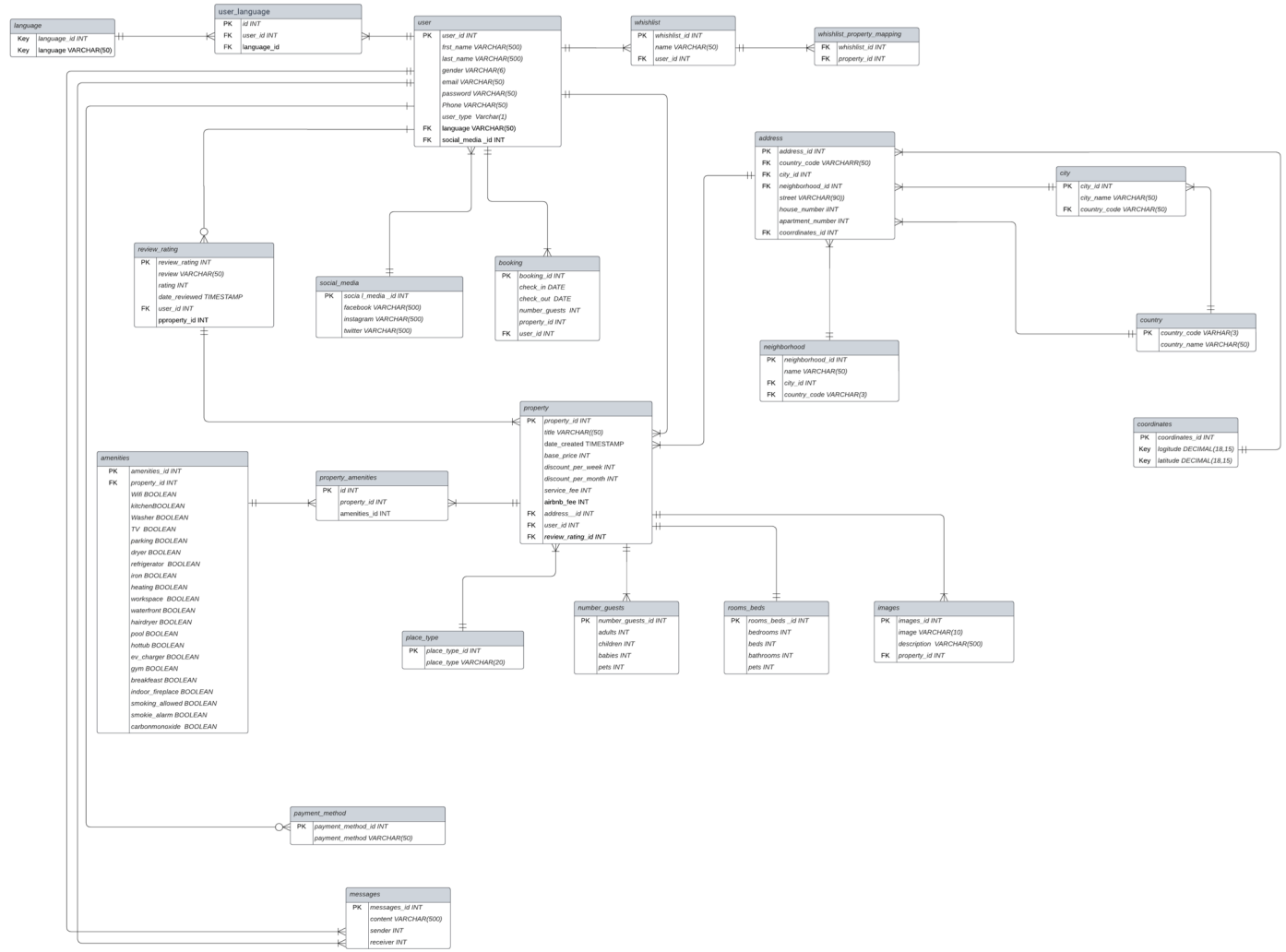
# SQL DATA MART

Airbnb SQL Data Mart

29 august 2023

https://github.com/Oumaymabamoh/Sql_Data_Mart

# AIRBNB ERM




ERD updated version

# CREATE TABLES

The SQL include:

- Create Table
- Attribute Type
- Primary Keys
- Foreign keys

PostgreSQL was selected as the database management system for this project to run SQL queries within the PgAdmin4 environment

To view the SQL files, their documentation, and the data dictionary, please refer to my GitHub repository. https://github.com/Oumaymabamoh/Sql_Data_Mart

```sql
-- Create Airbnb Schema
CREATE SCHEMA IF NOT EXISTS Airbnb;
```

# CREATE SCHEMA

# COUNTRY TABLE

| country_code [PK] character varying (3) | country_name character varying (50) |
|---|---|
| CHE | Switzerland |
| USA | United States |
| AUS | Australia |
| IT | Italy |
| BR | Brazil |
| MEX | Mexico |
| ES | Spain |
| MA | Morocco |
| UK | United Kingdom |
| CA | Canada |
| FR | France |
| GER | Germany |
| GR | Greece |
| NIG | Nigeria |
| CH | China |
| KE | Kenya |
| KOR | Korea |
| JP | Japan |
| NL | Netherlands |
| GHA | Ghana |
| PT | Portugal |

```sql
-- Table Airbnb.country
DROP TABLE IF EXISTS Airbnb.country CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.country (
  country_code VARCHAR(3) NOT NULL,
  country_name VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (country_code));

  SELECT * FROM Airbnb.country;
```

The Airbnb.country table holds country details. Each country is identified by a unique country_code and includes a country_name. The country_code serves as the primary key, ensuring uniqueness. This table helps categorize properties based on their respective countries.

# CITY TABLE

| city_id [PK] integer | city_name character varying (50) | country_code character varying (3) |
|---|---|---|
| 1 | Venise | IT |
| 2 | Berlin | GER |
| 3 | Kenya | KE |
| 4 | Lausanne | CHE |
| 5 | Miami | USA |
| 6 | New York | USA |
| 7 | Porto | PT |
| 8 | London | UK |
| 9 | Liverpool | UK |
| 10 | Perth | AUS |
| 11 | Amsterdam | NL |
| 12 | Accra | GHA |
| 13 | New Mexico | MEX |
| 14 | Manchester | UK |
| 15 | Paris | FR |
| 16 | Strasbourg | FR |
| 17 | Toronto | CA |
| 18 | Seville | ES |
| 19 | Ottawa | CA |
| 20 | Kiyoto | JP |
| 21 | Tokyo | JP |

```sql
-- Create Table Airbnb.city
DROP TABLE IF EXISTS Airbnb.city CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.city (
  city_id SERIAL PRIMARY KEY NOT NULL,
  city_name VARCHAR(50) NOT NULL,
  country_code VARCHAR(3) NOT NULL,
  CONSTRAINT fk_city_country
    FOREIGN KEY (country_code)
    REFERENCES Airbnb.country (country_code)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
-- Creating an index on country_code column
CREATE INDEX IF NOT EXISTS fk_city_country_idx ON Airbnb.city (country_code);
```

The Airbnb.city table stores city information. Each city is identified by a unique city_id and has a city_name. The table is linked to the Airbnb.country table using the country_code column through a foreign key constraint. An index is created on the country_code column for optimized queries involving city-country relationships. This table helps categorize properties based on different cities.

```sql
SELECT * FROM Airbnb.city;
```

# NEIGHBORHOOD TABLE

| neighborhood_id [PK] integer | name character varying (50) | city_id integer | country_code character varying (3) |
|---|---|---|---|
| 1 | Downtown | 17 | CA |
| 2 | Ribeira | 7 | PT |
| 3 | LaPalaya | 18 | ES |
| 4 | Kreuzberg | 2 | GER |
| 5 | Friedrichshain | 2 | GER |
| 6 | Castello | 1 | IT |
| 7 | Cannaregio | 1 | IT |
| 8 | Jordaan | 11 | NL |
| 9 | Westerpark | 11 | NL |
| 10 | Westend | 2 | GER |
| 11 | Ostend | 2 | GER |
| 12 | Downtown | 5 | USA |
| 13 | Manhatten | 6 | USA |
| 14 | SoHo | 6 | USA |
| 15 | OldTown | 21 | JP |
| 16 | Sydney H | 10 | AUS |
| 17 | ByWard Market | 19 | CA |
| 18 | Downtown | 17 | CA |
| 19 | Le Marais | 15 | FR |

```sql
-- Table Airbnb.neighborhood
DROP TABLE IF EXISTS Airbnb.neighborhood CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.neighborhood (
  neighborhood_id SERIAL PRIMARY KEY NOT NULL,
  name VARCHAR(50) NOT NULL,
  city_id INT NULL DEFAULT NULL,
  country_code VARCHAR(3) NULL DEFAULT NULL,
  CONSTRAINT fk_country
    FOREIGN KEY (country_code)
    REFERENCES Airbnb.country (country_code)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_neighborhood_city1
    FOREIGN KEY (city_id)
    REFERENCES Airbnb.city (city_id));
-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_neighborhood_city1_idx ON Airbnb.neighborhood (city_id);
CREATE INDEX IF NOT EXISTS fk_country_idx ON Airbnb.neighborhood (country_code);
```

```sql
SELECT * FROM Airbnb.neighborhood;
```

The table records neighborhood details. Each neighborhood is assigned a unique neighborhood_id and includes a name. The table is linked to related data in the Airbnb.country and Airbnb.city tables through foreign key constraints. Indexes are created for optimizing queries involving city_id and country_code. This table helps organize properties by neighborhoods within cities.

# COORDINATES TABLE

| coordinates_id [PK] integer | longitude numeric (18,15) | latitude numeric (18,15) |
|---|---|---|
| 1 | -5.994072000000000 | 37.392529000000000 |
| 2 | -118.243683000000000 | 34.052235000000000 |
| 3 | -73.935242000000000 | 40.730610000000000 |
| 4 | 13.404954000000000 | 52.520008000000000 |
| 5 | 6.633597000000000 | 46.519962000000000 |
| 6 | -85.766941000000000 | 86.783112000000000 |
| 7 | 91.146779000000000 | 0.184330000000000 |
| 8 | 132.221576000000000 | 62.039040000000000 |
| 9 | -59.259278000000000 | 44.039268000000000 |
| 10 | -142.833542053200000 | 7.667052000000000 |
| 11 | 33.367663211200000 | -16.574861209600000 |
| 12 | -108.235957568800000 | 11.580463513600000 |
| 13 | 72.692534574300000 | -24.338221772800000 |
| 14 | 100.873753942400000 | -6.204551114900000 |
| 15 | -136.260573865500000 | -56.484510723200000 |
| 16 | -138.967112241000000 | 6.964609945800000 |
| 17 | -74.162321011700000 | 52.518248685300000 |
| 18 | 22.596504619100000 | 53.208190755066800 |
| 19 | 103.709601429600000 | -10.733697666821800 |
| 20 | -3.017009888513400 | -105.228814580200000 |

```sql
-- Table Airbnb.coordinates
DROP TABLE IF EXISTS Airbnb.coordinates CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.coordinates (
  coordinates_id SERIAL PRIMARY KEY NOT NULL,
  longitude DECIMAL(18,15) NOT NULL,
  latitude DECIMAL(18,15) NOT NULL);
```

```sql
SELECT * FROM Airbnb.coordinates;
```

The Airbnb.coordinates table holds geographic coordinates. Each set of coordinates has a unique coordinates_id and includes longitude and latitude values. This table is used to pinpoint precise locations on a map for Airbnb properties and other related data.

# ADDRESS TABLE

| address_id [PK] integer | country_code character varying (3) | city_id integer | neighborhood_id integer | street character varying (90) | house_number integer | apartment_number integer | coordinates_id integer |
|---|---|---|---|---|---|---|---|
| 1 | NL | 11 | 8 | De Nieuwe Hoogstraat | 260 | 7 | 1 |
| 2 | NL | 11 | 9 | Javastraat | 19 | 40 | 2 |
| 3 | USA | 5 | 3 | fener sokak | 433 | 2 | 3 |
| 4 | FR | 15 | 19 | Revoli Street | 45 | 9 | 4 |
| 5 | FR | 16 | 19 | champs Street | 351 | 4 | 5 |
| 6 | JP | 20 | 15 | JPN Street | 11 | 1 | 6 |
| 7 | JP | 20 | 15 | Nakamise Dori | 33 | 10 | 7 |
| 8 | GER | 3 | 10 | Ostend Strasse | 184 | 50 | 8 |
| 9 | GER | 2 | 10 | Berliner Strasse | 1154 | 8 | 9 |
| 10 | ES | 18 | 3 | Las Rambalas | 2 | 20 | 10 |
| 11 | PT | 7 | 2 | Avenida Correia | 42 | 6 | 11 |
| 12 | ES | 18 | 3 | Champs Street | 1789 | 33 | 12 |
| 13 | PT | 7 | 2 | Rua de Paiva | 2010 | 3 | 13 |
| 14 | USA | 5 | 7 | Fifth Avenue | 212 | 71 | 14 |
| 15 | USA | 6 | 7 | Wall Street | 1096 | 34 | 15 |
| 16 | USA | 6 | 13 | Crosby Street | 2169 | 12 | 16 |
| 17 | CA | 19 | 18 | Yonge Street | 43 | 59 | 17 |
| 18 | CA | 19 | 18 | Sparks Street | 34 | 60 | 18 |
| 19 | AUS | 21 | 16 | Oxford Street | 3891 | 5 | 19 |
| 20 | AUS | 21 | 16 | Gloud Street | 59 | 14 | 20 |

The Airbnb.address table stores property addresses. Each address has a unique address_id and includes details like country_code, city_id, neighborhood_id, street, house_number, and apartment_number. The table links to related tables (Airbnb.country, Airbnb.city, Airbnb.neighborhood, and Airbnb.coordinates) through foreign key constraints. Indexes optimize queries based on linked attributes (country_code, city_id, neighborhood_id, coordinates_id).

```sql
-- Table Airbnb.address
DROP TABLE IF EXISTS Airbnb.address CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.address (
  address_id SERIAL PRIMARY KEY NOT NULL,
  country_code VARCHAR(3) NOT NULL,
  city_id INT NOT NULL,
  neighborhood_id INT NOT NULL,
  street VARCHAR(90) NOT NULL,
  house_number INT NOT NULL,
  apartment_number INT NULL DEFAULT NULL,
  coordinates_id INT NOT NULL,
  CONSTRAINT fk
    FOREIGN KEY (country_code)
    REFERENCES Airbnb.country (country_code)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk2
    FOREIGN KEY (city_id)
    REFERENCES Airbnb.city (city_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk3
    FOREIGN KEY (neighborhood_id)
    REFERENCES Airbnb.neighborhood (neighborhood_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk4
    FOREIGN KEY (coordinates_id)
    REFERENCES Airbnb.coordinates (coordinates_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_idx ON Airbnb.address (country_code);
CREATE INDEX IF NOT EXISTS fk2_idx ON Airbnb.address (city_id);
CREATE INDEX IF NOT EXISTS fk3_idx ON Airbnb.address (neighborhood_id);
CREATE INDEX IF NOT EXISTS fk4_idx ON Airbnb.address (coordinates_id);
```

```sql
SELECT * FROM Airbnb.address;
```

# AMENITIES TABLE

| amenities_id [PK] integer | property_id integer | wifi smallint | kitchen smallint | washer smallint | tv smallint | parking smallint | dryer smallint | refrigerator smallint | iron smallint | heating smallint | workspace smallint | waterfront smallint | hairdryer smallint | pool smallint | ho sm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | |
| 2 | 2 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | |
| 3 | 3 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | |
| 4 | 4 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 5 | 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | |
| 6 | 6 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | |
| 7 | 7 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 8 | 8 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | |
| 9 | 9 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| 10 | 10 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | |

The Airbnb.amenities table records property amenities. Each amenity has a unique amenities_id. Amenities such as WiFi, kitchen, TV, etc., are marked with SMALLINT values indicating availability. The table associates amenities with a property_id through a foreign key. This table allows properties to list available amenities for guests.

```
-- Table Airbnb.amenities
DROP TABLE IF EXISTS Airbnb.amenities CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.amenities (
  amenities_id SERIAL PRIMARY KEY NOT NULL,
  wifi SMALLINT,
  kitchen SMALLINT,
  washer SMALLINT,
  tv SMALLINT,
  parking SMALLINT,
  dryer SMALLINT,
  refrigerator SMALLINT,
  iron SMALLINT,
  heating SMALLINT,
  workspace SMALLINT,
  waterfront SMALLINT,
  hairdryer SMALLINT,
  pool SMALLINT,
  hottub SMALLINT,
  ev_charger SMALLINT,
  gym SMALLINT,
  breakfast SMALLINT,
  indoor_fireplace SMALLINT,
  smoking_allowed SMALLINT,
  smoke_alarm SMALLINT,
  carbonmonoxide_alarm SMALLINT);
```

```
SELECT * FROM Airbnb.amenities;
```

# SOCIAL MEDIA TABLE

| social_media_id [PK] integer | facebook character varying (500) | instagram character varying (500) | twitter character varying (500) |
|---|---|---|---|
| 1 | user_a | user_a | user1 |
| 2 | user_b | user_b | user2 |
| 3 | user_c | user_c | user_c |
| 4 | user_d | user_d | user_d |
| 5 | user_e | user_e | user_e |
| 6 | user_f | user_f | user_f |
| 7 | user_g | user_g | user_g |
| 8 | user_h | user_h | user_h |
| 9 | user_i | user_i | user_i |
| 10 | user_j | user_j | user_j |
| 11 | user_k | user_k | user_k |
| 12 | user_l | user_l | user_l |
| 13 | user_m | user_m | user_m |
| 14 | user_n | user_n | user_n |
| 15 | user_o | user_o | user_o |
| 16 | user_p | user_p | user_p |
| 17 | user_q | user_q | user_q |
| 18 | user_r | user_r | user_r |
| 19 | user_s | user_s | user_s |
| 20 | user_t | user_t | user_t |

```sql
-- Table Airbnb.social_media
DROP TABLE IF EXISTS Airbnb.social_media CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.social_media (
  social_media_id SERIAL PRIMARY KEY NOT NULL,
  facebook VARCHAR(500)  DEFAULT NULL,
  instagram VARCHAR(500) DEFAULT NULL,
  twitter VARCHAR(500)  DEFAULT NULL);
```

```sql
SELECT * FROM Airbnb.social_media;
```

The Airbnb.social_media table manages social media links for users. Each record has a unique social_media_id and includes optional links to Facebook, Instagram, and Twitter. This table allows users to provide their social media profiles for connecting on different platforms.

# LANGUAGE TABLE

| language<br>[PK] character varying (50) |
| --- |
| English |
| Deutch |
| Spanish |
| Arabic |
| French |
| Korean |
| japanese |
| Italish |
| korean |
| Portuguese |

```sql
-- Table Airbnb.language
DROP TABLE IF EXISTS Airbnb.language CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.language (
  language VARCHAR(50) NOT NULL,
  PRIMARY KEY (language));
```

```sql
SELECT * FROM Airbnb.language;
```

The Airbnb.language table holds language options. Each language is represented by a unique language value, forming the primary key. This table allows users to select preferred languages.

# USER TABLE

| user_type<br>character varying (1) | user_count<br>bigint |
|---|---|
| h | 6 |
| g | 19 |

The Airbnb.user table stores user information. Each user is identified by a unique user_id and includes details like first_name, last_name, gender, email, password, phone, user_type, and preferred language. The table also links to Airbnb.social_media through social_media_id, and enforces language choice using foreign key constraints referencing Airbnb.language. Indexes are created for optimized queries on social_media_id and language.

```
-- Table Airbnb.user
DROP TABLE IF EXISTS Airbnb.user CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.user (
  user_id SERIAL PRIMARY KEY NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  gender VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
  password VARCHAR(50) NOT NULL,
  phone VARCHAR(50) DEFAULT NULL,
  user_type VARCHAR(1) NOT NULL,
  language VARCHAR(50) NULL DEFAULT 'English',
  social_media_id INT NULL DEFAULT 1,
  CONSTRAINT fk_user_social_media1
    FOREIGN KEY (social_media_id)
    REFERENCES Airbnb.social_media (social_media_id),
  CONSTRAINT language
    FOREIGN KEY (language)
    REFERENCES Airbnb.language (language)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_user_social_media1_idx ON Airbnb.user (social_media_id);
CREATE INDEX IF NOT EXISTS language_idx ON Airbnb.user ("language");
```

```
SELECT user_type, COUNT(*) as user_count
FROM Airbnb.user
GROUP BY user_type;
```

# USER_LANGUAGE TABLE

| user_id integer | first_name character varying (50) | last_name character varying (50) | email character varying (50) | language character varying (50) |
|---|---|---|---|---|
| 1 | John | Doe | aa@example.org | English |
| 2 | Rita | Frazier | bb@example.net | Deutch |
| 3 | Andrew | Good | cc@example.org | Spanish |
| 4 | John | Char | dd@example.com | Arabic |
| 5 | Julie | Salinas | ee@example.org | French |
| 6 | John | Steph | ff@example.org | Korean |
| 7 | maria | Ith | gg@example.com | japanese |
| 8 | Heny | Jody | hh@example.org | Italish |
| 9 | Sarah | Too | ii@example.org | korean |
| 10 | Bella | Lee | jj@example.net | Portuguese |

The Airbnb.user_language table manages user language preferences. Each user is uniquely identified by a user_id and can specify their preferred language. This choice personalizes their experience on Airbnb, ensuring content and communication align with their language selection. To maintain data integrity, foreign key constraints link user_id to the Airbnb.user table and language_id to the Airbnb.language table, verifying valid user-language associations. This table enhances Airbnb's ability to offer a tailored multilingual platform experience, while indexes on social_media_id and language optimize query performance.

```
--Table Airbnb.user_language
-- Drop the table if it exists
DROP TABLE IF EXISTS Airbnb.user_language;

-- Create the table if it doesn't exist
CREATE TABLE IF NOT EXISTS Airbnb.user_language (
  id SERIAL PRIMARY KEY NOT NULL,
  user_id INT NOT NULL,
  language_id INT NOT NULL,
  CONSTRAINT fk_join_user
    FOREIGN KEY (user_id)
    REFERENCES Airbnb.user (id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_join_language
    FOREIGN KEY (language_id)
    REFERENCES Airbnb.language (id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_join_user_idx ON Airbnb.user_language (user_id);
CREATE INDEX IF NOT EXISTS fk_join_language_idx ON Airbnb.user_language (language_id);
```

```
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    u.email,
    l.language
FROM
    Airbnb.user_language ul
JOIN
    Airbnb.user u ON ul.user_id = u.user_id
JOIN
    Airbnb.language l ON ul.language_id = l.language_id;
```

# PROPERTY TABLE

| property_id [PK] integer | title character varying (50) | highest_rating integer |
|---|---|---|
| 3 | Excepteur sint | 5 |
| 19 | nulla pariatur | 5 |
| 4 | commodo consequat | 5 |
| 11 | Temporibus autem | 5 |
| 13 | Nam libero | 5 |

The  table connects wishlists and property listings. Each mapping associates a wishlist_id with a property_id, forming a connection between user wishlists and desired properties. The primary key consists of both wishlist_id and property_id. Foreign key constraints ensure the integrity of relationships with the Airbnb.property and Airbnb.wishlist tables. Indexes are created to optimize queries involving property and wishlist IDs.

```
CONSTRAINT fk_property_type_of_place1
   FOREIGN KEY (property_id)
   REFERENCES Airbnb.place_type (place_type_id)
   ON DELETE CASCADE
   ON UPDATE CASCADE,
CONSTRAINT fk_property_user1
   FOREIGN KEY (user_id)
   REFERENCES Airbnb.user (user_id)
   ON DELETE CASCADE
   ON UPDATE CASCADE,
CONSTRAINT fk_property_amenities
   FOREIGN KEY (property_id)
   REFERENCES Airbnb.amenities (amenities_id)
   ON DELETE CASCADE
   ON UPDATE CASCADE,
CONSTRAINT fk_property_price1
   FOREIGN KEY (price_id)
   REFERENCES Airbnb.price (price_id)
   ON DELETE CASCADE
   ON UPDATE CASCADE
 );

-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_property_address1_idx ON Airbnb.property (address_id);
CREATE INDEX IF NOT EXISTS fk_property_user1_idx ON Airbnb.property (user_id);
CREATE INDEX IF NOT EXISTS fk_property_review_rating_1_idx ON Airbnb.property (review_rating_id);
CREATE INDEX IF NOT EXISTS fk_property_amenities_idx ON Airbnb.property (property_id);
```

```
SELECT p.property_id, p.title, COALESCE(MAX(r.rating), 0) AS highest_rating
FROM Airbnb.property p
LEFT JOIN Airbnb.review_rating r ON p.review_rating_id = r.review_rating_id
GROUP BY p.property_id, p.title
ORDER BY highest_rating DESC
LIMIT 5;
```

# AMENITIES_PROPERTY TABLE

| property_id integer | property_title character varying (50) | user_first_name character varying (50) | user_last_name character varying (50) | amenities_id integer | wifi smallint | kitchen smallint | washer smallint | tv smallint |
|---|---|---|---|---|---|---|---|---|
| 1 | Lorem ipsum | John | Doe | 4 | 1 | 0 | 0 | 1 |
| 2 | consectetur adipiscing | Rita | Frazier | 6 | 1 | 0 | 1 | 0 |
| 3 | Excepteur sint | Julie | Salinas | 5 | 0 | 1 | 0 | 1 |
| 4 | commodo consequat | maria | Ith | 10 | 1 | 1 | 1 | 0 |
| 5 | ut perspiciatis | John | Steph | 9 | 0 | 0 | 1 | 1 |
| 6 | Quis autem vel | Una | Yong | 7 | 0 | 0 | 1 | 1 |
| 7 | suscipit laboriosam | Mary | Lily | 8 | 1 | 1 | 0 | 0 |
| 8 | Neque porro | Emily | Ghk | 19 | 0 | 1 | 0 | 1 |
| 9 | sed quia | Rose | Sky | 1 | 1 | 1 | 1 | 1 |
| 10 | At vero eos | Emily | Ghk | 2 | 1 | 1 | 0 | 1 |
| 11 | Temporibus autem | Noa | Nop | 11 | 0 | 1 | 0 | 1 |
| 12 | Itaque earum | Noa | Nop | 12 | 1 | 0 | 1 | 0 |
| 13 | Nam libero | John | Char | 20 | 0 | 1 | 1 | 0 |
| 14 | dolorum fuga | Una | Yong | 18 | 1 | 0 | 1 | 0 |
| 15 | adipisci velit | Heny | Jody | 13 | 0 | 0 | 0 | 1 |
| 16 | qui ratione | Adam | Efg | 17 | 0 | 1 | 1 | 0 |
| 17 | quasi architecto | Emily | Ghk | 21 | 1 | 0 | 1 | 1 |
| 18 | Ut enim | Sonia | Tuv | 16 | 1 | 1 | 0 | 1 |
| 19 | nulla pariatur | Mary | Lily | 14 | 1 | 1 | 0 | 0 |
| 20 | dolorem eum | Rose | Sky | 11 | 0 | 1 | 0 | 1 |
| 21 | irure dolor | Andrew | Good | 20 | 0 | 1 | 1 | 0 |

The Airbnb.property_amenities table serves as a bridge between property listings and their associated amenities. It maintains a record of amenities linked to specific properties. Each entry in this table includes a unique id, a property_id referring to the property the amenity belongs to, and an amenities_id that references a specific amenity from the Airbnb.amenities table. Foreign key constraints are applied to ensure data integrity, with amenities_id referencing Airbnb.amenities and property_id referencing Airbnb.property. Indexes are created for optimized queries on the amenities_id.

```sql
-- Table Airbnb.property_amenities
-- Drop the table if it exists
DROP TABLE IF EXISTS Airbnb.property_amenities;

CREATE TABLE IF NOT EXISTS Airbnb.property_amenities (
  id SERIAL PRIMARY KEY NOT NULL,
  property_id INT NOT NULL,
  amenities_id INT NOT NULL,
  CONSTRAINT fk_join_amenities
    FOREIGN KEY (amenity_id)
    REFERENCES Airbnb.amenities (id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_join_property
    FOREIGN KEY (property_id)
    REFERENCES Airbnb.property (id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Creating an index
CREATE INDEX IF NOT EXISTS join_amenities_idx ON Airbnb.property_amenities (amenities_id);
```

```sql
SELECT
    p.property_id,
    p.title AS property_title,
    u.first_name AS user_first_name,
    u.last_name AS user_last_name,
    a.amenities_id,
    a.wifi,
    a.kitchen,
    a.washer,
    a.tv
FROM
    Airbnb.property p
JOIN
    Airbnb.property_amenities pa ON p.property_id = pa.property_id
JOIN
    Airbnb.amenities a ON pa.amenities_id = a.amenities_id
JOIN
    Airbnb.user u ON p.user_id = u.user_id
ORDER BY
    p.property_id;
```

# IMAGE TABLE

| property_id [PK] integer | title character varying (50) | image_count bigint |
|---|---|---|
| 2 | consectetur adipiscing | 3 |
| 3 | Excepteur sint | 2 |
| 4 | commodo consequat | 3 |
| 5 | ut perspiciatis | 2 |
| 6 | Quis autem vel | 3 |
| 8 | Neque porro | 3 |
| 10 | At vero eos | 2 |
| 11 | Temporibus autem | 2 |
| 13 | Nam libero | 3 |
| 16 | qui ratione | 3 |
| 20 | dolorem eum | 3 |

The Airbnb.images table manages property images. Each image has a unique images_id and includes an image file name (image), a description (description), and is associated with a property (property_id). The foreign key constraint links the property to the Airbnb.property table, ensuring data consistency. An index on property_id optimizes queries related to property images.

```sql
-- Create Table Airbnb.images
CREATE TABLE IF NOT EXISTS Airbnb.images (
  images_id SERIAL PRIMARY KEY NOT NULL,
  image VARCHAR(10) DEFAULT 'file.jpg',
  description VARCHAR(500),
  property_id INT,
  CONSTRAINT fk_images_property_1
    FOREIGN KEY (property_id)
    REFERENCES Airbnb.property (property_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
-- Creating index
CREATE INDEX IF NOT EXISTS fk_images_property_1_idx ON Airbnb.images (property_id);
```

```sql
SELECT p.property_id, p.title, COUNT(i.images_id) AS image_count
FROM Airbnb.property p
JOIN Airbnb.images i ON p.property_id = i.property_id
GROUP BY p.property_id, p.title
HAVING COUNT(i.images_id) > 1;
```

# REVIEW RATING TABLE

| first_name character varying (50) | last_name character varying (50) | Average_rating numeric |
|---|---|---|
| luke | Lmn | 5.00 |
| Tim | May | 5.00 |
| Dan | Ijk | 5.00 |
| Sarah | Too | 5.00 |
| John | Doe | 5.00 |
| Sonia | Tuv | 5.00 |
| Una | Yong | 5.00 |
| Heny | Jody | 5.00 |
| Neil | Pal | 4.50 |
| Leah | black | 4.00 |
| maria | Ith | 4.00 |
| Noa | Nop | 4.00 |

The Airbnb.review_rating table stores reviews and ratings for properties. Each review has a unique review_rating_id, containing review text (review), a rating (rating), and review date (date_reviewed). Users and properties are linked through foreign keys (user_id, property_id). Indexes on these keys optimize queries for reviews and ratings.

```
-- Table Airbnb.review_rating
DROP TABLE IF EXISTS Airbnb.review_rating CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.review_rating (
  review_rating_id SERIAL PRIMARY KEY NOT NULL,
  review VARCHAR(500) NULL DEFAULT NULL,
  rating INT NULL DEFAULT NULL,
  date_reviewed TIMESTAMP NULL DEFAULT NULL,
  user_id INT NOT NULL,
  property_id INT NOT NULL,
 CONSTRAINT fk_review_rating_property1
    FOREIGN KEY (property_id)
    REFERENCES Airbnb.property (property_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_review_rating_user1
    FOREIGN KEY (user_id)
    REFERENCES Airbnb.user (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
 );
-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_review_rating_user1_idx ON Airbnb.review_rating (user_id);
CREATE INDEX IF NOT EXISTS fk_review_rating_property1_idx ON Airbnb.review_rating (property_id);
```

```
SELECT u.first_name, u.last_name,
ROUND(AVG(rr.rating), 2) AS "Average_rating"
FROM Airbnb.review_rating rr
JOIN Airbnb.user u ON rr.user_id = u.user_id
GROUP BY u.user_id
ORDER BY AVG(rr.rating) DESC
LIMIT 12;
```

# MESSAGE TABLE

```sql
CREATE OR REPLACE VIEW sender_receiver_message AS
SELECT
    CONCAT(sender_user.first_name, ' ', sender_user.last_name) AS sender_name,
    CONCAT(receiver_user.first_name, ' ', receiver_user.last_name) AS receiver_name,
    messages.content,
    messages.messages_id
FROM Airbnb.messages messages
JOIN Airbnb.user sender_user ON messages.sender = sender_user.user_id
JOIN Airbnb.user receiver_user ON messages.receiver = receiver_user.user_id;

--
SELECT * FROM sender_receiver_message;
```

| sender_name<br>text | receiver_name<br>text | content<br>character varying (500) | messages_id<br>integer |
|---|---|---|---|
| Sarah Too | Sam Bert | Duis aute irure dolor in reprehenderit in voluptate velit esse cill… | 2 |

The Airbnb.messages table stores messages between users. Each message has a unique messages_id. Messages contain text (content) and are sent by a user (sender) to another user (receiver). Foreign key constraints link the sender and receiver to the Airbnb.user table. Indexes are created on the sender and receiver columns for efficient querying.

```sql
-- Create Table Airbnb.messages
-- Drop Table if it exists
DROP TABLE IF EXISTS Airbnb.messages CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.messages (
  messages_id SERIAL PRIMARY KEY NOT NULL,
  content VARCHAR(500),
  sender INT,
  receiver INT,
  CONSTRAINT receiver_fk
    FOREIGN KEY (receiver)
    REFERENCES Airbnb.user (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT sender_fk
    FOREIGN KEY (sender)
    REFERENCES Airbnb.user (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
-- Creating indexes
CREATE INDEX IF NOT EXISTS sender_idx ON Airbnb.messages (sender);
CREATE INDEX IF NOT EXISTS receiver_idx ON Airbnb.messages (receiver);
```

```sql
SELECT *
FROM sender_receiver_message
WHERE sender_name = 'Sarah Too';
```

# NUMBER GUEST TABLE

| property_title character varying (50) | adults integer | children integer | babies integer | pets integer |
|---|---|---|---|---|
| Lorem ipsum | 2 | 1 | 0 | 0 |
| consectetur adipiscing | 4 | 2 | 1 | 1 |
| Excepteur sint | 3 | 0 | 0 | 0 |
| commodo consequat | 1 | 0 | 1 | 0 |
| ut perspiciatis | 2 | 0 | 0 | 1 |
| Quis autem vel | 6 | 3 | 2 | 0 |
| suscipit laboriosam | 2 | 1 | 1 | 1 |
| Neque porro | 4 | 2 | 0 | 0 |
| sed quia | 2 | 0 | 0 | 0 |
| At vero eos | 3 | 1 | 0 | 1 |
| Temporibus autem | 1 | 0 | 0 | 0 |
| Itaque earum | 2 | 1 | 1 | 0 |
| Nam libero | 4 | 0 | 0 | 0 |
| dolorum fuga | 2 | 0 | 0 | 1 |
| adipisci velit | 3 | 2 | 1 | 0 |
| qui ratione | 4 | 3 | 1 | 0 |
| quasi architecto | 1 | 0 | 0 | 0 |
| Ut enim | 2 | 0 | 0 | 1 |
| nulla pariatur | 1 | 3 | 1 | 0 |
| dolorem eum | 1 | 1 | 0 | 0 |
| irure dolor | 1 | 4 | 1 | 1 |

The  table stores information about the number of guests allowed at property listings on Airbnb.Each entry has a unique ID (number_guests_id) and records the number of adults, children, babies, and pets allowed. The table is linked to property listings through a foreign key (number_guests_id), ensuring accurate guest information for each property.

```sql
-- Table Airbnb.number_guests
DROP TABLE IF EXISTS Airbnb.number_guests CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.number_guests (
  number_guests_id SERIAL PRIMARY KEY NOT NULL,
  adults INT NOT NULL,
  children INT NULL DEFAULT NULL,
  babies INT NULL DEFAULT NULL,
  pets INT NULL DEFAULT NULL,
  CONSTRAINT fk_number_of_guests_property
    FOREIGN KEY (number_guests_id)
    REFERENCES Airbnb.property (property_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE);


  SELECT
      p.title AS property_title,
      ng.adults,
      ng.children,
      ng.babies,
      ng.pets
  FROM
      Airbnb.property p
  JOIN
      Airbnb.number_guests ng ON p.property_id = ng.number_guests_id;
```

# PRICE TABLE

| property_id integer | property_title character varying (50) | base_price integer | discount_per_week integer | discount_per_month integer | service_fee integer | airbnb_fee integer |
|---|---|---|---|---|---|---|
| 1 | Lorem ipsum | 59 | 7 | 12 | 10 | 8 |
| 2 | consectetur adipiscing | 45 | 8 | 10 | 5 | 3 |
| 3 | Excepteur sint | 68 | 5 | 12 | 2 | 4 |
| 4 | commodo consequat | 97 | 5 | 15 | 10 | 8 |
| 5 | ut perspiciatis | 32 | 2 | 12 | 10 | 5 |
| 6 | Quis autem vel | 29 | 4 | 15 | 18 | 2 |
| 9 | sed quia | 76 | 1 | 16 | 12 | 6 |
| 10 | At vero eos | 102 | 2 | 14 | 5 | 2 |
| 12 | Itaque earum | 46 | 6 | 13 | 9 | 14 |
| 13 | Nam libero | 111 | 1 | 13 | 12 | 8 |
| 14 | dolorum fuga | 119 | 0 | 11 | 9 | 5 |
| 16 | qui ratione | 120 | 5 | 14 | 3 | 5 |
| 18 | Ut enim | 92 | 6 | 10 | 18 | 2 |
| 19 | nulla pariatur | 62 | 3 | 10 | 20 | 4 |
| 20 | dolorem eum | 84 | 7 | 8 | 13 | 1 |
| 21 | irure dolor | 80 | 2 | 7 | 13 | 4 |

The table manages pricing details for property listings on the Airbnb platform. Each property's price is assigned a unique price_id, and the table stores various pricing components such as base_price, discount_per_week, discount_per_month, service_fee, and airbnb_fee. The table also includes a foreign key constraint to the Airbnb.property table via the price_id, ensuring that pricing information is associated with specific property listings. This table enables Airbnb to effectively track and manage property prices, discounts, and fees for different listings.

```sql
SELECT
    p.property_id,
    p.title AS property_title,
    pr.base_price,
    pr.discount_per_week,
    pr.discount_per_month,
    pr.service_fee,
    pr.airbnb_fee
FROM
    Airbnb.property p
JOIN
    Airbnb.price pr ON p.property_id = pr.price_id
WHERE
    pr.base_price BETWEEN 8 AND 120;
```


```sql
SELECT
    p.property_id,
    p.title AS property_title,
    pr.base_price,
    pr.discount_per_week,
    pr.discount_per_month,
    pr.service_fee,
    pr.airbnb_fee
FROM
    Airbnb.property p
JOIN
    Airbnb.price pr ON p.property_id = pr.price_id
WHERE
    pr.base_price BETWEEN 8 AND 120;
```

# PAYMENT METHODS TABLE

| user_id integer | first_name character varying (50) | last_name character varying (50) | payment_method character varying (50) |
|---|---|---|---|
| 1 | John | Doe | googlepay |
| 2 | Rita | Frazier | debit_card |
| 3 | Andrew | Good | applepay |
| 4 | John | Char | credit_card |
| 5 | Julie | Salinas | paypal |
| 6 | John | Steph | paypal |
| 7 | maria | Ith | debit_card |
| 8 | Heny | Jody | paypal |
| 9 | Sarah | Too | googlepay |
| 10 | Bella | Lee | debit_card |
| 11 | Adam | Efg | applepay |
| 12 | Emily | Ghk | paypal |
| 13 | Dan | Ijk | applepay |
| 14 | luke | Lmn | googlepay |
| 15 | Rose | Sky | googlepay |
| 16 | Una | Yong | credit_card |
| 17 | Noa | Nop | applepay |
| 18 | Sonia | Tuv | applepay |
| 19 | Mary | Lily | googlepay |
| 20 | Neil | Pal | credit_card |

The table stores information about different payment methods available on the Airbnb platform. Each payment method is assigned a unique payment_method_id and described by the payment_method column. The table includes a foreign key constraint to the Airbnb.user table through the payment_method_id, allowing users to associate specific payment methods with their profiles. This table facilitates the management of various payment options for Airbnb transactions.

```
-- Table Airbnb.payment_method
DROP TABLE IF EXISTS Airbnb.payment_method;

CREATE TABLE IF NOT EXISTS Airbnb.payment_method (
  payment_method_id SERIAL PRIMARY KEY NOT NULL,
  payment_method VARCHAR(50) NULL DEFAULT NULL,
  CONSTRAINT fk_paymentmethod_user_1
    FOREIGN KEY (payment_method_id)
    REFERENCES Airbnb.user (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

```
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    pm.payment_method
FROM
    Airbnb.user u
JOIN
    Airbnb.payment_method pm ON u.user_id = pm.payment_method_id;
```

# WISHLIST TABLE

| wishlist_id integer | wishlist_name character varying (50) | first_name character varying (50) | last_name character varying (50) |
|---|---|---|---|
| 1 | Nemo enim ipsam voluptatem | John | Doe |
| 14 | amet, consectetuer adipiscing | John | Steph |
| 20 | ante dictum mi, | Heny | Jody |
| 15 | Temporibus autem quibusdam | Heny | Jody |
| 9 | porttitor scelerisque neque. | Heny | Jody |

The Airbnb.wishlist table is designed to manage users' wishlists on the Airbnb platform. Each wishlist is identified by a unique wishlist_id and is associated with a specific user (user_id). The table stores the names of the wishlists, enabling users to create lists of properties they're interested in. The foreign key constraint ensures that each wishlist is connected to a valid user in the Airbnb.user table. This table allows Airbnb users to organize and keep track of properties they want to save for later consideration.

```sql
--Table Airbnb.wishlist
DROP TABLE IF EXISTS Airbnb.wishlist CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.wishlist (
  wishlist_id SERIAL PRIMARY KEY NOT NULL,
  name VARCHAR(50) NOT NULL,
  user_id INT NOT NULL,
  CONSTRAINT fk_wishlist_user1
    FOREIGN KEY (user_id)
    REFERENCES Airbnb.user (user_id));
-- CREATE INDEX
CREATE INDEX IF NOT EXISTS fk_wishlist_user1_idx ON Airbnb.wishlist (user_id);
```

```sql
SELECT
    w.wishlist_id,
    w.name AS wishlist_name,
    u.first_name,
    u.last_name
FROM
    Airbnb.wishlist w
JOIN
    Airbnb.user u ON w.user_id = u.user_id
limit 5;
```

# WISHLIST PROPERTY MAPPING TABLE

```sql
SELECT p.property_id, p.title
FROM Airbnb.property p
JOIN Airbnb.wishlist_property_mapping wpm ON p.property_id = wpm.property_id
WHERE wpm.wishlist_id = 5;
```

| property_id [PK] integer | title character varying (50) |
|---|---|
| 15 | adipisci velit |
| 17 | quasi architecto |

```sql
-- Table Airbnb.wishlist_property_mapping
DROP TABLE IF EXISTS Airbnb.wishlist_property_mapping CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.wishlist_property_mapping (
  wishlist_id INT NOT NULL,
  property_id INT NOT NULL,
  PRIMARY KEY (wishlist_id, property_id),
  CONSTRAINT fk_wishlist_property_mapping_property
    FOREIGN KEY (property_id)
    REFERENCES Airbnb.property (property_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_wishlist_property_mapping_wishlist
    FOREIGN KEY (wishlist_id)
    REFERENCES Airbnb.wishlist (wishlist_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Creating indexes
CREATE INDEX IF NOT EXISTS idx_wishlist_property_mapping_property ON Airbnb.wishlist_property_mapping (property_id);
CREATE INDEX IF NOT EXISTS idx_wishlist_property_mapping_wishlist ON Airbnb.wishlist_property_mapping (wishlist_id);
```

This table establishes a connection between Wishlist and the properties that users add to their Wishlist on the Airbnb platform.

# BOOKING TABLE

| property_id integer | booking_count bigint |
|---|---|
| 15 | 4 |
| 3 | 2 |
| 8 | 2 |
| 5 | 2 |
| 13 | 2 |
| 7 | 2 |
| 18 | 2 |
| 11 | 1 |
| 17 | 1 |
| 12 | 1 |
| 6 | 1 |
| 1 | 1 |

This table records booking information for accommodations on the Airbnb platform. It stores essential details related to guest reservations and stays.

```
-- Create Table Airbnb.booking
-- Drop Table if it exists
DROP TABLE IF EXISTS Airbnb.booking CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.booking (
  booking_id SERIAL PRIMARY KEY NOT NULL,
  check_in DATE NOT NULL,
  check_out DATE NOT NULL,
  number_guests INT NOT NULL,
  property_id INT NOT NULL,
  user_id INT NOT NULL,
  CONSTRAINT fk_booking_property1
    FOREIGN KEY (property_id)
    REFERENCES Airbnb.property (property_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_booking_user1
    FOREIGN KEY (user_id)
    REFERENCES Airbnb.user (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
-- Creating indexes
CREATE INDEX IF NOT EXISTS fk_booking_property1_idx ON Airbnb.booking (property_id);
CREATE INDEX IF NOT EXISTS fk_booking_user1_idx ON Airbnb.booking (user_id);
```

```
SELECT property_id, COUNT(*) AS booking_count
FROM Airbnb.booking
GROUP BY property_id
ORDER BY booking_count DESC;
```

# PLACE TYPE TABLE

| property_id integer | title character varying (50) | place_type character varying (20) |
|---|---|---|
| 1 | Lorem ipsum | apartment |
| 2 | consectetur adipiscing | room |
| 6 | Quis autem vel | room |
| 7 | suscipit laboriosam | room |
| 10 | At vero eos | apartment |
| 13 | Nam libero | room |
| 14 | dolorum fuga | apartment |
| 15 | adipisci velit | room |
| 16 | qui ratione | room |
| 20 | dolorem eum | room |
| 21 | irure dolor | room |

This table stores the different types of accommodation that hosts can assign to their properties on the Airbnb platform. Each type describes the type of accommodation offered so that guests can better understand what to expect from the accommodation.

```
-- Create Table Airbnb.place_type
-- Drop Table if it exists
DROP TABLE IF EXISTS Airbnb.place_type CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.place_type (
  place_type_id SERIAL PRIMARY KEY NOT NULL,
  place_type VARCHAR(20) NOT NULL,
  CONSTRAINT chk_place_type CHECK (place_type IN ('hotel', 'apartment', 'room', 'bed'))
);
```

```
SELECT b.property_id, b.title, a.place_type
FROM Airbnb.place_type a
JOIN Airbnb.property b
ON a.place_type_id = b.property_id
WHERE place_type = 'room' OR place_type = 'apartment'
ORDER BY property_id
```

# ROOMS BEDS TABLE

| avg_bedrooms numeric | avg_beds numeric | avg_bathrooms numeric |
|---|---|---|
| 1.81 | 2.24 | 1.43 |

table serves to store information about the number of bedrooms, beds, and bathrooms within a property listing. It allows tracking the accommodation specifics, such as the bedroom count, bed count, and bathroom count, which are essential details for potential guests evaluating properties on Airbnb. Each record in this table represents a specific configuration of rooms, beds, and bathrooms within a property listing.

```
-- Table Airbnb.rooms_beds
DROP TABLE IF EXISTS Airbnb.rooms_beds CASCADE;

CREATE TABLE IF NOT EXISTS Airbnb.rooms_beds (
  rooms_beds_id SERIAL PRIMARY KEY NOT NULL,
  bedrooms INT DEFAULT NULL,
  beds INT NOT NULL,
  bathrooms INT NULL DEFAULT NULL);


  SELECT
    ROUND(AVG(bedrooms), 2) AS avg_bedrooms,
    ROUND(AVG(beds), 2) AS avg_beds,
    ROUND(AVG(bathrooms), 2) AS avg_bathrooms
  FROM Airbnb.rooms_beds;
```

# THANK YOU