RentalData

Este projeto é uma implementação baseada em PostgreSQL de um modelo de banco de dados para uma plataforma de acomodação e reservas.

Ele contém scripts SQL para criar o esquema, popular com dados fictícios e executar consultas analíticas. Os scripts são executados dentro de um contêiner Docker PostgreSQL.

Índice

Estrutura

```
.
├── atividade3_3-1.sql            # cria o banco de dados e tabelas (sem restrições de integridade)
├── atividades-consultas/         # contém todos os scripts relacionados às consultas
│   ├── atividade3_3-2.sql        # carrega dados fictícios (INSERTs)
│   ├── atividade3_3-3.sql        # consultas básicas nos dados de propriedade
│   ├── atividade3_3-4.sql        # análise de reservas
│   └── atividade3_3-5.sql        # consultas avançadas e análises
├── consultas.sql                 # script mestre que executa todas as consultas
└── saida_consultas.txt           # saída de todas as consultas (para inclusão no relatório)
```

Requisitos

- Docker
- Imagem Docker do PostgreSQL (baixada automaticamente ao iniciar o contêiner)

Instalação

1. Inicie um contêiner PostgreSQL (caso não esteja em execução):

```
docker run -d \
  --name pgdev \
  -e POSTGRES_USER=lfelipediniz \
  -e POSTGRES_PASSWORD=1234 \
  -e POSTGRES_DB=atividade3_bd \
  -p 5434:5432 \
  postgres:latest
```

2. Copie todos os arquivos para dentro do contêiner:

```
docker cp . pgdev:/
```

Como Executar

Passo 1: Destruir e recriar o banco de dados + criar tabelas

```
docker exec -it pgdev \
  psql -U lfelipediniz -d postgres \
  -f /atividade3_3-1.sql
```

Passo 2: Executar todos os inserts e consultas, salvando a saída

```
docker exec -i pgdev \
  psql -U lfelipediniz -d atividade3_bd \
  -f /consultas.sql > saida_consultas.txt
```

Após isso, o arquivo `saida_consultas.txt` conterá a saída completa de:

- Inserção de dados
- Consultas básicas e avançadas

Observações Importantes

- **Se você executar `consultas.sql` mais de uma vez**, os mesmos dados serão inseridos várias vezes.

- Isso ocorre porque **não implementamos restrições de integridade (ex.: PRIMARY KEY, UNIQUE)** nesta versão.
- Esse comportamento foi intencional, seguindo as instruções de pular as restrições na fase inicial de modelagem.

Você pode limpar manualmente a base reexecutando:

```
docker exec -it pgdev \
  psql -U lfelipediniz -d postgres \
  -f /atividade3_3-1.sql
```

Explicação dos Comandos SQL do Passo 3.1

1. Inicialização do Banco de Dados

- `DROP DATABASE IF EXISTS atividade3_bd;`
  Garante que qualquer banco chamado `atividade3_bd` seja removido primeiro, evitando erros de "já existe" ao reexecutar o script.
- `CREATE DATABASE atividade3_bd;`
  Cria um novo banco de dados PostgreSQL chamado `atividade3_bd` .
- `\connect atividade3_bd`
  Muda a sessão psql para o banco recém-criado para que todos os `CREATE TABLE` seguintes sejam aplicados nele.

2. Limpeza de Tabelas

- `DROP TABLE IF EXISTS <table_name> CASCADE;`
  Para cada tabela, esse comando exclui-a com segurança se existir. A opção `CASCADE` também remove objetos dependentes (como chaves estrangeiras ou views) para evitar erros de drop.

3. Definições das Tabelas Principais

Cada `CREATE TABLE` define as colunas e tipos de dados sem impor chaves:

- **Tipos de string**
  - `VARCHAR(n)` : strings de tamanho variável até `n` caracteres (ex.: `VARCHAR(100)` para nomes).

    - `TEXT` : texto de comprimento ilimitado, usado quando o tamanho é imprevisível (ex.: endereços e mensagens).

- **Tipos de data/hora**
    - `DATE` , `TIME` , `TIMESTAMP` armazenam datas, horários e timestamps completos, respectivamente.

- **Tipos numéricos**
    - `INTEGER` : números inteiros.
    - `NUMERIC(precision, scale)` : decimais exatos (ex.: valores monetários com duas casas decimais).

- **Outros tipos**
    - `CHAR(1)` : campos de um caractere fixo (ex.: código de gênero).
    - `BOOLEAN` : valores verdadeiro/falso.

Nenhuma chave primária ou estrangeira é especificada aqui, mantendo o foco apenas na estrutura das tabelas.

4. Auto-incremento com `SERIAL`

- Colunas declaradas como `SERIAL` (ex.: `id_propriedade SERIAL`)
  Criam automaticamente uma coluna inteira que obtém seus valores de uma sequência oculta, facilitando a geração de identificadores únicos sem gerenciar manualmente as sequências.

5. Tabelas Associativas para Relacionamentos N:M

- Tabelas como `propriedade_comodidade` e `propriedade_regra`
  Representam relacionamentos N:M listando pares de colunas referenciadoras. Permanecem sem restrições nesta versão, apenas armazenando as associações.

Explicação do Script de Carga de Dados do Passo 3.2

Esta seção percorre cada parte lógica do script `atividade3_3-2.sql` , que popula o banco `atividade3_bd` com dados de exemplo. Não há repetições das definições de tabelas — apenas a justificativa de cada `INSERT` .

1) Inserção em `hospede`

- **Objetivo:** Carregar 15 registros de hóspedes para simular usuários reais.
- **Pontos-chave:**
  - Colunas listadas na mesma ordem do `CREATE TABLE` do Passo 3.1.
  - Literais de string em aspas correspondem a cada coluna `VARCHAR` ou `TEXT` .
  - `NULL` usado onde informações opcionais (telefone ou email) não são fornecidas.
  - Datas seguem o formato `YYYY-MM-DD` para o tipo `DATE` .
  - Senhas são armazenadas em texto puro para testes (em sistemas reais, seriam hashadas).

2) Inserção em `locador`

- **Objetivo:** Popular 15 registros de proprietários de imóveis.
- **Pontos-chave:**
  - Estrutura paralela a `hospede` : mesmas colunas, mas para anfitriões em vez de hóspedes.
  - Demonstra uso de `NULL` e strings em caixa mista.
  - Garante valores de CPF distintos suficientes para junções posteriores.

3) Inserção em `localizacao`

- **Objetivo:** Definir 15 localidades geográficas distintas.
- **Pontos-chave:**
  - Omite a coluna `id_localizacao` porque é `SERIAL` ; o PostgreSQL gera automaticamente.
  - Mistura cidades brasileiras em vários estados, ilustrando como armazenar metadados de endereço.
  - Códigos postais (`cep`) e nomes de bairro completam os dados de localização.

4) Inserção em `comodidade`

- **Objetivo:** Popular 15 nomes de comodidades para imóveis.
- **Pontos-chave:**
  - Apenas uma coluna (`nome`) é listada no `INSERT` .

- Vários valores em uma única instrução melhoram a performance de carga.
- Mostra como inserir em massa dados de lookup simples.

5) Inserção em `regra`

- **Objetivo:** Estabelecer 10 regras distintas de casa, com um indicador Booleano.
- **Pontos-chave:**
  - `tipo` descreve a regra (ex.: “Fumar” ou “Pets”).
  - `permitido` usa `TRUE` / `FALSE` para indicar aplicação da regra.
  - Essa tabela de lookup guia a lógica de negócio nas validações de reserva.

6) Inserção em `propriedade`

- **Objetivo:** Criar 15 listagens de imóveis vinculadas a anfitriões e localidades.
- **Pontos-chave:**
  - Omite a coluna `id_propriedade` porque é `SERIAL` ; PostgreSQL gere automaticamente.
  - Horários de `checkin` / `checkout` seguem o formato `HH:MM` para o tipo `TIME` .
  - Referências a `cpf_locador` e `id_localizacao` mostram como chaves estrangeiras conectariam as tabelas.

7) Inserção em `quarto`

- **Objetivo:** Definir quartos individuais dentro de propriedades (entidade dependente).
- **Pontos-chave:**
  - Identificador composto: (`id_propriedade` , `numero`) distingue cada quarto.
  - Booleanos (`banheiroprivativo`) usam `TRUE` / `FALSE` .
  - Ilustra relacionamentos 1:N sem restrições explícitas.

8) Inserção em `reserva`

- **Objetivo:** Carregar 15 registros de reservas para simular bookings.
- **Pontos-chave:**

- Datas (`datareserva` , `checkin` , `checkout`) em `YYYY-MM-DD` .
- Campos monetários (`imposto` , `precototal` , `precocomtaxa`) usam `NUMERIC(10,2)` .
- `status` é um indicador textual (ex.: `confirmada` , `pendente` , `cancelada`).
- Referências a `id_propriedade` e `cpf_hospede` vinculam cada reserva ao hóspede e ao imóvel.

Cada `INSERT` segue o padrão: lista as colunas alvo, depois fornece os valores correspondentes para cada linha. Isso garante integridade e clareza ao revisar ou estender o dataset.

Explicação das Consultas SQL do Passo 3.3

Esta seção demonstra três consultas básicas nas tabelas populadas: recuperar todas as linhas, agregar por tipo de propriedade e contar por cidade via join.

Resultado 3.3

```
lfelipediniz@caue-linux:~/Documents/Github/sql$ docker cp atividade3_3-3.sql pgdev:/atividade3_3-3.sql
docker exec -it pgdev \
  psql -U lfelipediniz -d atividade3_bd \
  -f /atividade3_3-3.sql
Successfully copied 2.56kB to pgdev:/atividade3_3-3.sql
 id_propriedade |        nome         |        tipo         |         endereco          | nbanheiros | nquartos | preco_noite | noite_min | noite_max | maxhospedes | checkin  | checkout | taxalimpeza | cpf_locador | id_localizacao
----------------+---------------------+---------------------+---------------------------+------------+----------+-------------+-----------+-----------+-------------+----------+----------+-------------+-------------+---------------
              1 | Casa Feliz          | casa_inteira        | Rua Alegre, 10            |          2 |        3 |      150.00 |         1 |         7 |           6 | 14:00:00 | 12:00:00 |       50.00 | 17171717171 |             1
              2 | Chalé Encantado     | casa_inteira        | Chalé Encantado, s/n      |          1 |        2 |      200.00 |         2 |         5 |           4 | 15:00:00 | 11:00:00 |       60.00 | 27272727272 |             2
              3 | Apartamento Solar   | quarto_individual   | Av. Solar, 205            |          1 |        1 |       80.00 |         1 |        10 |           2 | 13:00:00 | 10:00:00 |       30.00 | 18181818181 |             3
              4 | Loft Urbano         | casa_inteira        | Rua das Laranjeiras, 7    |          1 |        1 |      120.00 |         1 |         3 |           2 | 14:00:00 | 11:00:00 |       40.00 | 19191919191 |             4
              5 | Pousada do Lago     | casa_inteira        | Estrada Real, 1           |          3 |        5 |      300.00 |         2 |        10 |          10 | 16:00:00 | 10:00:00 |      100.00 | 14141414141 |             5
              6 | Studio Compacto     | quarto_individual   | Rua Sol Nascente, 12      |          1 |        1 |       90.00 |         1 |         5 |           2 | 13:00:00 | 12:00:00 |       35.00 | 33333333333 |             6
              7 | Sítio Verde         | casa_inteira        | Sítio Verde, Km 10        |          2 |        4 |      250.00 |         2 |         8 |           8 | 15:00:00 | 11:00:00 |       80.00 | 22232323232 |             7
              8 | Cobertura Panorâmica | casa_inteira       | Av. Panorâmica, 123       |          2 |        3 |      400.00 |         3 |         7 |           6 | 14:00:00 | 12:00:00 |      120.00 | 30303030303 |             8
              9 | Chácara Bom Jesus   | casa_inteira        | Chácara Bom Jesus, 5      |          2 |        6 |      350.00 |         2 |        10 |          12 | 15:00:00 | 10:00:00 |       90.00 | 28282828282 |             9
             10 | Apartamento Estilo  | quarto_compartilhado | Rua Cristal, 50          |          1 |        2 |       60.00 |         1 |        15 |           4 | 12:00:00 | 11:00:00 |       25.00 | 15151515151 |             1
             11 | Casa da Serra       | casa_inteira        | Rua da Serra, 100         |          2 |        4 |      180.00 |         1 |         7 |           5 | 14:00:00 | 12:00:00 |       55.00 | 66666666666 |             2
             12 | Flat Central        | quarto_individual   | Av. Central, 10           |          1 |        1 |      110.00 |         1 |        30 |           2 | 13:00:00 | 12:00:00 |       45.00 | 25252525252 |             3
             13 | Pipoca Hostel       | quarto_compartilhado | Praça Central, 5         |          1 |        4 |       40.00 |         1 |        20 |           8 | 15:00:00 | 10:00:00 |       15.00 | 44444444444 |             4
             14 | Casa do Sol         | casa_inteira        | Rua do Sol, 1             |          2 |        3 |      220.00 |         2 |         6 |           5 | 14:00:00 | 11:00:00 |       65.00 | 17171717171 |             5
             15 | Cabana Refúgio      | casa_inteira        | Estrada das Pedras, km 12 |          1 |        1 |      170.00 |         1 |         3 |           2 | 14:00:00 | 11:00:00 |       40.00 | 31313131313 |            10
(15 rows)

        tipo         | total_por_tipo
---------------------+----------------
 quarto_compartilhado |              2
 casa_inteira         |             10
 quarto_individual    |              3
(3 rows)

    cidade     | total_por_cidade
---------------+------------------
 São Carlos    |                2
 Belo Horizonte |               1
 Rio Claro     |                2
 Porto Alegre  |                1
 Ibitinga      |                2
 Florianópolis |                1
 Curitiba      |                1
 Fortaleza     |                1
 Campinas      |                2
 Araraquara    |                2
(10 rows)
```

1) Recuperar Todos os Registros de `propriedade`

```sql
SELECT *
FROM propriedade;
```

- **`SELECT *`**
  Busca todas as colunas da tabela `propriedade` .
- **`FROM propriedade`**
  Indica a tabela de origem.

- **Caso de uso:**
  Inspecionar rapidamente todas as propriedades, incluindo IDs gerados automaticamente, nomes, tipos, localidades, preços e referências de anfitrião — ideal para dump completo de dados ou debugging.

2) Contar Propriedades por Tipo

```sql
SELECT tipo,
       COUNT(*) AS total_por_tipo
FROM propriedade
GROUP BY tipo;
```

- **`SELECT tipo`**
  Seleciona a coluna `tipo`, que categoriza o imóvel (ex.: `casa_inteira`, `quarto_individual`).
- **`COUNT(*)`**
  Conta todas as linhas em cada grupo.
- **`AS total_por_tipo`**
  Renomeia a coluna de contagem para clareza.
- **`GROUP BY tipo`**
  Agrupa as linhas por valor de `tipo`.
- **Caso de uso:**
  Compreender a distribuição de anúncios por categoria, útil para análises e relatórios.

3) Contar Propriedades por Cidade via JOIN

```sql
SELECT l.cidade,
       COUNT(*) AS total_por_cidade
FROM propriedade p
JOIN localizacao l
  ON p.id_localizacao = l.id_localizacao
GROUP BY l.cidade;
```

- **`JOIN localizacao l ON p.id_localizacao = l.id_localizacao`**
  Liga cada propriedade (`p`) à sua localização (`l`) correspondendo `id_localizacao`.
- **`SELECT l.cidade`**
  Recupera o nome da cidade da tabela `localizacao`.

- `COUNT(*) AS total_por_cidade`
  Conta o número de propriedades em cada cidade, renomeando para legibilidade.
- `GROUP BY l.cidade`
  Agrupa resultados por cidade para produzir uma linha por localidade.
- **Caso de uso:**
  Analisar a distribuição geográfica de imóveis, essencial para insights de mercado e planejamento estratégico.

Explicação da Consulta SQL do Passo 3.4

Esta consulta recupera todas as reservas confirmadas com datas de check-in em ou após 24 de abril de 2025, e faz joins com tabelas relacionadas para enriquecer cada registro com nomes de anfitrião e hóspede, duração da estadia e preço por noite.

Resultado 3.4

```
lfelipediniz@caue-linux:~/Documents/Github/sql$ docker cp atividade3_3-4.sql pgdev:/atividade3_3-4.sql
docker exec -it pgdev \
  psql -U lfelipediniz -d atividade3_bd \
  -f /atividade3_3-4.sql
Successfully copied 2.56kB to pgdev:/atividade3_3-4.sql
 id_reserva | id_propriedade | cpf_hospede | cpf_locador | total_dias_locado | nome_proprietario | nome_hospede | preco_noite
------------+----------------+-------------+-------------+-------------------+-------------------+--------------+-------------
         15 |              1 | 16161616161 | 17171717171 |                 2 | Zuleica           | Fernando     |      150.00
          1 |              1 | 11111111111 | 17171717171 |                 5 | Zuleica           | Zé           |      150.00
          3 |              3 | 33333333333 | 18181818181 |                 4 | Carlos            | Pedro        |       80.00
          6 |              7 | 66666666666 | 22232323232 |                 4 | Mateus            | Joana        |      250.00
          7 |              8 | 77777777777 | 30303030303 |                 3 | Leandro           | Rafael       |      400.00
         11 |             12 | 12121212121 | 25252525252 |                 4 | Larissa           | Tiago        |      110.00
         14 |             15 | 15151515151 | 31313131313 |                 5 | Juliana           | Mariana      |      170.00
(7 rows)

lfelipediniz@caue-linux:~/Documents/Github/sql$
```

1) Critérios de Filtragem

```sql
  WHERE r.status = 'confirmada'
    AND r.checkin >= '2025-04-24';
```

- `r.status = 'confirmada'`
  Seleciona apenas reservas cujo status seja “confirmada”.
- `r.checkin >= '2025-04-24'`
  Garante que a data de check-in seja em ou após 24/04/2025.
- **Formato de data:**
  Usa o formato ISO `YYYY-MM-DD` esperado pelo PostgreSQL para comparações de `DATE`.

2) Colunas Principais Selecionadas

```sql
r.id_reserva,
p.id_propriedade,
r.cpf_hospede,
p.cpf_locador
```

- **`r.id_reserva`** : identificador único da reserva.
- **`p.id_propriedade`** : identificador do imóvel reservado.
- **`r.cpf_hospede`** : CPF do hóspede que fez a reserva.
- **`p.cpf_locador`** : CPF do proprietário do imóvel.

Esses quatro atributos formam as chaves principais para rastrear quem reservou qual imóvel.

3) Cálculo do Total de Dias Locados

```sql
(r.checkout - r.checkin) AS total_dias_locado,
```

- **Subtração de datas:**
  Subtrair dois valores `DATE` no PostgreSQL retorna um inteiro: o número de dias entre eles.
- **Alias `total_dias_locado` :**
  Renomeia o resultado para clareza, indicando a duração da estadia.

4) Joins com Tabelas Relacionadas

```sql
FROM reserva r
JOIN propriedade p ON r.id_propriedade = p.id_propriedade
JOIN locador loc ON p.cpf_locador   = loc.cpf
JOIN hospede hos ON r.cpf_hospede   = hos.cpf
```

- **`reserva r` → `propriedade p`**
  Relaciona cada reserva ao imóvel usando `id_propriedade` .
- **`propriedade p` → `locador loc`**
  Recupera detalhes do proprietário relacionando `cpf_locador` ao CPF do anfitrião.
- **`reserva r` → `hospede hos`**
  Recupera detalhes do hóspede relacionando `cpf_hospede` ao CPF do hóspede.

Esses joins enriquecem a reserva com informações de anfitrião e hóspede.

5) Recuperação de Nomes e Preço

```sql
loc.nome AS nome_proprietario,
hos.nome AS nome_hospede,
p.preco_noite
```

- **`loc.nome AS nome_proprietario`**
  Busca o nome do anfitrião na tabela `locador`.
- **`hos.nome AS nome_hospede`**
  Busca o nome do hóspede na tabela `hospede`.
- **`p.preco_noite`**
  Recupera a tarifa por noite da tabela `propriedade`.

Essas colunas fornecem detalhes legíveis e valores de preço para cada reserva confirmada.

Explicação das Consultas SQL do Passo 3.5

Esta seção analisa cinco consultas avançadas que exploram sobreposições entre hóspedes e anfitriões, métricas de desempenho de anfitriões, tendências de preço e comparações de idade.

Resultado 3.5

```
lfelipediniz@caue-linux:~/Documents/Github/sql$ docker cp atividade3_3-5.sql pgdev:/atividade3_3-5.sql
docker exec -it pgdev \
  psql -U lfelipediniz -d atividade3_bd \
  -f /atividade3_3-5.sql
Successfully copied 3.07kB to pgdev:/atividade3_3-5.sql
 cpf | nome | sobrenome
-----+------+-----------
(0 rows)

 nome | cidade | qtd_imoveis | total_locacoes
------+--------+-------------+----------------
(0 rows)

   mes    | media_todas | media_confirmadas
---------+-------------+-------------------
 2025-05 |      202.50 |            154.00
 2025-06 |      161.43 |            151.67
(2 rows)

     cpf     |   nome   | datanascimento
-------------+----------+----------------
 10101010101 | Paula    | 1991-08-08
 11111111111 | Zé       | 1985-03-12
 12121212121 | Tiago    | 1987-11-11
 13131313131 | Bianca   | 1993-05-20
 14141414141 | Eduardo  | 1982-03-03
 15151515151 | Mariana  | 1994-09-29
 16161616161 | Fernando | 1985-12-25
 22222222222 | Maria    | 1990-07-24
 33333333333 | Pedro    | 1978-01-05
 44444444444 | Ana      | 1995-10-12
 55555555555 | Lucas    | 1988-04-30
 66666666666 | Joana    | 1992-12-15
 77777777777 | Rafael   | 1983-09-09
 88888888888 | Carla    | 1996-06-18
 99999999999 | Bruno    | 1980-02-28
(15 rows)

     cpf     |  nome   | datanascimento
-------------+---------+----------------
 22222222222 | Maria   | 1990-07-24
 44444444444 | Ana     | 1995-10-12
 66666666666 | Joana   | 1992-12-15
 88888888888 | Carla   | 1996-06-18
 10101010101 | Paula   | 1991-08-08
 13131313131 | Bianca  | 1993-05-20
 15151515151 | Mariana | 1994-09-29
(7 rows)
```

1) Usuários que São Tanto Hóspedes quanto Anfitriões

- **Lógica:**
  Encontrar pessoas cujo CPF aparece em ambas as tabelas `hospede` e `locador`.
- **Técnica:**
  - **JOIN** no CPF idêntico: combina linhas onde o `cpf` do hóspede é igual ao `cpf` do anfitrião, retornando apenas quem está em ambas as tabelas.
- **Caso de uso:**
  Identificar usuários que atuam em dupla função — útil para tratamento especial ou auditoria.

2) Anfitriões com pelo Menos 5 Reservações

- **Lógica:**
  Listar nome e cidade de cada anfitrião, contar quantos imóveis distintos ele possui e o total de reservas desses imóveis, filtrando anfitriões com ≥ 5 reservas.
- **Técnicas:**
  - **Vários JOINs** para ligar `locador` → `propriedade` → `reserva` → `localizacao`.
  - `COUNT(DISTINCT p.id_propriedade)` : conta propriedades únicas por anfitrião.
  - `COUNT(r.id_reserva)` : total de reservas desses imóveis.
  - `GROUP BY loc.cpf, loc.nome, loc.cidade` : agrega métricas por anfitrião e cidade.
  - `HAVING COUNT(r.id_reserva) >= 5` : filtra grupos que atendem ao critério.
- **Caso de uso:**
  Destacar anfitriões de alta atividade para bonificações ou dashboards de performance.

3) Média da Tarifa Noturna por Mês (Todas vs. Confirmadas)

- **Lógica:**
  Para cada mês, calcular duas médias de `preco_noite` : uma para todas as reservas e outra apenas para as confirmadas.
- **Técnicas:**
  - `TO_CHAR(r.checkin, 'YYYY-MM') AS mes` : formata a data de `checkin` em “ano-mês” para agrupamento.
  - `AVG(p.preco_noite)` : calcula a média geral das tarifas.
  - `AVG(CASE WHEN r.status = 'confirmada' THEN p.preco_noite END)` : usa expressão condicional dentro de `AVG` para incluir apenas reservas confirmadas (NULLs são ignorados).
  - `ROUND(..., 2)` : arredonda cada média para duas casas decimais.
  - `GROUP BY mes` e `ORDER BY mes` : garante resultados em ordem cronológica.
- **Caso de uso:**
  Acompanhar tendências de preço e comparar receita efetiva confirmada vs. tarifas listadas.

4) Hóspedes Mais Jovens que Alguns Anfitriões

- **Lógica:**
  Retornar qualquer hóspede cuja data de nascimento seja posterior (i.e., mais jovem) à de pelo menos um anfitrião.
- **Técnicas:**
  - `WHERE EXISTS (SELECT 1 FROM locador l WHERE h.datanascimento > l.datanascimento)` :
    Para cada hóspede `h` , a subquery verifica se existe ao menos um anfitrião `l` nascido antes dele.
- **Caso de uso:**
  Perfilar sobreposição demográfica onde hóspedes são mais jovens que anfitriões.

5) Hóspedes Mais Jovens que Todos os Anfitriões

- **Lógica:**
  Encontrar hóspedes mais jovens que todos os anfitriões do sistema.
- **Técnicas:**
  - `WHERE h.datanascimento > ALL (SELECT l.datanascimento FROM locador l)` :
    Compara a data de nascimento de cada hóspede com todas as datas de nascimento dos anfitriões, retornando apenas quem for estritamente mais jovem que o anfitrião mais jovem.
- **Caso de uso:**
  Identificar o subconjunto mais jovem de hóspedes em relação a toda a população de anfitriões.

Cada consulta demonstra recursos-chave do SQL — JOINs para combinar tabelas, funções agregadas com GROUP BY/HAVING, formatação de datas, agregação condicional e subqueries usando EXISTS/ALL — para responder a perguntas de negócio.

📚 Contexto Acadêmico

Este repositório foi criado para a disciplina **SCC0240 – Sistemas de Banco de Dados** na USP.
Link oficial da disciplina