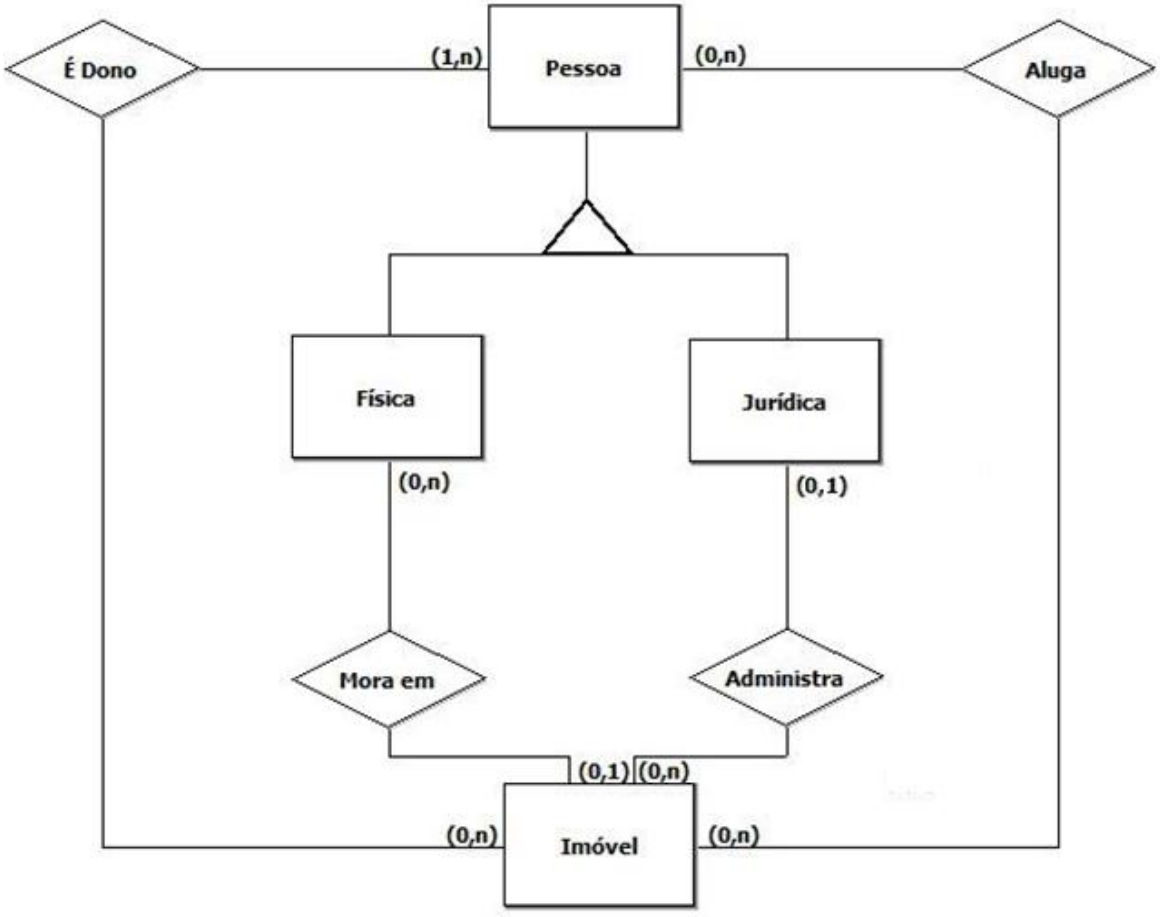
Qualquer **Pessoa** pode ser dona ou locatária de um imóvel. Não há limites para a quantidade de imóveis que uma pessoa tem, e estes imóveis podem ter um ou mais donos, assim como é possível que ele esteja sendo alugado por várias pessoas. É possível ter vários locadores.

Para futuras consultas, armazena-se em **Imóveis** um Identificador, o Valor de aluguel, o Tipo que pode ser comercial ou residencial, Telefone da Portaria e Síndico caso possua, Endereço, Nome do Prédio caso haja e o Número do Contrato assinado entre ambas as partes da negociação

Quando se aluga um imóvel deve haver um Fiador, que pode ser uma pessoa física ou jurídica. Sobre este é importante termos seu nome completo ou razão social, seu endereço, seu CPF ou CNPJ, telefone e Email.

A administração do condomínio de um **Imóvel** pode ser feita por um síndico ou por uma pessoa jurídica. Precisamos saber o valor do condomínio que pode ser NULL caso não exista a taxa e uma Descrição. A descrição do condomínio é um documento que contém informações sobre taxas cobradas pelos serviços oferecidos e pode ser NULL caso não haja um condomínio a ser pago.

## III. MODELAGEM E – R



# IV. ATRIBUTOS DAS ENTIDADES

1. **Pessoa:** É o cadastro da pessoa, sendo esta física ou jurídica, como inquilino, proprietário ou fiador.
   1.1. **Dados Bancários:** Agência, conta e banco da pessoa. Não é obrigatório. Os campos de agência e conta devem ser preenchidos apenas com números.
   1.2. **Email:** Um cadastro do email para fácil contato entre o administrador e as pessoas. É esperado receber um email que consiste na existência de um "@" entre o usuário e o provedor como no exemplo: alguém@provedor.com, alguém@provedor.com.br ou variantes.
   1.3. **Telefone:** Telefone residencial ou celular para fácil contato com a pessoa. É necessário o código de área composto de 2 dígitos seguido do número do telefone, sem separação por hífen. Máximo de 11 caracteres.
   1.4. **Id:** É um identificador único de 3 dígitos, letras ou números.
   1.5. **Nome completo:** O mesmo nome que consta em seu documento de identificação e no contrato. O tamanho máximo para um nome é de 100 caracteres e não são aceitos números ou caracteres especiais.
   1.6. **Endereço:** É um atributo composto que reuni vários itens necessários para a localização de uma pessoa. Estão inclusos CEP, Estado, Cidade, Bairro, Logradouro e Complemento. Estado, Cidade e Bairro possuem até 50 caracteres e Bairro, Logradouro e Complemento possuem até 20 caracteres, todos aceitam dígitos e letras. O CEP deve conter 8 números.

2. **Física:** Caracteriza uma pessoa física.
   2.1. **CPF:** Número de seu CPF, tamanho de onze dígitos e somente números que satisfaça seu algoritmo de verificação, exemplificado nesta URL: http://www.geradorcpf.com/algoritmo_do_cpf.htm
   2.2. **Identificação:** Podendo ser RG e UF, CNH, Passaporte, CRM, OAB, entre outros. São aceitos apenas letras e números e tem tamanho máximo de 20 caracteres.

3. **Jurídica**: Caracteriza uma pessoa jurídica.
   3.1. **CNPJ**: Número de CNPJ da empresa, que possui quatorze dígitos, somente números que satisfaça seu algoritmo de verificação exemplificado na seguinte URL: http://www.geradorcnpj.com/algoritmo_do_cnpj.htm
   3.2. **Responsável**: Nome completo, contendo até cem caracteres, do responsável pelo contrato assinado com o sistema. Este campo não aceita números ou caracteres especiais.
   3.3. **URL**: URL do site da empresa, seu modelo deve ser semelhante à http://nomedaempresa.com ou http://nomedaempresa.com.br, sempre com http:// antes do site e sempre com domínios válidos.

4. **Imóvel:** Apartamento ou Casa que esteja disponível para locação ou que esteja alugado.
   4.1. **Identificador:** Número responsável por identificar um apartamento ou casa. É único para cada um dos imóveis cadastrados. Possui 3 dígitos, letras ou números.
   4.2. **Valor:** É o valor de aluguel do imóvel. Deve estar cadastrado em real, conforme o acordado entre o administrador e o proprietário.
   4.3. **Tipo:** Distinção entre os tipos "comercial" ou "residencial", sendo reconhecidos como c ou r, respectivamente.
   4.4. **Telefone portaria:** Telefone da portaria do prédio no qual o imóvel está localizado. Pode ser NULL caso o complexo não possua uma portaria. Caso possua, deve ter um código de área de 2 dígitos antes do número e logo depois o número correspondente, sem separação por hífen. Máximo de 11 caracteres.
   4.5. **Síndico:** É um atributo composto que possui o nome do síndico responsável por aquele prédio no qual o imóvel está e um telefone de contato imediato. O nome possui até vinte caracteres e não aceita números ou caracteres especiais. O telefone contém o código de área (2 dígitos) antes do número, não há separação por hífen e há no máximo 11 caracteres. Se não houver síndico, ambos os campos podem ser NULL. Não pode haver apenas um campo NULL.
   4.6. **Endereço:** É um atributo composto que reuni vários itens necessários para a localização de uma pessoa. Estão inclusos CEP, Estado, Cidade, Bairro, Logradouro e Complemento. Estado, Cidade e Bairro possuem até 50 caracteres e Bairro, Logradouro e Complemento possuem até 20 caracteres, todos aceitam dígitos e letras. O CEP deve conter 8 números.
   4.7. **Nome do Prédio:** Nome do prédio – ou complexo – qual está localizado o imóvel. O tamanho máximo para o nome é de cinquenta caracteres.
   4.8. **Número do Contrato:** Número do contrato assinado entre ambas as partes. Deve ser exatamente igual ao que consta no documento. E seguir o modelo sete dígitos antes de um hífen e 3 dígitos depois, não aceitando caracteres especiais.

## V. ATRIBUTOS DOS RELACIONAMENTOS

- **Aluga**
  - **Fiador**: Atributo composto – e multivalorado – que representa quais são os fiadores daquela relação de aluguel. Possui:
    - **Tipo**: Campo que registra qual o tipo do fiador, como Conta Dedicada e Salário Dedicado, por exemplo. Aceita até 50 caracteres não especiais.
    - **Email:** Um cadastro do email para fácil contato entre o administrador e as pessoas. É esperado receber um email como no exemplo: alguém@provedor.com, alguém@provedor.com.br ou variantes, sempre com um "@" entre o usuário e o provedor.
    - **Telefone:** Telefone residencial, comercial ou celular para fácil contato com o fiador. É necessário o código de área antes do número em questão, composto de 2 dígitos, e logo após os números do telefone sem separação por hífen. Máximo de 11 caracteres.

        - **Nome completo ou Razão social:** O mesmo nome que consta em seu documento de identificação – não necessário para empresas - e no contrato. O tamanho máximo para um nome é de trinta e cinco caracteres não é permitido caracteres especiais.
        - **Endereço:** É um atributo composto que reuni vários itens necessários para a localização de uma pessoa. Estão inclusos CEP, Estado, Cidade, Bairro, Logradouro e Complemento. Estado, Cidade e Bairro possuem até 50 caracteres e Bairro, Logradouro e Complemento possuem até 20 caracteres, todos aceitam dígitos e letras. O CEP deve conter 8 números.
        - **CPF/CNPJ:** Número de seu CPF caso seja uma pessoa física e CNPJ caso seja uma pessoa jurídica. O CPF e CNPJ devem seguir seus respectivos algoritmos de verificação exemplificados nas URL's a seguir:
          http://www.geradorcpf.com/algoritmo_do_cpf.htm
          http://www.geradorcnpj.com/algoritmo_do_cnpj.htm
- **Administra**
    - **Valor**: Valor do condomínio a ser pago por aquele imóvel. Pode ser NULL caso não haja condomínio. Deve ser cadastrado na moeda brasileira.
    - **Descrição:** Um atributo multivalorado que possui informações sobre taxas, futuras cobranças, reembolsos e valor puro do condomínio. Pode ser NULL caso não haja condomínio.

## VI. MODELO LÓGICO

- **Pessoa** (id, nome, email, tel)
    - **Id** é uma chave primária e não pode receber *NULL.*
- **DadosB** (*idPessoa*, ag, cta, banco)
    - Onde **idPessoa** é uma chave estrangeira para o Id de Pessoa, também é uma chave primária e não pode receber *NULL.*
- **End** (*idPessoa, idImo*, cep, estado, cidade, bairro, log, compl)
    - Onde **idPessoa** e **idImo** são chaves estrangeiras para Ids de Pessoa e Imóveis (respectivamente). **idPessoa** também é uma chave primária e, portanto, não pode receber *NULL*.
- **Pfis** (*idPessoa*, cpf, ident)
    - Onde **idPessoa** é chave estrangeira para Id de Pessoa, porém também é uma chave primária e não pode receber *NULL.*
- **Pjur** (*idPessoa*, cnpj, resp, url)
    - Onde **idPessoa** é a chave estrangeira para Id de Pessoa, porém também é uma chave primária e não pode receber *NULL.*
- **Imovel** (id, nomePre, valor, tipo, numCont, telPort)
    - **Id** é chave primária e não pode receber *NULL.*
- **Administra** (*idPessoa*, *idImo*, idDesc, valor)
    - Onde **idPessoa**, **idImo** e **idDesc** são chaves estrangeiras para Ids de Pessoa (Jurídica), Imóveis e Descrição respectivamente. **IdImo** também é

chave primária, portanto não pode receber o valor *NULL*. Assim, se um imóvel não é administrado ele não constará nessa tabela.

- **Descrição** (id, taxas, futurasCobrancas, reembolsos, valCondBruto)
  - **Id** é uma chave primária e não pode receber *NULL*.
- **Sindico** (*idImo*, nome, tel)
  - Onde **idImo** é chave estrangeira para o Id de Imóveis, porém também é uma chave primária e não pode receber *NULL.*
- **Aluga** (*idPessoa*, *idImo*)
  - Onde **idPessoa** e **idImo** são chaves estrangeiras para Id de Pessoa e Id Imóvel. **idPessoa** é também chave primária e portanto não pode receber *NULL.*
- **Dono** (*idPessoa, idImo)*
  - Onde **idPessoa** e **idImo** são chaves estrangeiras para Id de Pessoa e Id Imóvel. **idPessoa** é também chave primária e portanto não pode receber *NULL.*
- **MoraNo** (*idPessoa*, *idImo*)
  - Onde **idPessoa** e **idImo** são chaves estrangeiras para Id de Pessoa e Id Imóvel. **idPessoa** é também chave primária e portanto não pode receber *NULL.*
- **Fiador** (*idPessoa*, *idImo*, tipo)
  - Onde **idPessoa** e **idImo** são chaves estrangeiras para Id de Pessoa e Id Imóvel, idImo é chave primária logo não pode receber *NULL.*
  - E **tipo** classifica qual o tipo do fiador.

# VII. RESTRIÇÕES

## Semântica (Business Rules)

- Todo aluguel precisa ter um fiador.

- O somatório das colunas da tabela Descrição tem que ser igual ao valor da coluna “Valor” da tabela Administra

## VIII. MUDANÇAS NO PROJETO

Ao revisarmos o projeto antigo verificamos que haviam alternativas melhores para alcançarmos o objetivo proposto com este trabalho.

Para melhorar as consultas e ter um resultado correto e optimizado, criamos a tabela Descrição a partir do atributo composto “Descrição” da entidade Imóvel.

Além disso, em nosso trabalho anterior a tabela feita para Fiador, que era um atributo composto do relacionamento “Aluga”, possuía colunas quase idênticas a tabela de Pessoa e suas especializações, criando uma redundância no projeto. Resolvemos esta questão fazendo com que a tabela Fiador possuísse uma chave estrangeira para Pessoa, assim não sendo necessário repetir todas colunas.

## IX. TÉCNICAS DE MAPEAMENTO RELACIONAL E LÓGICO

Durante o processo de construção do modelo E-R, **Administra** virou uma tabela separada pela de sua interação com ambas as entidades (e tabelas) **Imóvel** e **Pessoa**. Como a relação é de 1:n e a entidade de cardinalidade máxima 1(Pessoa Jurídica) não precisa obrigatoriamente aparecer em todos os casos, foi criada uma tabela adjacente. Deste modo a relação **Mora Em**, que é de 1:n, também ganhou uma tabela própria chamada de **MoraNo**.

Em **Pessoa**, por Dados Bancários ser um atributo composto foi criada a tabela **Dadosb** em separado, melhorando a performance do sistema ao fazer consultas que envolvam este atributo. Como o Endereço em **Pessoa** e Síndico em **Imóvel** também são atributos compostos, o mesmo caminho foi seguido, criando a tabela **End** e **Sindico**.

**Pfis** e **Pjur** são especializações de **Pessoa**, assim foi criada uma tabela para cada, minimizando as junções, o número de chaves e possibilitando colunas adicionais.

Como **Aluga** e **É Dono** são relacionamentos de n:n, cada um ganhou uma tabela própria, chamadas de **Aluga** e **Dono** respectivamente.

Em Aluga, há o atributo composto **Fiador**, que ganhou sua própria tabela de mesmo nome.

# X. AVALIAÇÃO DA QUALIDADE DO PROJETO LÓGICO

Tabelas:
**Pessoa** (id, nome, email, tel);
**Imovel** (id, nomePre, valor, tipo, numCont, telPort);
**Administra** (*idPessoa*, *idImo*, valor, idDesc).

a) Dependência funcional

Dependência funcional é uma restrição entre dois subconjuntos de atributos (A e B), sendo denotada por A -> B, onde A determina funcionalmente B.

- Tabela Pessoa: {id} -> {nome, email, tel}
- Tabela Imovel: {id} -> {nomePre, valor, tipo, numCont, telPort}
- Tabela Administra: {idImo} -> {idPessoa, valor, idDesc}

b) Formas normais:

- Primeira Forma Normal (1FN): Uma relação R está em 1FN se todos os atributos são indivisíveis.

As três tabelas estão na primeira forma normal. Nenhum dos atributos das tabelas pode ser multivalorado. Por exemplo, apesar de normalmente as pessoas terem mais de um e-mail ou telefone, o banco de dados só aceita um valor para os campos **email**, **tel** e **telPort**.

Para que a tabela Pessoa aceite mais que um telefone e continue na primeira forma normal seria necessário tirar o atributo **tel** (Pessoa (id, nome, email)) e criar uma tabela TelefonePessoa (*idPessoa*, tel), onde **idPessoa** iria ser chave estrangeira para a tabela Pessoa e o conjunto **idPessoa** e **tel** seria chave primaria.

- Segunda Forma Normal (2FN): Uma relação R está em 2FN se estiver em 1FN e nenhum atributo não-primo (que não faz parte da chave) depender funcionalmente de uma parte da chave.

As três tabelas estão na segunda forma normal. As chaves primarias das tabelas são formadas por apenas um atributo, logo não tem como os atributos não-primos dependerem funcionalmente apenas de uma parte da chave.

A tabela Administra não estaria na segunda forma normal caso idPessoa fosse chave primaria também. Ou seja, Administra (*idPessoa*, *idImo*, valor, idDesc). Neste caso, o atributo **valor** não dependeria de **idPessoa**, ou seja, não dependeria de parte da chave.

- Terceira Forma Normal (3FN): Uma relação R está na 3FN se estiver também na 2FN e nenhum atributo não-chave depender de outro atributo não-chave.

As três tabelas estão na segunda forma normal. Em nenhum dos três casos há um atributo não-chave que dependa de outro atributo não-chave.

Caso a tabela Imovel fosse: Imovel (id, nomePre, valor, desc, tipo, numCont, telPort), onde **desc** fosse um desconto calculado a partir do atributo valor, a tabela não estaria na terceira forma normal. Neste caso **desc** dependeria de valor, que não é chave.

- Forma Normal de Boyce-Codd (BCNF): Uma tabela está na BCNF se e somente se estiver na 3FN e todo determinante for chave candidata.

Como toda chave primaria é uma chave candidata, então nestas tabelas todo determinante é chave candidata. Portanto as tabelas estão na BCNF.

## XI. CONSULTAS EM ÁLGEBRA RELACIONAL

➔ Produto cartesiano:

**1)** Qual nome das pessoas físicas cadastradas no banco de dados?

AUX1 <- π (Pfis) IdPessoa

AUX2 <- π (Pessoa) Id, nome

AUX3 <- AUX1 X AUX2

AUX4 <- σ (AUX3) Id=IdPessoa

Resp <- π (AUX4) nome

➔ União:

**2)** A identidade das pessoas que são donas ou fiadoras de algum imóvel.

AUX1 <- π (Fiador) IdPessoa

AUX2 <- π (Dono) IdPessoa

Resp <- AUX1 U AUX2

➔ Interseção:

**3)** A identidade das pessoas jurídicas que alugam algum imóvel e administra qualquer outro(s):

AUX1 <- π (Pjur) IdPessoa

AUX2 <- π (Aluga) IdPessoa

AUX3 <- π (Administra) IdPessoa

AUX4 <- AUX2 ∩ AUX3

Resp <- AUX4 – AUX1

➔ Junção externa à esquerda:

**4)** Listar todas as pessoas (nome e identidade) e os imóveis (Id do imóvel) que ela é dona. Caso ela não seja dona de nenhum imóvel, atribuir NULL:

AUX1 <- π (Pessoa) Id, nome

AUX2 <- π (Dono) IdPessoa, IdImo

AUX3 <- δ (AUX2) Id -> IdPessoa

Resp <- AUX1 =|X| AUX3

➔ Divisão:

**5)** Listar as todas pessoas (identidade) que são donas de todos imóveis em Ipanema e Copacabana:

AUX1 <- π (End) bairro='Ipanema' v bairro='Copacabana'

AUX2 <- π (End) IdPessoa, bairro

Resp <- AUX2 ÷ AUX1

➔ Junção Natural:

**6)** Qual é o CPF/CNPJ da pessoa de identidade 115?

TEMP1 ← pFis |><| <id = 115> pJur

RESP ← π <cpf, cnpj> (TEMP1)

➔ Função de agregação:

**7)** Para cada dono de imóveis, qual é o valor total gasto com compras de imóveis.

TEMP1 ← π(Pessoa) id, nome x π(Dono) idPessoa, idImo

TEMP2 ← σ (TEMP1) <id = idPessoa>

TEMP3 <- δ (TEMP2) IdImo -> IdImoAux

TEMP4 ← π(TEMP3) nome, idImoAux x π(Imovel) idImo, valor

TEMP5 ← σ (TEMP5) <idImoAux = idImo>

TEMP6 ← nome G sum(valor) (TEMP5)

RESP ← π <nome, valor> (TEMP6)

## XII. Script DLL

**1)** Create Table administra

```
(
idpessoa varchar(3),
idimo varchar(3),
valor numeric(12,2),
constraint pk_adm PRIMARY KEY (idimo),
constraint fk_adm_pjur FOREIGN KEY (idpessoa) REFERENCES pjur(idpessoa),
constraint fk_adm_imo FOREIGN KEY (idimo) REFERENCES imovel(id)
)
```

alter table administra add column idDesc varchar(3)

alter table administra add constraint fk_adm_descr foreign key (idDesc) references descricao(id)

**2)** Create Table aluga

(

idpessoa varchar(3),
idimo varchar(3),

constraint pk_aluga PRIMARY KEY (idpessoa, idimo),

constraint fk_aluga_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id),

constraint fk_imovel FOREIGN KEY(idimo) REFERENCES imovel(id)

)

**3)** Create Table dadosb

(

idpessoa varchar(3),
ag numeric(10,0),
conta numeric(10,0),
banco varchar(20),

constraint pk_dadosb PRIMARY KEY(idpessoa),

constraint fk_dadosb_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id)

)

**4)** Create Table dono

(

idpessoa varchar(3),

idimo varchar(3),

constraint pk_dono PRIMARY KEY(idpessoa, idimo),

constraint fk_dono_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id),

constraint fk_dono_imovel FOREIGN KEY(idimo) REFERENCES imovel(id)

)

**5)** Create Table endereco

```
(
idpessoa varchar(3),
idimo varchar(3),
cep numeric(8,0),
estado varchar(50),
cidade varchar(50),
bairro varchar(50),
log varchar(20),
compl varchar(2),
constraint pk_imo PRIMARY KEY(idpessoa) REFERENCES pessoa(id),
constraint fk_end_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id),
constraint fk_end_imo FOREIGN KEY(idimo) REFERENCES imovel(id)
)
```

**6)** Create Table Fiador

```
(
idpessoa varchar(3),
idimo varchar(3) NOT NULL,
tipo varchar(5),
constraint pk_fiador PRIMARY KEY (idimo),
constraint fk_fiador_pessoa FOREIGN KEY (idpessoa) REFERENCES pessoa(id),
constraint fk_fiador_imovel FOREIGN KEY (idimo) REFERENCES imovel(id)
)
```

**7)** Create Table morano

```
(
idpessoa varchar(3) ,
idimo varchar(3),
constraint pk_mora PRIMARY KEY(idpessoa, idimo),
constraint fk_mora_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id),
constraint fk_mora_imovel FOREIGN KEY(idimo) REFERENCES imovel(id)
)
```

**8)** Create Table pfis

```
(
idpessoa varchar(3),
cpf numeric(11,0),
ident varchar(20),
constraint pk_pfis PRIMARY KEY (idpessoa),
constraint fk_pessoa_pfis FOREIGN KEY(idpessoa) REFERENCES pessoa(id),
)
```

**9)** Create Table Sindico

```
(
idimo varchar(3),
nome varchar(100),
telefone numeric(11,0),
constraint pk_sindico PRIMARY KEY (idimo),
constraint fk_sindico_imo FOREIGN KEY (idimo) REFERENCES imovel(id),
)
```

**10)** create table pjur

```
(
idpessoa varchar(3),
cnpj numeric(14),
resp varchar(100),
url varchar(100),
constraint pk_pjur PRIMARY KEY (idpessoa),
constraint fk_pjur_pessoa FOREIGN KEY(idpessoa) REFERENCES pessoa(id),
constraint uni_cnpj unique (cnpj)
)
```

**11)** create table pessoa

```
(
id varchar(3),
nome varchar(100),
email varchar(25),
tel numeric(11,0),
constraint pk_pessoa primary key (id),
constraint uni_email uni_pessoa unique(email)
)
```

**12)** create table imovel

```
(
id varchar(3),
nomepre varchar(5),
valor numeric(12,2),
tipo varchar(1),
numcontr numeric(10,0),
telport numeric(11,0),
constraint pk_imovel primary key (id),
constraint check_tipo check (tipo in ('R','C'))
)
```

**13)** create table descricao

(

id varchar(3),
taxas numeric(12,2),
futurasCobrancas numeric(12,2),
reembolsos numeric(12,2),
valCondbruto numeric(12,2),

constraint pk_descr primary key (id)

)

## XIII. CONSULTA SQL

➔ Simples

Para cada pessoa jurídica, liste sua id e seu CNPJ:

```
SELECT IdPessoa, CNPJ
FROM Pjur
```

Listar os síndicos (nome) que não tem telefone cadastrado:

```
SELECT nome
FROM Sindico
WHERE tel is NULL
```

➔ Distinct

Listar os diferentes valores de condomínio cobrados pelas administradoras:

```
SELECT DISTINCT valor
FROM Administra
```

➔ Between

Listar os imóveis (Id) com valor de condomínio entre 200 e 500 reais:

```
SELECT IdImo
FROM Administra
WHERE valor BETWEEN 200 AND 500
```

➔ Or, not in

Listar as pessoas (Id) que tem conta no banco Itaú ou Bradesco e não são donos de nenhum imóvel:

```
SELECT IdPessoa
FROM DadosB
WHERE banco = 'Itaú'
OR Banco = 'Bradesco'
AND idPessoa NOT IN(
        SELECT idPessoa
        FROM dono)
```

➔ Order by, and

Listar todos os imóveis, id e numCont, separando-os pelos seus donos.

```
SELECT A.nome, B.idPessoa, C.numContr
FROM Pessoa as A, Dono as B, Imovel as C
Where A.id = B.idPessoa
AND C.id = B.idImo
ORDER BY A.nome
```

➔ In, distinct

Listar as pessoas (Id) que são donas de algum apartamento no Leblon:

```
SELECT DISTINCT IdPessoa
FROM Dono
WHERE IdImo in
                (SELECT IdImo
                 FROM Endereco
                 WHERE bairro = 'Leblon' AND
                cidade = 'Rio de Janeiro')
```

➔ Exists

Listar a identidade das pessoas que moram no apartamento que são donas:

```
SELECT IdPessoa
FROM MoraNo
WHERE exists (
```

```
SELECT *
FROM Dono
WHERE Dono.IdPessoa = MoraNo.IdPessoa )
```

➔ Like ‘%_ _ _%’, order by, and

Listar o email das pessoas físicas que são do gmail:

```
SELECT P.email
FROM Pessoa as P, Pfis as F
WHERE P.id = F.IdPessoa
And P.email like ‘%@gmail.com%’
```

➔ Count

Quantos são os fiadores do tipo conta direta?

```
SELECT count (IdPessoa)
FROM Fiador
WHERE tipo = ‘Conta direta’
```

## Consultas em SQL Avançadas

➔ Insert:

Adicionar uma nova pessoa.

```
INSERT INTO Pessoa (id, nome, email, tel)
VALUES (‘e12’, ‘Helena’, ‘helena@aluno.puc-rio.com.br’, (21)2437-1254)
```

➔ Update:

Atualizar o telefone da Helena.

```
UPDATE Pessoa
SET tel = '(21)99587-1264'
WHERE id = 'e12'
```

➔ Delete:

Deletar a pessoa Helena do banco de dados.

```
DELETE from Pessoa
WHERE id = ‘e12’
```

➔ Group by, funções de agregação:

Listar o valor total gasto em imóveis por todos os donos de imóveis.

```
SELECT A.nome, sum(C.valor)
FROM Pessoa as A, Dono as B, Imovel as C
Where A.id = B.idPessoa
AND C.id = B.idImo
GROUP BY A.nome
```

➔ Having:

Listar nome dos síndicos dos prédios que possuem mais de dois donos de apartamentos cadastrados no banco de dado.

```
SELECT A.nome, count(B.idPessoa) as number
FROM Sindico as A, Dono as B
Where A.idImo = B.idImo
group by A.nome
HAVING count(B.idPessoa) > 2
```

➔ Consulta dentro do from:

Selecione as pessoas e o nome do prédio que elas possuem imóveis alugados e possuem conta no Banco do Brasil.

```
SELECT aluga.nome, aluga.nomePre
FROM (SELECT A.nome, A.id, C.nomePre
        FROM Pessoa as A, Aluga as B, Imovel as C
        WHERE A.id = B.idPessoa
        AND b.idImo = C.id) as aluga, dadosB
WHERE aluga.id = dadosB.idpessoa
AND dadosB.banco = 'Banco do Brasil'
```

## XIV. Índices

Índice secundário do atributo Idimo na tabela Endereco

```
create index as imo_index on endereco(idimo)
```

Índice secundário do atributo banco na tabela Dadosb

```
create index as banco_index on dadosb(banco)
```

## XV. Visões

**1)** Visão com nome da pessoa, banco, agencia e conta:

```
create view nome_dadosb as (
select nome, banco, ag, conta
from pessoa, dadosb
where id=idpessoa )
```

**2)** Visão com nome e email das pessoas que moram na cidade do Rio de Janeiro:

```
create view mora_rj as (
select nome, email
from endereco as E, (select nome, email, idimo
                     from morano, pessoa
                     where id=idpessoa) as P
where P.idimo=E.idimo and E.cidade='Rio de Janeiro'
)
```

**3)** Visão com os imóveis e seus donos:

```
create view pessoas_dono as(
select A.nome, B.id, b.nomePre, b.valor, b.tipo, b.numContr, b.telPort
from pessoa as A, imovel as B, dono as C
where A.id = C.idPessoa
and B.id = C.idImo)
```

**4)** Visão com imóveis e quem os aluga:

```
create view pessoas_aluga as
(
select A.nome, B.id, b.nomePre, b.valor, b.tipo, b.numContr, b.telPort
from pessoa as A, imovel as B, aluga as C
where A.id = C.idPessoa
and B.id = C.idImo
)
```

**5)** Visão com imóveis e seus moradores:

```
create view pessoas_morano as
(
select A.nome, B.id, b.nomePre, b.valor, b.tipo, b.numContr, b.telPort
from pessoa as A, imovel as B, morano as C
where A.id = C.idPessoa
and B.id = C.idImo
)
```

## XVI. Funções

**1)** Conta os imóveis na cidade:

```
create function conta_imo(cidade_imo varchar(50))
returns integer as $$
declare total integer
begin
select count(*) into total
from endereco
where cidade=cidade_imo;
return total;
end;
$$ LANGUAGE plpgsql;
```

**2)** Conta quantos imóveis uma pessoa jurídica administra:

```
create function conta_adm(id_pjur varchar(3))
returns integer as $$
declare qtd integer;
begin
select count(*) into qtd
from administra
return qtd;
end;
$$ LANGUAGE plpgsql;
```

**3)** Lista o id e o valor dos imóveis com valores maiores do que o indicado:

```
create function tabela_valores(cid_imo varchar(50), valor_imo numeric(12,2))
returns SETOF record as $$
declare result record;
begin
for result in select id, valor
from imovel, endereco
where id=idimo and cidade=cid_imo and valor > valor_imo
loop
return next result;
end loop;
end;
$$ LANGUAGE plpgsql;
EXEMPLO DE CHAMADA: select *
                    from tabela_valores('Rio de Janeiro','100000') AS (id
varchar(3),valor numeric(12,2))
```

## XVII. Triggers

**1)** Trigger que verifica se um imóvel tem fiador antes de lançá-lo na tabela aluga.

```
create function verifica_fiador()
returns trigger as $ver_fiador$;
declare qtd integer;
begin
select count(*) into qtd
from fiador as F
where F.idimo=NEW.idimo;
if(qtd=0) then
return NULL;
end if;
return NEW;
end;
$ver_fiador$ LANGUAGE plpgsql;

create trigger ver_fiador before insert or update on aluga for each row execute
procedure verifica_fiador();
```

**2)** Verifica se uma pessoa jurídica tem seus dados bancários cadastrados.

```
create function verifica_dados()
returns trigger as $ver_dados$;
declare qtd integer;
begin
select count(*) into qtd
from dadosb
where idpessoa=NEW.idpessoa;
if(qtd=0) then
return NULL;
```

```
end if;

return NEW;

end;

$ver_dados$ LANGUAGE plpgsql;


create trigger ver_dados before insert or update on pjur for each row execute
procedure verifica_dados();
```