```
/*Create Database, Schema (better structure) and load extensions*/

CREATE DATABASE rentals_db
    WITH
    OWNER = admin
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;

CREATE SCHEMA users
    AUTHORIZATION admin;

CREATE SCHEMA shared
    AUTHORIZATION admin;

CREATE SCHEMA inbox
    AUTHORIZATION admin;

CREATE SCHEMA rentals
    AUTHORIZATION admin;

CREATE SCHEMA hosts
    AUTHORIZATION admin;


CREATE EXTENSION pgcrypto;


/*Create tables for Countries, States, and Cities
Data source: https://github.com/dr5hn/countries-states-cities-
database*/

CREATE TABLE IF NOT EXISTS shared.currencies
(
     currency char(3) PRIMARY KEY,
     currency_symbol varchar(10)

);

ALTER TABLE IF EXISTS shared.currencies
    OWNER to admin;


CREATE TABLE shared.countries (
    iso2 character(2) NOT NULL PRIMARY KEY,
     iso3 character(3) NOT NULL,
     country_name varchar(100) NOT NULL,
     numeric_code smallint NOT NULL,
     phone_code smallint NOT NULL,
     currency character(3) DEFAULT NULL,
     region varchar(255) DEFAULT NULL,
     latitude decimal(10,8) DEFAULT NULL,
     longitude decimal(11,8) DEFAULT NULL,
     emoji varchar(191),
```

```
      CONSTRAINT currency FOREIGN KEY (currency) REFERENCES
shared.currencies (currency) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID

);

ALTER TABLE IF EXISTS shared.countries
    OWNER to admin;

CREATE TABLE shared.states (
      state_id varchar(100) NOT NULL,
      state_name varchar(100) NOT NULL,
      iso2_country character(2) NOT NULL,
      state_code character(5) NOT NULL,
      latitude decimal(10,8) DEFAULT NULL,
      longitude decimal(11,8) DEFAULT NULL,
      PRIMARY KEY (state_id),
      CONSTRAINT country_of_state FOREIGN KEY (iso2_country) REFERENCES
shared.countries(iso2) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
);

ALTER TABLE IF EXISTS shared.states
    OWNER to admin;


CREATE TABLE shared.cities (
      city_id varchar(100) NOT NULL,
      city_name varchar(100) NOT NULL,
      iso2_country character(2) NOT NULL,
      state_id varchar(100) NOT NULL,
      latitude decimal(10,8) DEFAULT NULL,
      longitude decimal(11,8) DEFAULT NULL,
      wiki_dataID varchar(10) DEFAULT NULL,
      PRIMARY KEY (city_id),
      CONSTRAINT country_of_city FOREIGN KEY (iso2_country) REFERENCES
shared.countries(iso2) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
      CONSTRAINT state_of_city FOREIGN KEY (state_id) REFERENCES
shared.states(state_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
);

ALTER TABLE IF EXISTS shared.cities
    OWNER to admin;
```

```
/*Create tables for Addresses
Data source: https://mockaroo.com*/


CREATE TABLE shared.addresses
(
    address_id serial NOT NULL,
    address_1 character varying(42) NOT NULL,
    address_2 character varying(42),
    city_id character varying(100) NOT NULL,
    state_id character varying(100) NOT NULL,
    iso2_country character(2) NOT NULL,
    zip_code integer NOT NULL,
    PRIMARY KEY (address_id),
    CONSTRAINT iso2_country FOREIGN KEY (iso2_country)
        REFERENCES shared.countries (iso2) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT state_id FOREIGN KEY (state_id)
        REFERENCES shared.states (state_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT city_id FOREIGN KEY (city_id)
        REFERENCES shared.cities (city_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
);

ALTER TABLE IF EXISTS shared.addresses
    OWNER to admin;


/*Create tables for Langagues
Data source: https://pastebin.com/raw/ppdMS687*/


CREATE TABLE shared.languages
(
    iso2_language character(2) NOT NULL,
    language_name varchar(100) NOT NULL,
    native_name varchar(100) NOT NULL,
    PRIMARY KEY (iso2_language)

);

ALTER TABLE IF EXISTS shared.languages
    OWNER to admin;
```

```
/*Create tables for User
Data source: https://github.com/dr5hn/countries-states-cities-
database*/

CREATE TABLE users.users
(
      user_id serial NOT NULL,
      first_name varchar(100) NOT NULL,
      last_name varchar(100) NOT NULL,
      email varchar(320) NOT NULL,
      pass_salt varchar(128) NOT NULL,
      pass_hash varchar(128) NOT NULL,
      area_code character(2) NOT NULL,
      phone bigint NOT NULL,
      verified_phone boolean NOT NULL,
      photo bytea,
      birth_date date,
      gender character(1),
      government_id varchar(30),
      active_user boolean NOT NULL DEFAULT TRUE,
      created_date date DEFAULT current_date,
      last_updated date DEFAULT current_date,
      address_id integer DEFAULT NULL UNIQUE,
      host boolean NOT NULL DEFAULT FALSE,
      PRIMARY KEY (user_id),
      CONSTRAINT id_underaged_users CHECK (birth_date < (current_date -
interval '18' year)) NOT VALID,
      CONSTRAINT check_dates1 CHECK (created_date > birth_date) NOT
VALID,
      CONSTRAINT check_dates2 CHECK (created_date <= last_updated) NOT
VALID,
      CONSTRAINT gender_check CHECK (gender in ('M', 'F')) NOT VALID,
      CONSTRAINT address_id FOREIGN KEY (address_id) REFERENCES
shared.addresses(address_id) MATCH SIMPLE
         ON UPDATE CASCADE
         NOT VALID,
      CONSTRAINT area_code FOREIGN KEY (area_code) REFERENCES
shared.countries (iso2) MATCH SIMPLE
         ON UPDATE CASCADE
         NOT VALID

);

ALTER TABLE IF EXISTS users.users
    OWNER to admin;


/*Create tables for Wallet*/

CREATE TABLE users.wallet
(
    stripe_customer_id varchar(128) NOT NULL,
```

```
    user_id bigint NOT NULL UNIQUE,
    billing_address_id bigint NOT NULL UNIQUE,
    PRIMARY KEY (stripe_customer_id),
    CONSTRAINT user_id FOREIGN KEY (user_id)
        REFERENCES users.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT billing_address FOREIGN KEY (billing_address_id)
        REFERENCES shared.addresses (address_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID

);

ALTER TABLE IF EXISTS users.wallet
    OWNER to admin;



/*Create tables for Emergency Contact
Data source: https://mockaroo.com*/

CREATE TABLE users.emergency_contact
(
    contact_id serial NOT NULL,
    contact_name varchar(255) NOT NULL,
    relationship varchar(100) NOT NULL,
    preferred_language varchar(100) NOT NULL,
    email varchar(320),
    area_code character(2) NOT NULL,
    phone bigint NOT NULL,
     user_id bigint NOT NULL,
    PRIMARY KEY (contact_id),
    CONSTRAINT preferred_language FOREIGN KEY (preferred_language)
        REFERENCES shared.languages (iso2_language) MATCH SIMPLE
        ON UPDATE CASCADE
        NOT VALID,
    CONSTRAINT country_area_code FOREIGN KEY (area_code)
        REFERENCES shared.countries (iso2) MATCH SIMPLE
        ON UPDATE CASCADE
        NOT VALID,
    CONSTRAINT user_id FOREIGN KEY (user_id)
        REFERENCES users.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
);

ALTER TABLE IF EXISTS users.emergency_contact
    OWNER to admin;
```

```
/*Create tables for Reward Points*/

CREATE TABLE users.reward_points
(
    points_id serial NOT NULL,
    user_id bigint NOT NULL,
     date_transaction date NOT NULL,
    points integer NOT NULL,
    PRIMARY KEY (points_id),
    CONSTRAINT user_id FOREIGN KEY (user_id)
        REFERENCES users.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
);

ALTER TABLE IF EXISTS users.reward_points
    OWNER to admin;

/*Create tables for Hosts
Data source: https://mockaroo.com*/


CREATE TABLE hosts.hosts
(
     host_id serial NOT NULL,
     user_id bigint NOT NULL UNIQUE,
     main_language character(2) NOT NULL,
     secondary_language character(2),
     read_rate integer DEFAULT NULL,
     response_time interval DEFAULT NULL,
     verified boolean NOT NULL DEFAULT FALSE,
     about text,
     PRIMARY KEY (host_id),
     CONSTRAINT user_id FOREIGN KEY (user_id) REFERENCES
users.users(user_id) MATCH SIMPLE
        ON UPDATE CASCADE
           ON DELETE CASCADE
        NOT VALID,
     CONSTRAINT main_language FOREIGN KEY (main_language) REFERENCES
shared.languages(iso2_language) MATCH SIMPLE
        ON UPDATE NO ACTION
           ON DELETE NO ACTION
        NOT VALID,
     CONSTRAINT secondary_language FOREIGN KEY (secondary_language)
REFERENCES shared.languages(iso2_language) MATCH SIMPLE
        ON UPDATE NO ACTION
           ON DELETE NO ACTION
        NOT VALID

);

ALTER TABLE IF EXISTS hosts.hosts
```

```
    OWNER to admin;


/*-Create tables for Chat Session
Data source: https://mockaroo.com*/

CREATE TABLE inbox.chat_sessions
(
      user_traveler_id bigint NOT NULL,
      user_host_id bigint NOT NULL,
      subject varchar(64),
      status varchar(8) DEFAULT NULL,
      time_started timestamp DEFAULT CURRENT_TIMESTAMP,
      PRIMARY KEY (user_traveler_id, user_host_id),
      CONSTRAINT user_traveler_id FOREIGN KEY (user_traveler_id)
REFERENCES users.users(user_id) MATCH SIMPLE
         ON UPDATE NO ACTION
            ON DELETE NO ACTION
         NOT VALID,
      CONSTRAINT user_host_id FOREIGN KEY (user_host_id) REFERENCES
hosts.hosts(user_id) MATCH SIMPLE
         ON UPDATE NO ACTION
            ON DELETE NO ACTION
         NOT VALID,
      CONSTRAINT status_options CHECK (status in ('unread',
'active','archived')) NOT VALID,
      CONSTRAINT traveler_and_host CHECK (user_host_id !=
user_traveler_id) NOT VALID

);

ALTER TABLE IF EXISTS inbox.chat_sessions
    OWNER to admin;

/*-Create tables for Messages
Data source: https://mockaroo.com*/


CREATE TABLE inbox.messages
(
      message_id serial NOT NULL,
      user_traveler_id bigint NOT NULL,
      user_host_id bigint NOT NULL,
      sender bigint NOT NULL,
      message_content text NOT NULL,
      read_status boolean DEFAULT FALSE,
      sent_at timestamp DEFAULT CURRENT_TIMESTAMP,
      PRIMARY KEY (message_id),
      CONSTRAINT traveler_host_key FOREIGN KEY (user_traveler_id,
user_host_id) REFERENCES inbox.chat_sessions
         ON UPDATE CASCADE
            ON DELETE CASCADE
         NOT VALID,
```

```
CONSTRAINT sender FOREIGN KEY (sender) REFERENCES users.users(user_id)
MATCH SIMPLE

        ON UPDATE CASCADE
           ON DELETE CASCADE
        NOT VALID,

     CONSTRAINT check_sender CHECK (sender = user_traveler_id or
sender = user_host_id) NOT VALID

);

ALTER TABLE IF EXISTS inbox.messages
    OWNER to admin;

/*-Create Triggers for Messages Read Rate*/

CREATE OR REPLACE FUNCTION inbox.calculate_read_rate() RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
IF (NEW.sender != NEW.user_host_id) THEN

     UPDATE hosts.hosts
     SET read_rate =

     (SELECT (
     ((SELECT count(inbox.messages.read_status)
     FROM inbox.messages
     WHERE inbox.messages.user_host_id = NEW.user_host_id
     AND inbox.messages.sender != NEW.user_host_id
     AND inbox.messages.read_status = 'true') * 100)
     /
     (SELECT count(inbox.messages.read_status)
     FROM inbox.messages
     WHERE inbox.messages.user_host_id = NEW.user_host_id
     AND inbox.messages.sender != NEW.user_host_id
     ) ) AS read_rate)

     WHERE user_id = NEW.user_host_id
     ;


END IF;
RETURN NEW;
END
$$

CREATE TRIGGER host_average_read_rate AFTER INSERT ON inbox.messages
FOR EACH ROW
EXECUTE PROCEDURE inbox.calculate_read_rate()
;
```

```
CREATE TRIGGER host_average_read_rate_update AFTER UPDATE ON
inbox.messages
FOR EACH ROW
EXECUTE PROCEDURE inbox.calculate_read_rate()
;
/*-Create Triggers for Messages Response Time*/

CREATE OR REPLACE FUNCTION inbox.calculate_response_time() RETURNS
TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN

      UPDATE hosts.hosts
      SET response_time =

            (WITH
            thread_starts AS (
            SELECT message_id, user_traveler_id, user_host_id, sender,
sent_at,
            COALESCE(LAG((user_traveler_id, user_host_id, sender))
            OVER ordered != (user_traveler_id, user_host_id, sender),
true) AS thread_start
            FROM inbox.messages
            WHERE user_host_id = NEW.user_host_id
            WINDOW ordered AS (PARTITION BY user_traveler_id,
user_host_id ORDER BY sent_at
            )),

            thread_responses AS(
            SELECT user_traveler_id, user_host_id, sender, sent_at,
            LEAD(sent_at) OVER ordered AS responded_at,
            COUNT(*) OVER unordered AS threads
            FROM thread_starts
            WHERE thread_start
            WINDOW ordered AS (PARTITION BY user_traveler_id,
user_host_id ORDER BY sent_at),
            unordered AS (PARTITION BY user_traveler_id, user_host_id))

            SELECT
            AVG(COALESCE(responded_at, CURRENT_TIMESTAMP) - sent_at) AS
avg_response_time
            FROM thread_responses
            WHERE sender != user_host_id AND (responded_at IS NOT NULL
OR threads = 1)
            GROUP BY user_host_id)

      WHERE user_id = NEW.user_host_id;

RETURN NEW;
END
```

```
$$

CREATE TRIGGER host_response_time AFTER INSERT ON inbox.messages
FOR EACH ROW
EXECUTE PROCEDURE inbox.calculate_response_time()
;

/*-Create tables for Amenitites
Data source: airbnb.com*/

CREATE TABLE rentals. amenities
(
      amenity varchar(128) NOT NULL PRIMARY KEY,
      category varchar(128) NOT NULL,
      description varchar(255)

);

ALTER TABLE IF EXISTS rentals. amenities
    OWNER to admin;

/*-Create tables for Cancellation Policies
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.cancellation_policies
(
    policy_title varchar (128) PRIMARY KEY,
    description varchar (500) NOT NULL,
    valid_for_long_term_stay boolean NOT NULL
);


ALTER TABLE IF EXISTS rentals.cancellation_policies
    OWNER to admin;

/*-Create tables for Preparation Time Prior Arrival.
The nights before and after each reservation are blocked according to
the preparation time chosen.
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.preparation_time
(
    preparation_time interval PRIMARY KEY,
    description varchar (500) NOT NULL

);


ALTER TABLE IF EXISTS rentals.preparation_time
    OWNER to admin;
```

```
/*-Create tables for Advance Notice.
Time that the host requires as notice prior reservations by guests
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.advance_notice
(
    advance_notice interval PRIMARY KEY,
    description varchar (500) NOT NULL

);


ALTER TABLE IF EXISTS rentals.advance_notice
    OWNER to admin;


/*-Create tables for Availability Window.
Time that host allows guests to make reservation in the future.
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.availability_window
(
    availability_window interval PRIMARY KEY,
    description varchar (500) NOT NULL

);


ALTER TABLE IF EXISTS rentals.availability_window
    OWNER to admin;


/*-Create tables for Spaces.
Spaces available in the rentals
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.spaces
(
    spaces varchar(128) PRIMARY KEY
);


ALTER TABLE IF EXISTS rentals.spaces
    OWNER to admin;


/*-Create tables for Shared Options.
Group of people spaces are shared.
Data source: airbnb.com*/
```

```
CREATE TABLE IF NOT EXISTS rentals.share_options
(
    sharing_groups varchar(128) PRIMARY KEY
);


ALTER TABLE IF EXISTS rentals.share_options
    OWNER to admin;


/*-Create tables for Relevant Time.
Time periods for check ins, check outs and other rental operations.
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.relevant_time
(
    relevant_time time PRIMARY KEY
);


ALTER TABLE IF EXISTS rentals.relevant_time
    OWNER to admin;

/*-Create tables for Property Categories.
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.property_categories
(
    product_categories varchar(128) PRIMARY KEY,
      description varchar(500)
);


ALTER TABLE IF EXISTS rentals.property_categories
    OWNER to admin;


/*-Create tables for Property Type.
Data source: airbnb.com*/


CREATE TABLE IF NOT EXISTS rentals.property_type
(
      property_id serial PRIMARY KEY,
    property_type varchar(128),
      property_category varchar(128) NOT NULL,
      description varchar(500),
      CONSTRAINT property_category FOREIGN KEY(property_category)
REFERENCES rentals.property_categories(product_categories)
```

```
);

ALTER TABLE IF EXISTS rentals.property_type
    OWNER to admin;


/*-Create tables for Rules.
Data source: airbnb.com*/



CREATE TABLE IF NOT EXISTS rentals.rules
(
     rule_title varchar(128) PRIMARY KEY,
    rule_owner bigint,
     description varchar(500),
     CONSTRAINT rule_owner FOREIGN KEY(rule_owner) REFERENCES
hosts.hosts(host_id)
     ON DELETE CASCADE
     ON UPDATE CASCADE

);

ALTER TABLE IF EXISTS rentals.rules
    OWNER to admin;


/*-Create tables for Rentals.*/


CREATE TABLE rentals.rentals
(
     rental_id serial PRIMARY KEY,
     host_id bigint NOT NULL,
     active_rental boolean NOT NULL DEFAULT TRUE,
     title varchar(128) NOT NULL,
     description varchar(500) NOT NULL,
     allowed_guests smallint NOT NULL,
     address_id bigint NOT NULL,
     price money NOT NULL,
     listing_currency char(3) NOT NULL,
     weekly_discount decimal(5,4) DEFAULT NULL,
     monthly_discount decimal(5,4) DEFAULT NULL,
     early_bird_discount decimal(5,4) DEFAULT NULL,
     last_minute_discount decimal(5,4) DEFAULT NULL,
     cleaning_fee money DEFAULT NULL ,
     extra_guest_daily_fee money DEFAULT NULL,
     pet_fee money DEFAULT NULL,
     min_stay interval NOT NULL DEFAULT '1 Day',
     max_stay interval NOT NULL DEFAULT '5 Years',
     check_in_window time NOT NULL DEFAULT '12:00:00',
     check_out_window time NOT NULL DEFAULT '10:00:00',
     cancellation_policy varchar(128) NOT NULL DEFAULT 'Flexible',
```

```
      guest_id_required boolean NOT NULL DEFAULT TRUE,
      instant_booking boolean NOT NULL DEFAULT TRUE CHECK(CASE WHEN
instant_booking = 'TRUE' THEN guest_id_required = 'TRUE' END),
      property_id bigint NOT NULL,
      listing_type varchar(15) NOT NULL DEFAULT 'Entire Place' CHECK
(listing_type in ('Entire Place', 'Private Room', 'Shared Room')),
      rating numeric(3,2),
      creation_date date NOT NULL DEFAULT CURRENT_DATE,
      last_updated timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
      CONSTRAINT host_id FOREIGN KEY (host_id) REFERENCES
hosts.hosts(host_id) MATCH SIMPLE
            ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID,
      CONSTRAINT address_id FOREIGN KEY (address_id) REFERENCES
shared.addresses(address_id) MATCH SIMPLE
            ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID,
      CONSTRAINT listing_currency FOREIGN KEY (listing_currency)
REFERENCES shared.currencies(currency) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT check_in_window FOREIGN KEY (check_in_window)
REFERENCES rentals.relevant_time(relevant_time) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT check_out_window FOREIGN KEY (check_out_window)
REFERENCES rentals.relevant_time(relevant_time) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT cancelation_policy FOREIGN KEY (cancellation_policy)
REFERENCES rentals.cancellation_policies(policy_title) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT property_id FOREIGN KEY (property_id) REFERENCES
rentals.property_type(property_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID


);

ALTER TABLE IF EXISTS rentals.rentals
    OWNER to admin;


/*-Create tables for Calendar Availability.*/
```

```
CREATE TABLE rentals.calendar_availability
(
      rental_id bigint PRIMARY KEY,
      availability_window interval NOT NULL DEFAULT ‘5 years’,
      preparation_time interval NOT NULL DEFAULT ‘0 days,
      advance_notice interval NOT NULL DEFAULT ‘0 days,
      same_day_booking_max_time time DEFAULT ‘13:00:00’,
      CONSTRAINT shared_check CHECK(CASE WHEN advance_notice !=
'00:00:00' THEN same_day_booking_max_time IS NULL ELSE
same_day_booking_max_time IS NOT NULL END) NOT VALID,
      CONSTRAINT preparation_time FOREIGN KEY (preparation_time)
REFERENCES rentals.preparation_time(preparation_time) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT availability_window FOREIGN KEY (availability_window)
REFERENCES rentals.availability_window(availability_window) MATCH
SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT advance_notice FOREIGN KEY (advance_notice) REFERENCES
rentals.advance_notice(advance_notice) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT same_day_booking_max_time FOREIGN KEY
(same_day_booking_max_time) REFERENCES
rentals.relevant_time(relevant_time) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
            ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID



);

ALTER TABLE IF EXISTS rentals.calendar_availability
    OWNER to admin;

/*-Create tables for Combinations of Amenities for each Rantal.*/



CREATE TABLE rentals.rentals_and_amenities
(
```

```
      rental_id bigint,
      amenity varchar(128),
      PRIMARY KEY(rental_id, amenity),

      CONSTRAINT amenity FOREIGN KEY (amenity) REFERENCES
rentals.amenities(amenity) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
            ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.rentals_and_amenities
    OWNER to admin;

/*-Create tables for Spacaes available for each Rental.*/


CREATE TABLE rentals.rentals_and_spaces
(
      rental_id bigint,
      spaces varchar(128),
      quantity smallint,
      shared_spaces boolean NOT NULL DEFAULT 'false',
      shared_with varchar(128) DEFAULT NULL,
      PRIMARY KEY(rental_id, spaces),
      CONSTRAINT shared_check CHECK(CASE WHEN shared_spaces = 'false'
THEN shared_with IS NULL ELSE shared_with IS NOT NULL END) NOT VALID,
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
            ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.rentals_and_spaces
    OWNER to admin;

/*-Create tables of rules applying to each rental.*/


CREATE TABLE rentals.rentals_and_rules
(
      rule_id serial PRIMARY KEY,
      rental_id bigint NOT NULL,
      rule_title varchar(128) NOT NULL,
      active boolean NOT NULL DEFAULT TRUE,
      CONSTRAINT rule_title FOREIGN KEY (rule_title) REFERENCES
rentals.rules(rule_title) MATCH SIMPLE
```

```
           ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID,
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
           ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.rentals_and_rules
    OWNER to admin;

CREATE UNIQUE INDEX rental_rule ON rentals.rentals_and_rules
(rental_id, rule_title);



/*-Create tables for Custom daily pricing for each rental.*/


CREATE TABLE rentals.custom_pricing
(
      rental_id bigint,
      scheduled_date date,
      discounted_price money,
      continuous_intervals interval,
      PRIMARY KEY(rental_id, scheduled_date),
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
          ON DELETE CASCADE
         ON UPDATE CASCADE
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.custom_pricing
    OWNER to admin;


/*-Create tables for Reservations.*/

CREATE TABLE rentals.reservations
(
      reservation_id serial PRIMARY KEY,
      rental_id bigint NOT NULL,
      guest_id bigint NOT NULL,
      daily_price money NOT NULL,
      reservation_duration interval NOT NULL ,
      reservation_made_on date NOT NULL DEFAULT CURRENT_DATE CHECK
(reservation_starts > reservation_made_on),
      reservation_starts date NOT NULL,
      reservation_ends date NOT NULL CHECK (reservation_ends >
reservation_starts),
```

```
      last_updated timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
      confirmed_by_host boolean NOT NULL DEFAULT 'FALSE',
      check_in boolean NOT NULL CHECK(CASE WHEN check_in = 'TRUE' THEN
      confirmed_by_host = 'TRUE' END),
      check_out boolean NOT NULL CHECK(CASE WHEN check_in = 'FALSE'
THEN check_out = 'FALSE' END),
      total_price money NOT NULL,
      number_of_extra_guests smallint NOT NULL,
      applied_fees_and_discounts text[],
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.reservations
    OWNER to admin;

/*-Create tables for Reviews.*/


CREATE TABLE rentals.reviews
(
      review_id serial PRIMARY KEY,
      reservation_id bigint NOT NULL,
      rental_id bigint NOT NULL,
      guest_id bigint NOT NULL,
      comment_text varchar(500) NOT NULL,
      rating numeric(3,2) NOT NULL CHECK (rating >= 0 and rating <= 5),
      CONSTRAINT reservation_id FOREIGN KEY (reservation_id) REFERENCES
rentals.reservations(reservation_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT guest_id FOREIGN KEY (guest_id) REFERENCES
users.users(user_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID,
      CONSTRAINT rental_id FOREIGN KEY (rental_id) REFERENCES
rentals.rentals(rental_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID
      );

ALTER TABLE IF EXISTS rentals.reviews
    OWNER to admin;
```

```
/*-Create tables for Historization of Rule. This table will allow
hosts, guests and administrators to check which rules where active for
each reservation.*/

CREATE TABLE rentals.rules_historization
(
      historization_id serial PRIMARY KEY,
      rule_id bigint NOT NULL,
      updated_on timestamp NOT NULL,
      new_active_status boolean NOT NULL DEFAULT TRUE,
      CONSTRAINT rule_id FOREIGN KEY (rule_id) REFERENCES
rentals.rentals_and_rules(rule_id) MATCH SIMPLE
            ON DELETE NO ACTION
         ON UPDATE NO ACTION
         NOT VALID

      );

ALTER TABLE IF EXISTS rentals.rules_historization
    OWNER to admin;

/*-Create Functions and Triggers for Historization.*/


CREATE OR REPLACE FUNCTION rentals.historized_rules() RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN

      INSERT INTO rentals.rules_historization(
      rule_id, updated_on, new_active_status)
      VALUES (NEW.rule_id, CURRENT_TIMESTAMP, NEW.active);

RETURN NEW;
END
$$

CREATE TRIGGER historization_of_rules AFTER INSERT ON
rentals.rentals_and_rules
FOR EACH ROW
EXECUTE PROCEDURE rentals.historized_rules()
;

CREATE TRIGGER historization_of_rules_on_update AFTER UPDATE ON
rentals.rentals_and_rules
FOR EACH ROW
EXECUTE PROCEDURE rentals.historized_rules()
;

CREATE TRIGGER historization_of_rules_on_delete BEFORE DELETE ON
rentals.rentals_and_rules
FOR EACH ROW
```

```
EXECUTE PROCEDURE rentals.historized_rules()
;


/*-Create Functions and Triggers for Calculating Ratings.*/

CREATE OR REPLACE FUNCTION rentals.calculate_rating() RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN

      UPDATE rentals.rentals
      SET rating =

      (SELECT avg(rentals.reviews.rating)
      FROM rentals.reviews
      WHERE rentals.reviews.rental_id = NEW.rental_id)

      WHERE rental_id = NEW.rental_id
      ;

RETURN NEW;
END
$$

CREATE TRIGGER calculate_rental_rating AFTER INSERT ON rentals.reviews
FOR EACH ROW
EXECUTE PROCEDURE rentals.calculate_rating()
;


/*-Create Functions and Triggers for Checkign Some Valid Parameters
for Reservations.*/

CREATE OR REPLACE FUNCTION rentals.check_valid_reservation() RETURNS
TRIGGER
LANGUAGE plpgsql
AS
$$
DECLARE
      min_stayV interval;
      max_stayV interval;
BEGIN

      SELECT min_stay INTO min_stayV FROM rentals.rentals WHERE
rental_id = NEW.rental_id;
      SELECT max_stay INTO max_stayV FROM rentals.rentals WHERE
rental_id = NEW.rental_id;
      IF (NEW.reservation_duration < min_stayV OR
NEW.reservation_duration > max_stayV) THEN
      RAISE EXCEPTION 'The reservation exceeds duration limits ';
      END IF;
```

```
RETURN NEW;
END
$$

CREATE TRIGGER check_valid_reservation_duration BEFORE INSERT ON
rentals.reservations
FOR EACH ROW
EXECUTE PROCEDURE rentals.check_valid_reservation()
```

ROLES AND USERS FOR THE DATABASE.

For deployment it is highly recommended not to use the admin user to connect to the database since this user is a superuser and can jeopardize the database's security. A user named user_01 has been created which has limited privileges. This are inherited from a role named user_group. This role can be given to other users if several connections are needed.

```
CREATE ROLE users_group WITH
      NOLOGIN
      NOSUPERUSER
      NOCREATEDB
      NOCREATEROLE
      NOINHERIT
      NOREPLICATION
      CONNECTION LIMIT -1;

GRANT INSERT, UPDATE, SELECT ON ALL TABLES IN SCHEMA hosts TO
users_group;
GRANT INSERT, UPDATE, SELECT ON ALL TABLES IN SCHEMA inbox TO
users_group;
GRANT INSERT, UPDATE, SELECT ON ALL TABLES IN SCHEMA rentals TO
users_group;
GRANT INSERT, UPDATE, SELECT ON ALL TABLES IN SCHEMA users TO
users_group;
GRANT SELECT ON ALL TABLES IN SCHEMA shared TO users_group;


CREATE ROLE user01 WITH
      LOGIN
      NOSUPERUSER
      NOCREATEDB
      NOCREATEROLE
      INHERIT
      NOREPLICATION
      CONNECTION LIMIT -1
      PASSWORD 'xxxxxx';

GRANT users_group TO user01;
```