RentalData

This project is a PostgreSQL-based implementation of a database model for an accommodation and reservation platform, following academic specifications.

It contains SQL scripts to create the schema, populate it with fictitious data, and run analytical queries. The scripts are executed inside a Docker PostgreSQL container.

Table of Contents

Structure

```
.
├── atividade3_3-1.sql          # creates the database and tables (no
integrity constraints)
├── atividades-consultas/       # contains all query-related scripts
│   ├── atividade3_3-2.sql      # loads fictitious data (INSERTs)
│   ├── atividade3_3-3.sql      # basic queries on property data
│   ├── atividade3_3-4.sql      # reservation analysis
│   └── atividade3_3-5.sql      # advanced queries and analytics
├── consultas.sql               # master script that runs all queries
└── saida_consultas.txt         # output of all queries (for report
inclusion)
```

Requirements

- Docker
- PostgreSQL Docker image (automatically pulled when running container)

Installation

1. Start a PostgreSQL container (if not running):

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

2. Copy all files into the container:

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

How to Run

Step 1: Drop and recreate the database + create tables

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

Step 2: Execute all inserts and queries, saving output

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

After this, the file `saida_consultas.txt` will contain the full output of:

- Data insertion
- Basic and advanced queries

Important Notes

- **If you run `consultas.sql` more than once**, it will insert the same data multiple times.
- That's because we **did not implement integrity constraints (e.g. PRIMARY KEY, UNIQUE)** in this version.

- This behavior was intentional, following the instructions to skip constraints in the initial modeling phase.

You can manually clean the base by re-running:

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

Explanation of Step 3.1 SQL Commands

1. Database Initialization

- `DROP DATABASE IF EXISTS atividade3_bd;`
  Ensures any existing database named `atividade3_bd` is removed first, avoiding "already exists" errors when re-running the script.
- `CREATE DATABASE atividade3_bd;`
  Creates a brand-new PostgreSQL database called `atividade3_bd` .
- `\connect atividade3_bd`
  Switches the psql session into the newly created database so that all subsequent `CREATE TABLE` statements apply there.

2. Table Cleanup

- `DROP TABLE IF EXISTS <table_name> CASCADE;`
  For each table, this command safely deletes the table if it exists. The `CASCADE` option also removes any dependent objects (like foreign-key relationships or views) to prevent drop errors.

3. Main Table Definitions

Each `CREATE TABLE` defines an entity's columns and data types without enforcing keys:

- **String types**
  - `VARCHAR(n)` : variable-length strings up to `n` characters (e.g., `VARCHAR(100)` for names).
  - `TEXT` : unlimited-length text, used where size is unpredictable (e.g., addresses and messages).
- **Date/time types**

    - `DATE`, `TIME`, `TIMESTAMP` store calendar dates, clock times, and full timestamps, respectively.
- **Numeric types**
    - `INTEGER`: whole numbers.
    - `NUMERIC(precision, scale)`: exact decimals (e.g., money values with two decimal places).
- **Other types**
    - `CHAR(1)`: fixed-length one-character fields (e.g., gender code).
    - `BOOLEAN`: true/false values.

No primary keys or foreign keys are specified here, keeping the focus solely on table structure.

4. Auto-Increment with `SERIAL`

- Columns declared as `SERIAL` (e.g., `id_propriedade SERIAL`)
  Automatically create an integer column that pulls its values from a hidden sequence, making it easy to generate unique identifiers without manually managing sequences.

5. Associative Tables for Many-to-Many

- Tables like `propriedade_comodidade` and `propriedade_regra`
  Represent N:M relationships by listing pairs of referencing columns. They remain unconstrained in this version, simply holding the associations.

Explanation of Step 3.2 SQL Data-Load Script

This section walks through each logical section of the `atividade3_3-2.sql` script, which populates the `atividade3_bd` database with sample data. No table definitions are repeated here—only the rationale behind each `INSERT`.

1) Inserting into `hospede`

- **Purpose:** Load 15 guest records to simulate real users.
- **Key Points:**
    - Columns listed in the same order as the `CREATE TABLE` in Step 3.1.
    - String literals in quotes match each `VARCHAR` or `TEXT` column.
    - `NULL` used where optional information (phone or email) isn't provided.

- Dates follow the `YYYY-MM-DD` format for the `DATE` type.
- Passwords are stored in plain text here for testing (in real systems, they'd be hashed).

2) Inserting into `locador`

- **Purpose:** Populate 15 property-owner records.
- **Key Points:**
  - Structure parallels `hospede` : same columns, but for hosts instead of guests.
  - Demonstrates handling of `NULL` and mixed-case strings.
  - Ensures there are enough distinct `cpf` values to join later.

3) Inserting into `localizacao`

- **Purpose:** Define 15 distinct geographic locations.
- **Key Points:**
  - Omits the `id_localizacao` column because it's `SERIAL` ; PostgreSQL auto-generates it.
  - Mix of Brazilian cities spread across states, illustrating how to store address metadata.
  - Postal codes (`cep`) and neighborhood names complete the location data.

4) Inserting into `comodidade`

- **Purpose:** Seed 15 amenity names for properties.
- **Key Points:**
  - Only one column (`nome`) is listed in the `INSERT` .
  - Multiple values in a single statement improve load performance.
  - Shows how to bulk-insert simple lookup data.

5) Inserting into `regra`

- **Purpose:** Establish 10 distinct house rules, with a Boolean flag for allowance.
- **Key Points:**
  - `tipo` describes the rule (e.g., "Fumar" or "Pets").
  - `permitido` uses `TRUE` / `FALSE` to indicate rule enforcement.
  - This lookup table drives business logic in booking validations.

6) Inserting into `propriedade`

- **Purpose:** Create 15 property listings tied to hosts and locations.
- **Key Points:**
  - Omits the `id_propriedade` column because it's `SERIAL`; PostgreSQL auto-generates it.
  - Times for `checkin` / `checkout` follow the `HH:MM` format for the `TIME` type.
  - References to `cpf_locador` and `id_localizacao` demonstrate how foreign keys would connect tables.

7) Inserting into `quarto`

- **Purpose:** Define individual rooms within properties (weak entity).
- **Key Points:**
  - Composite identifier: (`id_propriedade` , `numero`) together distinguish each room.
  - Booleans (`banheiroprivativo`) use `TRUE` / `FALSE` .
  - Illustrates one-to-many relationships without explicit constraints.

8) Inserting into `reserva`

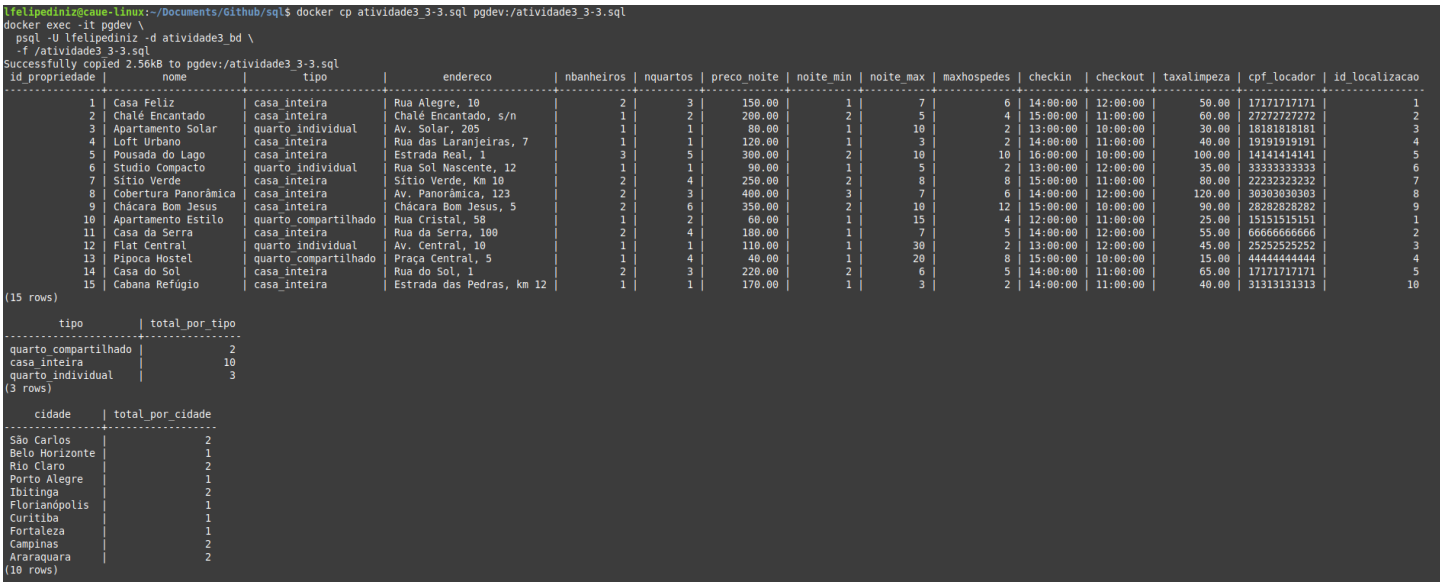
- **Purpose:** Load 15 booking records to simulate reservations.
- **Key Points:**
  - Dates (`datareserva` , `checkin` , `checkout`) in `YYYY-MM-DD` .
  - Monetary fields (`imposto` , `precototal` , `precocomtaxa`) use `NUMERIC(10,2)` .
  - `status` is a text flag (e.g., `confirmada` , `pendente` , `cancelada`).
  - References `id_propriedade` and `cpf_hospede` tie each booking to a guest and property.

Each `INSERT` follows the pattern: list the target columns, then provide matching values for each row. This ensures data integrity and clarity when reviewing or extending the dataset.

Explanation of Step 3.3 SQL Queries

This section demonstrates three basic queries on the populated tables: retrieving all rows, aggregating by property type, and counting by city through a join.

3.3 Result



```
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 | 22232322232 |              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, 58            |          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) Retrieve All Records from `propriedade`

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

- **`SELECT *`**
  Fetches every column from the `propriedade` table.
- **`FROM propriedade`**
  Indicates the source table.
- **Use Case:**
  Quickly inspect all properties, including auto-generated IDs, names, types, locations, pricing, and host references—ideal for a full data dump or debugging.

2) Count Properties by Type

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

- **`SELECT tipo`**
  Chooses the `tipo` column, which categorizes property (e.g., `casa_inteira`, `quarto_individual`).
- **`COUNT(*)`**
  Counts all rows in each group.

- **`AS total_por_tipo`**
  Renames the count column for clarity.
- **`GROUP BY tipo`**
  Aggregates rows sharing the same `tipo` value.
- **Use Case:**
  Understand distribution of listings by category, useful for analytics and reporting.

3) Count Properties per City 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`**
  Links each property (`p`) to its location (`l`) by matching `id_localizacao` .
- **`SELECT l.cidade`**
  Retrieves the city name from the `localizacao` table.
- **`COUNT(*) AS total_por_cidade`**
  Tallies the number of properties in each city, renaming for readability.
- **`GROUP BY l.cidade`**
  Groups results by city to produce one row per location.
- **Use Case:**
  Analyze geographic distribution of properties, essential for market insights and strategic planning.

Explanation of Step 3.4 SQL Query

This query retrieves all confirmed reservations with check-in dates on or after April 24, 2025, and joins related tables to enrich each record with owner and guest names, stay duration, and nightly price.

3.4 Result

```
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) Filtering Criteria

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

- **`r.status = 'confirmada'`**
  Selects only reservations whose status is “confirmed.”
- **`r.checkin >= '2025-04-24'`**
  Ensures the reservation’s check-in date is on or after April 24, 2025.
- **Date Format:**
  Uses the ISO `YYYY-MM-DD` format expected by PostgreSQL for `DATE` comparisons.

2) Core Columns Selected

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

- **`r.id_reserva`** : Unique reservation identifier.
- **`p.id_propriedade`** : Identifier of the property being booked.
- **`r.cpf_hospede`** : CPF (ID) of the guest making the reservation.
- **`p.cpf_locador`** : CPF (ID) of the property owner.

These four attributes form the primary keys for tracking who booked which property.

3) Calculating Total Days Rented

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

- **Date Subtraction:**
  Subtracting two `DATE` values in PostgreSQL yields an integer: the number of days between them.
- **Alias `total_dias_locado` :**
  Renames the result for clarity, indicating the length of stay.

4) Joining Related Tables

```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`**
  Links each reservation to its property record using `id_propriedade` .
- **`propriedade p → locador loc`**
  Retrieves the owner's details by matching the property's `cpf_locador` to the owner's CPF.
- **`reserva r → hospede hos`**
  Retrieves the guest's details by matching the reservation's `cpf_hospede` to the guest's CPF.

These joins enrich the reservation with both owner and guest information.

5) Retrieving Names and Price

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

- **`loc.nome AS nome_proprietario`**
  Fetches the owner's name from the `locador` table.
- **`hos.nome AS nome_hospede`**
  Fetches the guest's name from the `hospede` table.
- **`p.preco_noite`**
  Retrieves the nightly rate from the `propriedade` table.

These columns deliver human-readable details and pricing for each confirmed booking.

Explanation of Step 3.5 SQL Queries

This section analyzes five advanced queries that explore overlaps between guests and hosts, performance metrics for hosts, price trends, and age comparisons.

3.5 Result

```
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) Users Who Are Both Guests and Hosts

- **Logic:**
  Find individuals whose CPF appears in both `hospede` and `locador`.

- **Technique:**
  - **`JOIN` on identical CPF**: Matches rows where a guest's `cpf` equals a host's `cpf`, returning only those in both tables.
- **Use Case:**
  Identify users who play dual roles—helpful for special treatment or auditing.

2) Hosts with at Least 5 Rentals

- **Logic:**
  List each host's name and city, count how many distinct properties they own, and total reservations across those properties, filtering to hosts with ≥ 5 bookings.
- **Techniques:**
  - **Multiple `JOIN`s** to link `locador` → `propriedade` → `reserva` → `localizacao`.
  - `COUNT(DISTINCT p.id_propriedade)`: Counts unique properties per host.
  - `COUNT(r.id_reserva)`: Totals all reservations for those properties.
  - `GROUP BY l.cpf, l.nome, loc.cidade`: Aggregates metrics per host and city.
  - `HAVING COUNT(r.id_reserva) >= 5`: Filters groups to hosts meeting the threshold.
- **Use Case:**
  Spotlight high-activity hosts for rewards or performance dashboards.

3) Average Nightly Rate per Month (All vs. Confirmed)

- **Logic:**
  For each month, compute two averages of `preco_noite`: one over all reservations and one restricted to those confirmed.
- **Techniques:**
  - `TO_CHAR(r.checkin, 'YYYY-MM') AS mes`: Formats the `checkin` date into a "year-month" string for grouping.
  - `AVG(p.preco_noite)`: Calculates the overall average rate.
  - `AVG(CASE WHEN r.status = 'confirmada' THEN p.preco_noite END)`: Uses a conditional expression inside `AVG` to include only confirmed bookings (NULLs ignored).
  - `ROUND(..., 2)`: Rounds each average to two decimal places.
  - `GROUP BY mes` and `ORDER BY mes`: Ensures chronological results.
- **Use Case:**
  Track pricing trends and compare actual confirmed revenue vs. list rates.

4) Guests Younger Than Some Host

- **Logic:**
  Return any guest whose birth date is later (i.e., younger) than at least one host's birth date.
- **Techniques:**
  - `WHERE EXISTS (SELECT 1 FROM locador l WHERE h.datanascimento > l.datanascimento)` :
    For each guest `h`, the subquery checks if there is at least one host `l` born before them.
- **Use Case:**
  Profile the demographic overlap where guests outnumber older hosts.

5) Guests Younger Than All Hosts

- **Logic:**
  Find guests younger than every host in the system.
- **Techniques:**
  - `WHERE h.datanascimento > ALL (SELECT l.datanascimento FROM locador l)` :
    Compares each guest's birth date against the entire set of host birth dates, returning only those strictly younger than the youngest host.
- **Use Case:**
  Identify the youngest subset of guests relative to the entire host population.

Each query demonstrates key SQL features—JOINs for combining tables, aggregate functions with GROUP BY/HAVING, date formatting, conditional aggregation, and subqueries using EXISTS/ALL—to answer complex business questions.

📚 Academic Context

This repository was created for the **SCC0240 – Database Systems** course at USP.
Official course link