My SQL Project

The objective of the project

> To create a platform for users who want to read their horoscopes and find a trustworthy fortune teller. Also, assist the fortune teller and the customer in using the appointment, chatting, and payment system. The income from this project comes from commissions paid to fortune tellers who have customers to horoscope with.

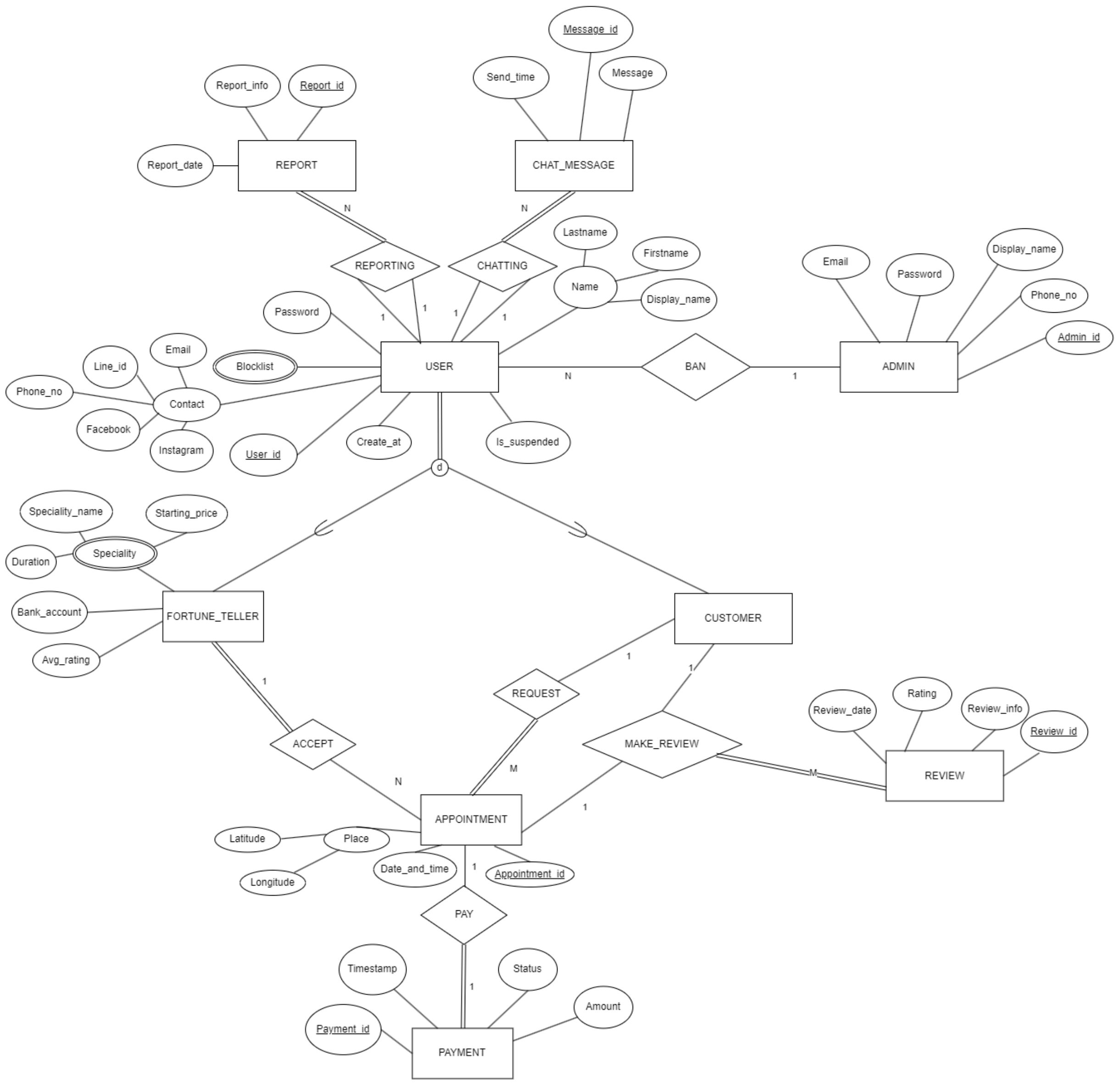
I use PostgreSQL for this project.

This is my code for create table.

```
-- This script was generated by the ERD tool in pgAdmin 4.
-- Please log an issue at https://redmine.postgresql.org/projects/pgadmin4/issues/new if you find any bugs, including reproduction steps.
BEGIN;

CREATE TYPE Usertype AS ENUM('Customer','FortuneTeller');

CREATE TABLE IF NOT EXISTS public.USER
(
    User_id character(10) NOT NULL,
    Firstname character varying(30) NOT NULL,
    Lastname character varying(30) NOT NULL,
    Email character varying(255) NOT NULL UNIQUE,
    Password character varying(25) NOT NULL,
    Phone_no character(10),
    Display_name character varying(10) NOT NULL,
    Is_suspended boolean NOT NULL,
    Create_at date,
    Line_id character varying(255),
    Facebook character varying(255),
    Instagram character varying(255),
    User_type Usertype NOT NULL,
    Banned_by_admin_id character(10),
    PRIMARY KEY (User_id)
);

CREATE TABLE IF NOT EXISTS public.BLOCKLIST
(
    User_id character(10) NOT NULL,
    Blocked_user_id character(10) NOT NULL,
    PRIMARY KEY (User_id, Blocked_user_id)
);

CREATE TABLE IF NOT EXISTS public.CUSTOMER
(
    Customer_id character(10) NOT NULL,
    PRIMARY KEY (Customer_id)
);

CREATE TABLE IF NOT EXISTS public.ADMIN
(
    Admin_id character(10) NOT NULL,
    Phone_no character(10),
    Display_name character varying(10) NOT NULL,
    Email character varying(255) NOT NULL UNIQUE,
    Password character varying(25) NOT NULL,
    PRIMARY KEY (Admin_id)
);

CREATE TABLE IF NOT EXISTS public.FORTUNE_TELLER
(
    Fortune_teller_id character(10) NOT NULL,
    Bank_account character varying(10),
    Avg_rating numeric,
    PRIMARY KEY (Fortune_teller_id)
);

CREATE TABLE IF NOT EXISTS public.SPECIALITY
(
    Fortune_teller_id character(10) NOT NULL,
    Speciality_name character varying(45) NOT NULL,
    Starting_price numeric,
    Duration numeric,
    PRIMARY KEY (Fortune_teller_id, Speciality_name)
);

CREATE TABLE IF NOT EXISTS public.APPOINTMENT
(
    Appointment_id character(10) NOT NULL,
  Customer_id character(10) NOT NULL,
    Date_and_time timestamp without time zone NOT NULL,
    Latitude numeric NOT NULL,
    Longitude numeric NOT NULL,
    Fortune_teller_id character(10),
    PRIMARY KEY (Appointment_id,Customer_id)
);

CREATE TYPE STAT AS ENUM('Pending','Paid' ,'Failed');

CREATE TABLE IF NOT EXISTS public.PAYMENT
(
    Payment_id character(10) NOT NULL,
    Appointment_id character(10) NOT NULL,
  Customer_id character(10) NOT NULL,
    Amount numeric NOT NULL,
    Status STAT NOT NULL,
    Timestamp timestamp without time zone,
    PRIMARY KEY (Payment_id, Appointment_id,Customer_id)
);

CREATE TABLE IF NOT EXISTS public.CHAT_MESSAGE
(
    Message_id character(10) NOT NULL,
    Message character varying(256) NOT NULL,
    Sent_time timestamp without time zone NOT NULL,
    PRIMARY KEY (Message_id)
);

CREATE TABLE IF NOT EXISTS public.CHATTING
(
    Message_id character(10) NOT NULL,
    Sender_id character(10) NOT NULL,
    Receiver_id character(10) NOT NULL,
    PRIMARY KEY (Message_id, Sender_id, Receiver_id)
);

CREATE TABLE IF NOT EXISTS public.REPORT
(
    Report_id character(10) NOT NULL,
    Report_info character varying(256),
    Report_date timestamp without time zone NOT NULL,
    PRIMARY KEY (Report_id)
);

CREATE TABLE IF NOT EXISTS public.REPORTING
(
    Report_id character(10) NOT NULL,
    Reporter_id character(10) NOT NULL,
    Reported_id character(10) NOT NULL,
    PRIMARY KEY (Report_id, Reporter_id, Reported_id)
);
```

```
CREATE TABLE IF NOT EXISTS public.MAKE_REVIEW
(
    Appointment_id character(10) NOT NULL,
    Review_id character(10) NOT NULL,
    Customer_id character(10) NOT NULL,
    PRIMARY KEY (Appointment_id, Review_id, Customer_id)
);

CREATE TABLE IF NOT EXISTS public.REVIEW
(
    Review_id character(10) NOT NULL,
    Rating numeric,
    Review_info character varying(256) NOT NULL,
    Review_date timestamp without time zone NOT NULL,
    PRIMARY KEY (Review_id)
);

ALTER TABLE IF EXISTS public.USER
    ADD FOREIGN KEY (Banned_by_admin_id)
    REFERENCES public.ADMIN (Admin_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.BLOCKLIST
    ADD FOREIGN KEY (User_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.CUSTOMER
    ADD FOREIGN KEY (Customer_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.FORTUNE_TELLER
    ADD FOREIGN KEY (Fortune_teller_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.SPECIALITY
    ADD FOREIGN KEY (Fortune_teller_id)
    REFERENCES public.FORTUNE_TELLER (Fortune_teller_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.APPOINTMENT
    ADD FOREIGN KEY (Fortune_teller_id)
    REFERENCES public.FORTUNE_TELLER (Fortune_teller_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.APPOINTMENT
    ADD FOREIGN KEY (Customer_id)
    REFERENCES public.CUSTOMER (Customer_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.PAYMENT
    ADD FOREIGN KEY (Appointment_id, Customer_id)
    REFERENCES public.APPOINTMENT (Appointment_id, Customer_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.CHATTING
    ADD FOREIGN KEY (Message_id)
    REFERENCES public.CHAT_MESSAGE (Message_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.CHATTING
    ADD FOREIGN KEY (Sender_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.CHATTING
    ADD FOREIGN KEY (Receiver_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.REPORTING
    ADD FOREIGN KEY (Reporter_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.REPORTING
    ADD FOREIGN KEY (Reported_id)
    REFERENCES public.USER (User_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.REPORTING
    ADD FOREIGN KEY (Report_id)
    REFERENCES public.REPORT (Report_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.MAKE_REVIEW
    ADD FOREIGN KEY (Appointment_id, Customer_id)
    REFERENCES public.APPOINTMENT (Appointment_id, Customer_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

ALTER TABLE IF EXISTS public.MAKE_REVIEW
    ADD FOREIGN KEY (Review_id)
    REFERENCES public.REVIEW (Review_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

END
```

This is my ER Diagram



This is an interesting insight from my database.

1. Query data to find fortune tellers that create income from the platform sort by descending.

```
SELECT
    Fortune_teller_id,
    Firstname,
    Lastname,
    Display_name,
    SUM(Amount) AS Total_Amount
FROM
    public.User
    INNER JOIN Fortune_teller ON user_id = fortune_teller_id
    NATURAL JOIN Payment
    NATURAL JOIN Appointment
```

```
WHERE
    Status = 'Paid'
GROUP BY
    Fortune_teller_id,
    Firstname,
    Lastname,
    Display_name
ORDER BY
    Total_Amount DESC;
```

2. Query data to find fortune tellers that have a rating of more than one and have a horoscope of more than one sort by rating descending.

```
SELECT
    F.Fortune_teller_id,
    U.Firstname,
    U.Lastname,
    U.Display_name,
    AVG(F.Avg_rating) AS Avg_rating,
    COUNT(*) AS Number_of_Appointment
FROM
    public.User U
    INNER JOIN Fortune_teller F ON U.user_id = F.fortune_teller_id
    INNER JOIN Appointment A ON A.fortune_teller_id = F.fortune_teller_id
WHERE
    F.Avg_rating > 3
GROUP BY
    F.Fortune_teller_id,
    U.Firstname,
    U.Lastname,
    U.Display_name
HAVING
    COUNT(*) > 1
ORDER BY
    Avg_rating DESC;
```

3. Query data to find customers who used the service in 2023. Sort by total spending in 2023 and average spending per time.

```
SELECT
    Customer_id,
    Firstname,
    Lastname,
    Display_name,
    SUM(Amount) AS Total_Amount,
  COUNT(*) AS Total_Appointment,
  CAST(SUM(Amount) / COUNT(*) AS DOUBLE PRECISION) AS Avg_spend_per_appointment
FROM
    public.User
    INNER JOIN Customer ON user_id = Customer_id
  NATURAL JOIN Appointment
  NATURAL JOIN Payment
WHERE
    Status = 'Paid'
    AND EXTRACT(YEAR FROM Date_and_time) = 2023
GROUP BY
    Customer_id,
    Firstname,
    Lastname,
    Display_name
ORDER BY
    Total_Amount,Avg_spend_per_appointment DESC;
```