CREATION OF AIRBNB DATABASE

NAME : ZONUNMAWIA

MATRICULATION : 9200610

TUTOR : MUSHARAF DOGER

COURSE : PROJECT: BUILD A DATA MART IN SQL

COURSE OF STUDY: BSC COMPUTER SCIENCE

LOCATION

Store the different location of listing, user address and host address within the system. The table is crucial for providing all location related information for all the user including administrator.

```
CREATE TABLE LOCATION(
    location_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    location_name VARCHAR(30) NOT NULL,
    street VARCHAR(30),
    city VARCHAR(30) NOT NULL,
    country VARCHAR(30) NOT NULL,
    pincode INT NOT NULL
);
```

location_id [PK] integer	location_name character varying (30)	street character varying (30)	city character varying (30)	country character varying (30)	pincode integer
1	Burj Kalifa1	Join Avenue Street1	New York1	India1	123456
2	Burj Kalifa2	Join Avenue Street2	New York2	India2	113456
3	Burj Kalifa3	Join Avenue Street3	New York3	India3	126456
4	Burj Kalifa4	Join Avenue Street4	New York4	India4	223456
5	Burj Kalifa5	Join Avenue Street5	New York5	India5	323456
6	Burj Kalifa6	Join Avenue Street6	New York6	India6	111456
7	Burj Kalifa7	Join Avenue Street7	New York7	India7	123246
8	Burj Kalifa8	Join Avenue Street8	New York8	India8	121236
9	Burj Kalifa9	Join Avenue Street9	New York9	India9	123426
10	Burj Kalifa10	Join Avenue Street10	New York10	India10	123458
11	Burj Kalifa11	Join Avenue Street11	New York11	India11	123459
12	Burj Kalifa12	Join Avenue Street12	New York12	India12	223458
13	Burj Kalifa13	Join Avenue Street13	New York13	India13	123499
14	Burj Kalifa14	Join Avenue Street14	New York14	India14	122156
15	Burj Kalifa15	Join Avenue Street15	New York15	India15	122196
16	Burj Kalifa16	Join Avenue Street16	New York16	India16	463456
17	Burj Kalifa17	Join Avenue Street17	New York17	India17	863456
18	Burj Kalifa18	Join Avenue Street18	New York18	India18	543456
19	Burj Kalifa19	Join Avenue Street19	New York19	India19	673456
20	Burj Kalifa20	Join Avenue Street20	New York20	India20	773456

Using INSERT command to add the different location (for listing, host address and guest address) with their description.

```
INSERT INTO LOCATION (location_name, street, city, country, pincode) VALUES
    ('Burj Kalifa1', 'Join Avenue Street1', 'New York1', 'India1', 123456),
    ('Burj Kalifa2', 'Join Avenue Street2', 'New York2', 'India2', 113456),
    ('Burj Kalifa3', 'Join Avenue Street3', 'New York3', 'India3', 126456),
    ('Burj Kalifa4', 'Join Avenue Street4', 'New York4', 'India4', 223456),
    ('Burj Kalifa5', 'Join Avenue Street5', 'New York5', 'India5', 323456),
    ('Burj Kalifa6', 'Join Avenue Street6', 'New York6', 'India6', 111456),
    ('Burj Kalifa7', 'Join Avenue Street7', 'New York7', 'India7', 123246),
    ('Burj Kalifa8', 'Join Avenue Street8', 'New York8', 'India8', 121236),
    ('Burj Kalifa9', 'Join Avenue Street9', 'New York9', 'India9', 123426),
    ('Burj Kalifa10', 'Join Avenue Street10', 'New York10', 'India10', 123458),
    ('Burj Kalifa11', 'Join Avenue Street11', 'New York11', 'India11', 123459),
    ('Burj Kalifa12', 'Join Avenue Street12', 'New York12', 'India12', 223458),
    ('Burj Kalifa13', 'Join Avenue Street13', 'New York13', 'India13', 123499),
    ('Burj Kalifa14', 'Join Avenue Street14', 'New York14', 'India14', 122156),
    ('Burj Kalifa15', 'Join Avenue Street15', 'New York15', 'India15', 122196),
    ('Burj Kalifa16', 'Join Avenue Street16', 'New York16', 'India16', 463456),
    ('Burj Kalifa17', 'Join Avenue Street17', 'New York17', 'India17', 863456),
    ('Burj Kalifa18', 'Join Avenue Street18', 'New York18', 'India18', 543456),
    ('Burj Kalifa19', 'Join Avenue Street19', 'New York19', 'India19', 673456),
    ('Burj Kalifa20', 'Join Avenue Street20', 'New York20', 'India20', 773456);
```

LOCATION : Test Case : Calculate the number of host of each locations who were born before 1970, between 1970 and 2000, and after 2000.

The queries use the connection between the location and host while ensuring connection between each location and listing doesn't fail. This give an insights for the user regarding the difference age distribution among the hosts

```
SELECT
    CASE
        WHEN h.date_of_birth < '1970-01-01' THEN 'Before 1970'
        WHEN h.date_of_birth BETWEEN '1970-01-01' AND '2000-12-31' THEN '1970 to 2000'
        ELSE 'After 2000'
    END AS date_of_birth_group,
    COUNT(*) AS count
FROM LOCATION l
    JOIN HOST h ON l.location_id = h.location_id
    JOIN LISTING ls ON ls.location_id = l.location_id
GROUP BY date_of_birth_group;
```

	date_of_birth_group text	count bigint
1	After 2000	2
2	1970 to 2000	11
3	Before 1970	7

SELECT Clause : It segment the entire table into three part based on the date_of_birth of the host and then select count of number of hosts that born between each three sub-age group.
FROM Clause : Specify the table (location) from which the data are retrieved along with joining the table with two other tables (listing and host).
GROUP BY Clause : Groups the result set with the three sub-age group, enabling the counting of hosts among each subgroup.

PHOTO

Store the different photo of listing, user profile picture and host profile picture within the system. The table is crucial for storing all photos and its information for all the user including administrator.

```
CREATE TABLE PHOTO(
    photo_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    photo BYTEA NOT NULL,
    extension VARCHAR(10) NOT NULL
);
```

photo_id [PK] integer	photo bytea	extension character varying (10)
1	[binary dat...	jpg
2	[binary dat...	webp
3	[binary dat...	webp
4	[binary dat...	jpg
5	[binary dat...	webp
6	[binary dat...	png
7	[binary dat...	jpeg
8	[binary dat...	jpeg
9	[binary dat...	jpg
10	[binary dat...	png
11	[binary dat...	jpg
12	[binary dat...	jpeg
13	[binary dat...	webp
14	[binary dat...	jpg
15	[binary dat...	jpg
16	[binary dat...	jpg
17	[binary dat...	jpg
18	[binary dat...	jpg
19	[binary dat...	jpg
20	[binary dat...	jpg

Using INSERT command to add the different picture with information.

```
INSERT INTO PHOTO(photo,extension) VALUES
    ('/Documents/IU collaborative work/Figure1.jpg', 'jpg'),
    ('/Documents/IU collaborative work/Figure3.webp', 'webp'),
    ('/Documents/IU collaborative work/Figure8.webp', 'webp'),
    ('/Documents/IU collaborative work/Figure9.jpg', 'jpg'),
    ('/Documents/IU collaborative work/Figure11.webp', 'webp'),
    ('/Documents/IU collaborative work/Figure12.png', 'png'),
    ('/Documents/IU collaborative work/Figure13.jpeg', 'jpeg'),
    ('/Documents/IU collaborative work/Figure14.jpeg', 'jpeg'),
    ('/Documents/IU collaborative work/Figure16.jpg', 'jpg'),
    ('/Documents/IU collaborative work/Figure17.png', 'png'),
    ('/Documents/IU collaborative work/Figure18.jpg', 'jpg'),
    ('/Documents/IU collaborative work/Figure19.jpeg', 'jpeg'),
    ('/Documents/IU collaborative work/Figure20.webp', 'webp'),
    ('/Documents/Manali/DSC_0001.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0002.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0003.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0004.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0005.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0006.jpg', 'jpg'),
    ('/Documents/Manali/DSC_0007.jpg', 'jpg');
```

PHOTO : Test Case : Calculate the number of picture which are host profile picture.

The queries use the connection between the host, photos and photo to calculate the number of host profile picture by using aggregation function and grouping. This give an insights for the administrator and developer regarding the difference photo distribution.

```
-- Count photos by photo type
SELECT ps.owner_type AS photo_type, COUNT(*)
FROM PHOTO p
    JOIN PHOTOS ps ON p.photo_id = ps.photo_id
    JOIN HOST h ON h.host_id = ps.owner_id AND ps.owner_type = 'host'
GROUP BY ps.owner_type;
```

	photo_type character varying (10)	count bigint
1	host	6

SELECT Clause : It select the owner type column as photo type upon which grouping is done and then count the total number of host for each photo type.
FROM Clause : Specify the table (Photo) from which the data are retrieved along with joining the table with two other tables (photos and host).
GROUP BY Clause : Groups the result set with the three sub-age group, enabling the counting of hosts among each subgroup.

HOST

Store all the necessary information of host within the system. The table is crucial and provide all the relevant information for all the user and would help the backend for any processing.

```
CREATE TABLE HOST (
    host_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    location_id INT NOT NULL UNIQUE,
    phone_number VARCHAR(15) NOT NULL,
    date_of_birth DATE,
    email VARCHAR(75) NOT NULL,
    password VARCHAR(35) NOT NULL,

    CONSTRAINT location_fk FOREIGN KEY(location_id) REFERENCES LOCATION(location_id) ON DELETE CASCADE

);
```

host_id [PK] integer	first_name character varying (30)	last_name character varying (30)	location_id integer	phone_number character varying (15)	date_of_birth date	email character varying (75)	password character varying (35)
1	Zuala1	Mithun1	4	12912321231	1970-01-03	zualamithun1@gmail.com	zualamithun1
2	Zuala2	Mithun2	5	12398621231	1912-01-03	zualamithun2@gmail.com	zualamithun2
3	Zuala3	Mithun3	14	6639862123	1980-01-03	zualamithun3@gmail.com	zualamithun3
4	Zuala4	Mithun4	15	5349862123	1990-01-03	zualamithun4@gmail.com	zualamithun4
5	Zuala5	Mithun5	1	298621231	2000-01-03	zualamithun5@gmail.com	zualamithun5
6	Zuala6	Mithun6	11	9098621231	1920-01-03	zualamithun6@gmail.com	zualamithun6
7	Zuala7	Mithun7	2	5598621231	1960-01-03	zualamithun7@gmail.com	zualamithun7
8	Zuala8	Mithun8	12	5398621231	1950-01-03	zualamithun8@gmail.com	zualamithun8
9	Zuala9	Mithun9	3	8798621231	1973-01-03	zualamithun9@gmail.com	zualamithun9
10	Zuala10	Mithun10	13	3298621231	1978-01-03	zualamithun10@gmail.com	zualamithun10
11	Zuala11	Mithun11	6	2398621231	1979-01-03	zualamithun11@gmail.com	zualamithun11
12	Zuala12	Mithun12	16	8898621231	1973-01-03	zualamithun12@gmail.com	zualamithun12
13	Zuala13	Mithun13	7	7798621231	1968-01-03	zualamithun13@gmail.com	zualamithun13
14	Zuala14	Mithun14	17	5698621231	1998-01-03	zualamithun14@gmail.com	zualamithun14
15	Zuala15	Mithun15	8	9898621231	2005-01-03	zualamithun15@gmail.com	zualamithun15
16	Zuala16	Mithun16	18	8098621231	1945-01-03	zualamithun16@gmail.com	zualamithun16
17	Zuala17	Mithun17	9	898621231	1973-01-03	zualamithun17@gmail.com	zualamithun17
18	Zuala18	Mithun18	19	3498621231	1962-01-03	zualamithun18@gmail.com	zualamithun18
19	Zuala19	Mithun19	20	8198621231	2000-01-03	zualamithun19@gmail.com	zualamithun19
20	Zuala20	Mithun20	10	1898621231	2001-01-03	zualamithun20@gmail.com	zualamithun20

Using INSERT command to add the different host with their description.

```
INSERT INTO HOST(first_name, last_name, location_id, phone_number, date_of_birth, email, password) VALUES
    ('Zuala1', 'Mithun1', 4, 12912321231, '1970-01-03', 'zualamithun1@gmail.com', 'zualamithun1'),
    ('Zuala2', 'Mithun2', 5, 12398621231, '1912-01-03', 'zualamithun2@gmail.com', 'zualamithun2'),
    ('Zuala3', 'Mithun3', 14, 6639862123, '1980-01-03', 'zualamithun3@gmail.com', 'zualamithun3'),
    ('Zuala4', 'Mithun4', 15, 5349862123, '1990-01-03', 'zualamithun4@gmail.com', 'zualamithun4'),
    ('Zuala5', 'Mithun5', 1, 298621231, '2000-01-03', 'zualamithun5@gmail.com', 'zualamithun5'),
    ('Zuala6', 'Mithun6', 11, 9098621231, '1920-01-03', 'zualamithun6@gmail.com', 'zualamithun6'),
    ('Zuala7', 'Mithun7', 2, 5598621231, '1960-01-03', 'zualamithun7@gmail.com', 'zualamithun7'),
    ('Zuala8', 'Mithun8', 12, 5398621231, '1950-01-03', 'zualamithun8@gmail.com', 'zualamithun8'),
    ('Zuala9', 'Mithun9', 3, 8798621231, '1973-01-03', 'zualamithun9@gmail.com', 'zualamithun9'),
    ('Zuala10', 'Mithun10', 13, 3298621231, '1978-01-03', 'zualamithun10@gmail.com', 'zualamithun10'),
    ('Zuala11', 'Mithun11', 6, 2398621231, '1979-01-03', 'zualamithun11@gmail.com', 'zualamithun11'),
    ('Zuala12', 'Mithun12', 16, 8898621231, '1973-01-03', 'zualamithun12@gmail.com', 'zualamithun12'),
    ('Zuala13', 'Mithun13', 7, 7798621231, '1968-01-03', 'zualamithun13@gmail.com', 'zualamithun13'),
    ('Zuala14', 'Mithun14', 17, 5698621231, '1998-01-03', 'zualamithun14@gmail.com', 'zualamithun14'),
    ('Zuala15', 'Mithun15', 8, 9898621231, '2005-01-03', 'zualamithun15@gmail.com', 'zualamithun15'),
    ('Zuala16', 'Mithun16', 18, 8098621231, '1945-01-03', 'zualamithun16@gmail.com', 'zualamithun16'),
    ('Zuala17', 'Mithun17', 9, 0898621231, '1973-01-03', 'zualamithun17@gmail.com', 'zualamithun17'),
    ('Zuala18', 'Mithun18', 19, 3498621231, '1962-01-03', 'zualamithun18@gmail.com', 'zualamithun18'),
    ('Zuala19', 'Mithun19', 20, 8198621231, '2000-01-03', 'zualamithun19@gmail.com', 'zualamithun19'),
    ('Zuala20', 'Mithun20', 10, 1898621231, '2001-01-03', 'zualamithun20@gmail.com', 'zualamithun20');
```

HOST : Test Case : Calculate the number of host using different payment method by joining the host table with the payment table while ensuring that each host are under a user(administration).

The queries use the connection between the host, user and payment to calculate the number of host using different payment method by using aggregation function and grouping. This give an insights for the user regarding the difference payment method which the host accept.

```
-- Count payments by type for hosts
SELECT payment_type, COUNT(*)
FROM HOST h
    JOIN PAYMENT p ON h.host_id = p.person_id AND p.person_type = 'host'
    JOIN "USER" a ON a.host_id = h.host_id
GROUP BY payment_type;
```

	payment_type character varying (15)	count bigint
1	net_banking	3
2	upi	5
3	credit_debit	2

SELECT Clause : It select the different payment type upon which grouping is done and then count the total number of host (or objects) for each different payment method.
FROM Clause : Specify the table (Host) from which the data are retrieved along with joining the table with two other tables (user and payment).
GROUP BY Clause : Groups the result set with the different payment group, enabling the counting of hosts among each subgroup.

GUEST

Store all the necessary information of guest within the system. The table is crucial and provide all the relevant information for all the host and administrator regarding the guest.

```
CREATE TABLE GUEST(
    guest_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    location_id INT NOT NULL UNIQUE,
    phone_number VARCHAR(15) NOT NULL,
    date_of_birth DATE NOT NULL,
    email VARCHAR(75) NOT NULL,
    password VARCHAR(35) NOT NULL,
    emergency_contact VARCHAR(15),

    CONSTRAINT location_fk FOREIGN KEY(location_id) REFERENCES LOCATION(location_id) ON DELETE CASCADE
);
```

guest_id [PK] integer	first_name character varying (30)	last_name character varying (30)	location_id integer	phone_number character varying (15)	date_of_birth date	email character varying (75)	password character varying (35)
1	Hminga1	Huna1	4	12912321231	1970-01-03	hmingamithun1@gmail.com	zualaHuna1
2	Hminga2	Huna2	5	12398621231	1912-01-03	hmingamithun2@gmail.com	zualaHuna2
3	Hminga3	Huna3	14	6639862123	1980-01-03	hmingamithun3@gmail.com	zualaHuna3
4	Hminga4	Huna4	15	5349862123	1990-01-03	hmingamithun4@gmail.com	zualaHuna4
5	Hminga5	Huna5	1	298621231	2000-01-03	hmingamithun5@gmail.com	zualaHuna5
6	Hminga6	Huna6	11	9098621231	1920-01-03	hmingamithun6@gmail.com	zualaHuna6
7	Hminga7	Huna7	2	5598621231	1960-01-03	hmingamithun7@gmail.com	zualaHuna7
8	Hminga8	Huna8	12	5398621231	1950-01-03	hmingamithun8@gmail.com	zualaHuna8
9	Hminga9	Huna9	3	8798621231	1973-01-03	hmingamithun9@gmail.com	zualaHuna9
10	Hminga10	Huna10	13	3298621231	1978-01-03	hmingamithun10@gmail.com	zualaHuna10
11	Hminga11	Huna11	6	2398621231	1979-01-03	hmingamithun11@gmail.com	zualaHuna11
12	Hminga12	Huna12	16	8898621231	1973-01-03	hmingamithun12@gmail.com	zualaHuna12
13	Hminga13	Huna13	7	7798621231	1968-01-03	hmingamithun13@gmail.com	zualaHuna13
14	Hminga14	Huna14	17	5698621231	1998-01-03	hmingamithun14@gmail.com	zualaHuna14
15	Hminga15	Huna15	8	9898621231	2005-01-03	hmingamithun15@gmail.com	zualaHuna15
16	Hminga16	Huna16	18	8098621231	1945-01-03	hmingamithun16@gmail.com	zualaHuna16
17	Hminga17	Huna17	9	898621231	1973-01-03	hmingamithun17@gmail.com	zualaHuna17
18	Hminga18	Huna18	19	3498621231	1962-01-03	hmingamithun18@gmail.com	zualaHuna18
19	Hminga19	Huna19	20	8198621231	2000-01-03	hmingamithun19@gmail.com	zualaHuna19
20	Hminga20	Huna20	10	1898621231	2001-01-03	hmingamithun20@gmail.com	zualaHuna20

Using INSERT command to add the different guest with their information

```
INSERT INTO GUEST(first_name, last_name, location_id, phone_number, date_of_birth, email, password) VALUES
    ('Hminga1', 'Huna1', 4, 12912321231, '1970-01-03', 'hmingamithun1@gmail.com', 'zualaHuna1'),
    ('Hminga2', 'Huna2', 5, 12398621231, '1912-01-03', 'hmingamithun2@gmail.com', 'zualaHuna2'),
    ('Hminga3', 'Huna3', 14, 6639862123, '1980-01-03', 'hmingamithun3@gmail.com', 'zualaHuna3'),
    ('Hminga4', 'Huna4', 15, 5349862123, '1990-01-03', 'hmingamithun4@gmail.com', 'zualaHuna4'),
    ('Hminga5', 'Huna5', 1, 298621231, '2000-01-03', 'hmingamithun5@gmail.com', 'zualaHuna5'),
    ('Hminga6', 'Huna6', 11, 9098621231, '1920-01-03', 'hmingamithun6@gmail.com', 'zualaHuna6'),
    ('Hminga7', 'Huna7', 2, 5598621231, '1960-01-03', 'hmingamithun7@gmail.com', 'zualaHuna7'),
    ('Hminga8', 'Huna8', 12, 5398621231, '1950-01-03', 'hmingamithun8@gmail.com', 'zualaHuna8'),
    ('Hminga9', 'Huna9', 3, 8798621231, '1973-01-03', 'hmingamithun9@gmail.com', 'zualaHuna9'),
    ('Hminga10', 'Huna10', 13, 3298621231, '1978-01-03', 'hmingamithun10@gmail.com', 'zualaHuna10'),
    ('Hminga11', 'Huna11', 6, 2398621231, '1979-01-03', 'hmingamithun11@gmail.com', 'zualaHuna11'),
    ('Hminga12', 'Huna12', 16, 8898621231, '1973-01-03', 'hmingamithun12@gmail.com', 'zualaHuna12'),
    ('Hminga13', 'Huna13', 7, 7798621231, '1968-01-03', 'hmingamithun13@gmail.com', 'zualaHuna13'),
    ('Hminga14', 'Huna14', 17, 5698621231, '1998-01-03', 'hmingamithun14@gmail.com', 'zualaHuna14'),
    ('Hminga15', 'Huna15', 8, 9898621231, '2005-01-03', 'hmingamithun15@gmail.com', 'zualaHuna15'),
    ('Hminga16', 'Huna16', 18, 8098621231, '1945-01-03', 'hmingamithun16@gmail.com', 'zualaHuna16'),
    ('Hminga17', 'Huna17', 9, 0898621231, '1973-01-03', 'hmingamithun17@gmail.com', 'zualaHuna17'),
    ('Hminga18', 'Huna18', 19, 3498621231, '1962-01-03', 'hmingamithun18@gmail.com', 'zualaHuna18'),
    ('Hminga19', 'Huna19', 20, 8198621231, '2000-01-03', 'hmingamithun19@gmail.com', 'zualaHuna19'),
    ('Hminga20', 'Huna20', 10, 1898621231, '2001-01-03', 'hmingamithun20@gmail.com', 'zualaHuna20');
```

GUEST : Test Case : Calculate the number of guest using different payment method by joining the guest table with the payment table while ensuring that each guest are under a user.

The queries use the connection between the guest, user and payment to calculate the number of guest using different payment method by using aggregation function and grouping. This give an insights for the host regarding the difference payment method which the guest accept.

```
-- Count payments by type for guests
SELECT payment_type, COUNT(*)
FROM GUEST g
    JOIN PAYMENT p ON g.guest_id = p.person_id AND p.person_type = 'guest'
    JOIN "USER" a ON a.guest_id = g.guest_id
GROUP BY payment_type;
```

payment_type character varying (15)	count bigint
net_banking	2
upi	5
credit_debit	3

SELECT Clause : It select the different payment type upon which grouping is done and then count the total number of guest (or objects) for each different payment method.
FROM Clause : Specify the table (Guest) from which the data are retrieved along with joining the table with two other tables (user and payment).
GROUP BY Clause : Groups the result set with the different payment group, enabling the counting of guest among each subgroup.

WISHLIST

Link the guest with the listing that they include in their individual wish-list within the system. The table is crucial and provide all the relevant information for all the host regarding the guest preference.

```
CREATE TABLE WISHLIST(
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    listing_id INT NOT NULL UNIQUE,
    guest_id INT NOT NULL,

    CONSTRAINT listing_fk FOREIGN KEY(listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE,
    CONSTRAINT guest_fk FOREIGN KEY(guest_id) REFERENCES GUEST(guest_id) ON DELETE CASCADE
);
```

id [PK] integer	listing_id integer	guest_id integer
1	11	1
2	12	2
3	13	3
4	14	4
5	15	5
6	16	6
7	17	7
8	18	8
9	19	9
10	20	10
11	1	11
12	2	12
13	3	13
14	4	14
15	5	15
16	6	16
17	7	17
18	8	18
19	9	19
20	10	20

Using INSERT command to add the listing id which for each guest as their wishlist.

```
INSERT INTO WISHLIST(listing_id, guest_id) VALUES
    (11,1),
    (12,2),
    (13,3),
    (14,4),
    (15,5),
    (16,6),
    (17,7),
    (18,8),
    (19,9),
    (20,10),
    (1,11),
    (2,12),
    (3,13),
    (4,14),
    (5,15),
    (6,16),
    (7,17),
    (8,18),
    (9,19),
    (10,20);
```

WISHLIST : Test Case : Calculate the number of guest which has wishlist who were born before 1970, between 1970 and 2000, and after 2000.

The queries use the connection between the guest, wishlist and listing to calculate the number of guest with wishlist who born before 1970, between 1970 and 2000, and after 2000 by using aggregation function and grouping. This ensure the connection between guest, wishlist and listing.

```
-- Group guests by date of birth with wishlist
SELECT
    CASE
        WHEN g.date_of_birth < '1970-01-01' THEN 'Before 1970'
        WHEN g.date_of_birth BETWEEN '1970-01-01' AND '2000-12-31' THEN '1970 to 2000'
        ELSE 'After 2000'
    END AS date_of_birth_group,
    COUNT(*) AS count
FROM WISHLIST w
    JOIN GUEST g ON g.guest_id = w.guest_id
    JOIN LISTING l ON w.listing_id = l.listing_id
GROUP BY date_of_birth_group;
```

	date_of_birth_group text	count bigint
1	After 2000	2
2	1970 to 2000	11
3	Before 1970	7

SELECT Clause : It select and segment the entire table into three part based on date of birth and then count the total number of guest (or objects) for each different birth group.
FROM Clause : Specify the table (Wishlist) from which the data are retrieved along with joining the table with two other tables (guest and listing).
GROUP BY Clause : Groups the result set with the different age group, enabling the counting of guest among each subgroup.

LISTING

Store all the necessary reference of listing within the system. The table is crucial by acting as a central module for all the listing providing the relevant reference of information for the system.

```
CREATE TABLE LISTING(
    listing_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    place_id INT NOT NULL UNIQUE,
    location_id INT NOT NULL UNIQUE UNIQUE,

    CONSTRAINT place_fk FOREIGN KEY(place_id) REFERENCES PLACE(place_id),
    CONSTRAINT location_fk FOREIGN KEY(location_id) REFERENCES LOCATION(location_id) ON DELETE SET NULL
);
```

listing_id [PK] integer	place_id integer	location_id integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Using INSERT command to add the id of place, host and location for each listing.

```
INSERT INTO LISTING (place_id, location_id)
VALUES (1, 1),
       (2, 2),
       (3, 3),
       (4, 4),
       (5, 5),
       (6, 6),
       (7, 7),
       (8, 8),
       (9, 9),
       (10, 10),
       (11, 11),
       (12, 12),
       (13, 13),
       (14, 14),
       (15, 15),
       (16, 16),
       (17, 17),
       (18, 18),
       (19, 19),
       (20, 20);
```

LISTING : Test Case : Calculate the number of listings which are given 4 star in the review which are with or without wifi.

The queries use the connection between the wifi, listing and review to calculate the number of listings with and without wifi for which are given 4 star as review by using aggregation function and grouping. It also join with amenities to ensure and check the connection with listing.

```
-- Count listings with wifi that have 4-star reviews
SELECT wifi, COUNT(*)
FROM LISTING l
    JOIN LISTING_AMENITIES la ON l.listing_id = la.listing_id
    JOIN AMENITIES a ON la.amenities_id = a.amenities_id
    JOIN REVIEW r ON l.listing_id = r.listing_id
WHERE r.star = 4
GROUP BY wifi;
```

	wifi boolean	count bigint
1	false	2
2	true	4

SELECT Clause : It select the wifi column segment the entire data to listing with and without wifi.
FROM Clause : Specify the table (Listing) from which the data are retrieved along with joining the table with three other tables (amenities, place and review).
GROUP BY Clause : Groups the result set with the different wifi sub-group, enabling the counting of listing with 4-star review among each subgroup.

PLACE

Store all the necessary information of the place for each listing within the system. The table is crucial by providing all the necessary description which entailed all the relevant reference and information for all the user.

```
CREATE TABLE PLACE(
    place_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    type VARCHAR(25) NOT NULL,
    description TEXT NOT NULL,
    max_adult INT NOT NULL,
    max_children INT NOT NULL,
    allowed_pet BOOLEAN NOT NULL,
    exterior_security_camera BOOLEAN NOT NULL,
    security BOOLEAN NOT NULL
);
```

place_id [PK] integer	type character varying (25)	description text	max_adult integer	max_children integer	allowed_pet boolean	exterior_security_camera boolean
1	Apartment	Cozy 1-bedroom apartment in the city center	2	1	false	true
2	Villa	Luxurious 5-bedroom villa with private pool	10	5	true	true
3	Cabin	Rustic cabin in the mountains	4	2	true	false
4	Hostel	Affordable hostel in the heart of the city	10	5	false	false
5	Hotel	Upscale 4-star hotel with spa and fitness center	4	2	false	true
6	Cottage	Charming cottage near a lake	6	3	true	false
7	Townhouse	Modern townhouse in a gated community	6	4	true	true
8	Bungalow	Beachfront bungalow with ocean views	4	2	false	true
9	Condo	Spacious 2-bedroom condo near shopping and restauran...	4	2	false	true
10	Chalet	Cozy chalet in a ski resort	8	4	true	false
11	Loft	Industrial-style loft in an urban area	2	1	false	true
12	Penthouse	Luxurious penthouse with panoramic city views	4	2	false	true
13	Ranch	Spacious ranch-style home in the countryside	8	6	true	false
14	Farmhouse	Rustic farmhouse on a working farm	6	4	true	false
15	Treehouse	Unique treehouse experience in a forest setting	2	1	false	false
16	Yurt	Traditional yurt in a scenic mountain meadow	4	2	true	false
17	Houseboat	Charming houseboat docked in a marina	4	2	false	false
18	Castle	Historic castle turned into a luxury hotel	10	6	false	true
19	Igloo	Unique ice igloo experience in the Arctic	2	1	false	false
20	Tent	Camping tent in a scenic nature reserve	4	2	true	false

Using INSERT command to add all the information required for a specific place.

```
INSERT INTO PLACE (type, description, max_adult, max_children, allowed_pet, exterior_security_camera, security)
VALUES
    ('Apartment', 'Cozy 1-bedroom apartment in the city center', 2, 1, false, true, true),
    ('Villa', 'Luxurious 5-bedroom villa with private pool', 10, 5, true, true, true),
    ('Cabin', 'Rustic cabin in the mountains', 4, 2, true, false, false),
    ('Hostel', 'Affordable hostel in the heart of the city', 10, 5, false, false, true),
    ('Hotel', 'Upscale 4-star hotel with spa and fitness center', 4, 2, false, true, true),
    ('Cottage', 'Charming cottage near a lake', 6, 3, true, false, false),
    ('Townhouse', 'Modern townhouse in a gated community', 6, 4, true, true, true),
    ('Bungalow', 'Beachfront bungalow with ocean views', 4, 2, false, true, true),
    ('Condo', 'Spacious 2-bedroom condo near shopping and restaurants', 4, 2, false, true, true),
    ('Chalet', 'Cozy chalet in a ski resort', 8, 4, true, false, true),
    ('Loft', 'Industrial-style loft in an urban area', 2, 1, false, true, true),
    ('Penthouse', 'Luxurious penthouse with panoramic city views', 4, 2, false, true, true),
    ('Ranch', 'Spacious ranch-style home in the countryside', 8, 6, true, false, false),
    ('Farmhouse', 'Rustic farmhouse on a working farm', 6, 4, true, false, false),
    ('Treehouse', 'Unique treehouse experience in a forest setting', 2, 1, false, false, false),
    ('Yurt', 'Traditional yurt in a scenic mountain meadow', 4, 2, true, false, false),
    ('Houseboat', 'Charming houseboat docked in a marina', 4, 2, false, false, true),
    ('Castle', 'Historic castle turned into a luxury hotel', 10, 6, false, true, true),
    ('Igloo', 'Unique ice igloo experience in the Arctic', 2, 1, false, false, false),
    ('Tent', 'Camping tent in a scenic nature reserve', 4, 2, true, false, false);
```

PLACE : Test Case : Calculate the average of the maximum number of adult and maximum number of children for each place type.

The queries use the connection between the place, tags and listing to calculate the average of the maximum number of adult and maximum number of children for each place type.
by using aggregation function and grouping.

```
-- Average max adults and children allowed per place type
SELECT type, AVG(max_adult) AS average_adult_allowed, AVG(max_children) AS average_child_allowed
FROM PLACE p
    JOIN LISTING l ON p.place_id = l.place_id
    JOIN TAGS t ON t.place_id = p.place_id
GROUP BY type;
```

type character varying (25)	average_adult_allowed numeric	average_child_allowed numeric
Apartment	2.0000000000000000	1.0000000000000000000
Bungalow	4.0000000000000000	2.0000000000000000
Ranch	8.0000000000000000	6.0000000000000000
Condo	4.0000000000000000	2.0000000000000000
Igloo	2.0000000000000000	1.0000000000000000000
Farmhouse	6.0000000000000000	4.0000000000000000
Chalet	8.0000000000000000	4.0000000000000000
Tent	4.0000000000000000	2.0000000000000000
Hotel	4.0000000000000000	2.0000000000000000
Penthouse	4.0000000000000000	2.0000000000000000
Cottage	6.0000000000000000	3.0000000000000000
Houseboat	4.0000000000000000	2.0000000000000000
Castle	10.0000000000000000	6.0000000000000000
Hostel	10.0000000000000000	5.0000000000000000
Yurt	4.0000000000000000	2.0000000000000000
Loft	2.0000000000000000	1.0000000000000000000
Villa	10.0000000000000000	5.0000000000000000
Cabin	4.0000000000000000	2.0000000000000000
Treehouse	2.0000000000000000	1.0000000000000000000
Townhouse	6.0000000000000000	4.0000000000000000

SELECT Clause : It select the type column representing the different place type along with evaluating the average aggregation function to max_adult and max_children
FROM Clause : Specify the table (Place) from which the data are retrieved along with joining the table with two other tables (Tags and listing).
GROUP BY Clause : Groups the result set with the different place type, enabling the averaging of max_adult and max_child among each subgroup.

AMENITIES

Store all the necessary information of amenities for each individual listing within the system. The table is crucial by providing all the information regarding amenities available for each place.

```
CREATE TABLE AMENITIES(
    amenities_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    number_bedrooms INT NOT NULL,
    num_bathrooms INT NOT NULL,
    wifi BOOLEAN NOT NULL,
    interior_security_camera BOOLEAN,
    ac BOOLEAN,
    laundry TEXT,
    house_rules TEXT NOT NULL,
    cancellation_policy TEXT NOT NULL,
    additional_services TEXT,
    check_in_check_out_instructions TEXT NOT NULL,
    others TEXT

);
```

amenities_id [PK] integer	number_bedrooms integer	num_bathrooms integer	wifi boolean	interior_security_camera boolean	ac boolean	laundry text	house_rules text
1	1	1	true	false	true	Coin-operated laundry available on-site	No smoking, no parties, qu
2	5	4	true	true	true	In-unit washer and dryer	Pets allowed with a fee, nc
3	2	1	false	false	false	No laundry facilities	No parties, quiet hours aft
4	6	3	true	false	true	[null]	No smoking, no pets allow
5	4	2	true	true	true	Coin-operated laundry room on each floor	No smoking indoors, quiet
6	2	1	false	false	false	In-unit washer and dryer	Pets allowed with a fee, nc
7	3	2	true	false	true	[null]	No smoking, no pets allow
8	2	1	true	false	true	Coin-operated laundry facilities on-site	No smoking indoors, quiet
9	2	2	true	true	true	In-unit washer and dryer	Pets allowed with a fee, nc
10	4	3	true	false	true	[null]	No smoking, no pets allow
11	1	1	false	false	false	Coin-operated laundry facilities on each flo...	No smoking indoors, quiet
12	3	2	true	true	true	In-unit washer and dryer	Pets allowed with a fee, nc
13	4	3	true	false	true	[null]	No smoking, no pets allow
14	3	2	false	false	true	Coin-operated laundry facilities on-site	No smoking indoors, quiet
15	1	1	true	false	false	In-unit washer and dryer	Pets allowed with a fee, nc
16	2	1	false	false	false	[null]	No smoking, no pets allow
17	6	5	true	true	true	Coin-operated laundry room on each floor	No smoking indoors, quiet
18	1	1	false	false	false	In-unit washer and dryer	Pets allowed with a fee, nc
19	2	1	true	false	true	[null]	No smoking, no pets allow
20	6	5	true	true	true	Coin-operated laundry facilities on-site	No smoking indoors, quiet

Using INSERT command to add all the information required for all amenities

```
INSERT INTO AMENITIES (number_bedrooms, num_bathrooms, wifi, interior_security_camera, ac, laundry, house_rules, cancellation_
VALUES
    (1, 1, true, false, true, 'Coin-operated laundry available on-site', 'No smoking, no parties, quiet hours after 10 PM', '2
    (5, 4, true, true, true, 'In-unit washer and dryer', 'Pets allowed with a fee, no smoking indoors', '48-hour cancellation
    (2, 1, false, false, false, 'No laundry facilities', 'No parties, quiet hours after 11 PM, maximum occupancy strictly enfo
    (6, 3, true, false, true, NULL, 'No smoking, no pets allowed', '72-hour cancellation policy, full refund if cancelled with
    (4, 2, true, true, true, 'Coin-operated laundry room on each floor', 'No smoking indoors, quiet hours after 9 PM', '48-hou
    (2, 1, false, false, false, 'In-unit washer and dryer', 'Pets allowed with a fee, no parties, quiet hours after 10 PM', '2
    (3, 2, true, false, true, NULL, 'No smoking, no pets allowed, maximum occupancy strictly enforced', 'Non-refundable, no ca
    (2, 1, true, false, true, 'Coin-operated laundry facilities on-site', 'No smoking indoors, quiet hours after 11 PM', '72-h
    (2, 2, true, true, true, 'In-unit washer and dryer', 'Pets allowed with a fee, no parties, quiet hours after 9 PM', '48-ho
    (4, 3, true, false, true, NULL, 'No smoking, no pets allowed, maximum occupancy strictly enforced', 'Non-refundable, no ca
    (1, 1, false, false, false, 'Coin-operated laundry facilities on each floor', 'No smoking indoors, quiet hours after 10 PM
    (3, 2, true, true, true, 'In-unit washer and dryer', 'Pets allowed with a fee, no parties, quiet hours after 11 PM', '72-h
    (4, 3, true, false, true, NULL, 'No smoking, no pets allowed, maximum occupancy strictly enforced', 'Non-refundable, no ca
    (3, 2, false, false, true, 'Coin-operated laundry facilities on-site', 'No smoking indoors, quiet hours after 9 PM', '48-h
    (1, 1, true, false, false, 'In-unit washer and dryer', 'Pets allowed with a fee, no parties, quiet hours after 10 PM', '24
    (2, 1, false, false, false, NULL, 'No smoking, no pets allowed, maximum occupancy strictly enforced', 'Non-refundable, no
    (6, 5, true, true, true, 'Coin-operated laundry room on each floor', 'No smoking indoors, quiet hours after 11 PM', '72-ho
    (1, 1, false, false, false, 'In-unit washer and dryer', 'Pets allowed with a fee, no parties, quiet hours after 9 PM', '48
    (2, 1, true, false, true, NULL, 'No smoking, no pets allowed, maximum occupancy strictly enforced', 'Non-refundable, no ca
    (6, 5, true, true, true, 'Coin-operated laundry facilities on-site', 'No smoking indoors, quiet hours after 10 PM', '24-ho
```

AMENITIES : Test Case : Calculate the number of ac available on each property wherein the listing has wi-fi.

The queries use the connection between the amenities, property and listing to calculate the number of ac available on each property wherein the listing has an AC by using aggregation function and grouping.

```
-- Count listings with AC and wifi
SELECT ac, COUNT(*)
FROM AMENITIES a
    JOIN LISTING_AMENITIES la ON la.amenities_id = a.amenities_id
    JOIN LISTING l ON l.listing_id = la.listing_id
WHERE a.wifi = true
GROUP BY ac;
```

	ac boolean	count bigint
1	false	1
2	true	13

SELECT Clause : It select the AC column representing whether the property has an AC or not along with counting the number of AC for those property which has wifi
FROM Clause : Specify the table (Amenities) from which the data are retrieved along with joining the table with two other tables (listing_amenities and listing).
GROUP BY Clause : Groups the result set based on existence of AC, enabling the counting of listing for each subgroup.

PHOTOS

Link guest, listing and host to each photo. The table is crucial by storing and ensuring the identification of each photo in the PHOTO table.

```
CREATE TABLE PHOTOS(
    photos_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    owner_id INT NOT NULL,
    owner_type VARCHAR(10) NOT NULL CHECK (owner_type IN ('listing','host','guest')),
    photo_id INT NOT NULL,

    CONSTRAINT photo_fk FOREIGN KEY(photo_id) REFERENCES PHOTO(photo_id) ON DELETE CASCADE
);
```

photos_id [PK] integer	owner_id integer	owner_type character varying (10)	photo_id integer
1	1	listing	1
2	2	host	2
3	3	listing	3
4	4	guest	4
5	5	listing	5
6	6	host	6
7	7	host	7
8	8	listing	8
9	9	guest	9
10	10	guest	10
11	11	host	11
12	12	listing	12
13	13	listing	13
14	14	host	14
15	15	host	15
16	16	listing	16
17	17	guest	17
18	18	guest	18
19	19	listing	19
20	20	listing	20

Using INSERT command to add all relevant information for each photos.

```
INSERT INTO PHOTOS (owner_id,owner_type, photo_id)
VALUES (1,'listing',1),(2,'host',2),(3,'listing',3),(4,'guest',4),(5,'listing',5),
    (6,'host',6),(7,'host',7),(8,'listing',8),(9,'guest',9),(10,'guest',10),
    (11,'host',11),(12,'listing',12),(13,'listing',13),(14,'host',14),(15,'host',15),
    (16,'listing',16),(17,'guest',17),(18,'guest',18),(19,'listing',19),(20,'listing',20);
```

PHOTOS : Test Case **:** Count the number of property which has interior security cameras.

The queries use the connection between the property, amenities, and listing to count the number of properties with interior security cameras by using aggregation function and grouping. It also ensure connection between listing and amenities.

```
SELECT ps.owner_type, COUNT(*)
FROM PHOTOS ps
    JOIN PHOTO p ON ps.photo_id = p.photo_id
GROUP BY ps.owner_type
```

owner_type character varying (10)	count bigint
host	6
guest	5
listing	9

SELECT Clause : It selects the interior security camera column and the count the number of property which has interior security camera.
FROM Clause : Specifies the tables (PROPERTY, AMENITIES, LISTING) from which the data are retrieved, joining them on appropriate keys.
GROUP BY Clause : Groups the result set by the interior security camera column, enabling the count of properties in each subgroup.

AVAILABILITY

Store all the necessary information of availability for each listing within the system. This table is crucial for providing the available date ranges for each listing.

```
CREATE TABLE AVAILABILITY(
    availability_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    start_available DATE NOT NULL,
    end_available DATE NOT NULL,
    listing_id INT NOT NULL,

    CONSTRAINT listing_fk FOREIGN KEY (listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE
);
```

availability_id [PK] integer	start_available date	end_available date	listing_id integer
1	2023-06-01	2023-06-15	1
2	2023-07-10	2023-07-20	2
3	2023-08-15	2023-08-25	3
4	2023-09-01	2023-09-10	4
5	2023-10-01	2023-10-15	5
6	2023-11-01	2023-11-15	6
7	2023-12-15	2024-01-05	7
8	2024-02-01	2024-02-15	8
9	2024-03-10	2024-03-20	9
10	2024-04-01	2024-04-15	10
11	2024-05-01	2024-05-15	11
12	2024-06-01	2024-06-15	12
13	2024-07-10	2024-07-20	13
14	2024-08-15	2024-08-25	14
15	2024-09-01	2024-09-10	15
16	2024-10-01	2024-10-15	16
17	2024-11-01	2024-11-15	17
18	2024-12-15	2025-01-05	18
19	2025-02-01	2025-02-15	19
20	2025-03-10	2025-03-20	20

Using INSERT command to add all relevant information for each availability.

```
INSERT INTO AVAILABILITY (start_available, end_available, listing_id)
VALUES ('2023-06-01', '2023-06-15', 1),
       ('2023-07-10', '2023-07-20', 2),
       ('2023-08-15', '2023-08-25', 3),
       ('2023-09-01', '2023-09-10', 4),
       ('2023-10-01', '2023-10-15', 5),
       ('2023-11-01', '2023-11-15', 6),
       ('2023-12-15', '2024-01-05', 7),
       ('2024-02-01', '2024-02-15', 8),
       ('2024-03-10', '2024-03-20', 9),
       ('2024-04-01', '2024-04-15', 10),
       ('2024-05-01', '2024-05-15', 11),
       ('2024-06-01', '2024-06-15', 12),
       ('2024-07-10', '2024-07-20', 13),
       ('2024-08-15', '2024-08-25', 14),
       ('2024-09-01', '2024-09-10', 15),
       ('2024-10-01', '2024-10-15', 16),
       ('2024-11-01', '2024-11-15', 17),
       ('2024-12-15', '2025-01-05', 18),
       ('2025-02-01', '2025-02-15', 19),
       ('2025-03-10', '2025-03-20', 20);
```

AVAILABILITY : Test Case : Count the number of listing which will start available in the years 2023, 2024, and 2025 and order them by year.

The queries use the connection between the availability and listing to count the number of availabilities starting in the specified years.

```sql
-- Count availability by year
SELECT
    EXTRACT(YEAR FROM a.start_available) AS year,
    COUNT(*) AS count
FROM AVAILABILITY a
    JOIN LISTING l ON a.listing_id = l.listing_id
WHERE EXTRACT(YEAR FROM a.start_available) IN (2023, 2024, 2025)
GROUP BY EXTRACT(YEAR FROM a.start_available)
ORDER BY year;
```

year numeric	count bigint
2023	7
2024	11
2025	2

SELECT Clause : It selects the year extracted from the start available date and the count the number of availabilities starting in each year.
FROM Clause : Specifies the tables (AVAILABILITY, LISTING) from which the data are retrieved, joining them on appropriate keys and filtering for specific years.
GROUP BY Clause : Groups the result set by the extracted year, enabling the count of availabilities in each subgroup.
ORDER BY Clause : Orders the result set by the year.

REVIEW

Store all the necessary information of reviews for each listing within the system. This table is crucial for capturing feedback, including star ratings and descriptions, associated with listing and guest.

```
CREATE TABLE REVIEW(
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    star INT NOT NULL,
    description TEXT,
    listing_id INT NOT NULL,
    guest_id INT NOT NULL,

    CONSTRAINT list_fk FOREIGN KEY(listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE,
    CONSTRAINT guest_fk FOREIGN KEY(guest_id) REFERENCES GUEST(guest_id)  ON DELETE CASCADE
);
```

id [PK] integer	star integer	description text	listing_id integer	guest_id integer
1	5	Amazing place! Highly recommended.	1	20
2	4	Great location, but a bit noisy at night.	2	19
3	3	The property was clean, but the amenities were basic.	3	18
4	5	Fantastic host, beautiful property, and excellent service.	4	17
5	2	The place was not well-maintained, and the description was misleading.	5	16
6	4	Good value for money, but the check-in process could be improved.	6	15
7	5	Exceeded our expectations! We had a wonderful stay.	7	13
8	3	Average experience, nothing exceptional.	8	12
9	4	The property was comfortable and the host was friendly.	9	11
10	2	The place was not as advertised, and the amenities were lacking.	10	10
11	5	Absolutely stunning property! We loved every minute of our stay.	11	9
12	4	The location was great, but the place could use some updates.	12	8
13	3	It was an okay stay, but we expected more for the price.	13	7
14	5	Fantastic experience! The host went above and beyond.	14	6
15	2	The property was not well-maintained, and the check-out process was confusi...	15	5
16	4	Good value for money, and the host was responsive.	16	4
17	5	Highly recommended! We had a great time and made wonderful memories.	17	3
18	3	The place was decent, but nothing too special.	18	2
19	4	The property was clean and comfortable, and the host was friendly.	19	1
20	2	The description was misleading, and the place did not meet our expectations.	20	14

Using INSERT command to add all relevant information for each review.

```
INSERT INTO REVIEW (star, description, listing_id, guest_id)
VALUES (5, 'Amazing place! Highly recommended.', 1, 20),
       (4, 'Great location, but a bit noisy at night.', 2, 19),
       (3, 'The property was clean, but the amenities were basic.', 3, 18),
       (5, 'Fantastic host, beautiful property, and excellent service.', 4, 17),
       (2, 'The place was not well-maintained, and the description was misleading.', 5, 16),
       (4, 'Good value for money, but the check-in process could be improved.', 6, 15),
       (5, 'Exceeded our expectations! We had a wonderful stay.', 7, 13),
       (3, 'Average experience, nothing exceptional.', 8, 12),
       (4, 'The property was comfortable and the host was friendly.', 9, 11),
       (2, 'The place was not as advertised, and the amenities were lacking.', 10, 10),
       (5, 'Absolutely stunning property! We loved every minute of our stay.', 11, 9),
       (4, 'The location was great, but the place could use some updates.', 12, 8),
       (3, 'It was an okay stay, but we expected more for the price.', 13, 7),
       (5, 'Fantastic experience! The host went above and beyond.', 14, 6),
       (2, 'The property was not well-maintained, and the check-out process was confusing.', 15, 5),
       (4, 'Good value for money, and the host was responsive.', 16, 4),
       (5, 'Highly recommended! We had a great time and made wonderful memories.', 17, 3),
       (3, 'The place was decent, but nothing too special.', 18, 2),
       (4, 'The property was clean and comfortable, and the host was friendly.', 19, 1),
       (2, 'The description was misleading, and the place did not meet our expectations.', 20, 14);
```

REVIEW : Test Case : Calculate the average maximum number of children for each star rating.

The queries use the connection between the review, listing and place to calculate the average maximum number of children for each star rating by using aggregation function and grouping.

```
SELECT star, AVG(max_children)
    FROM REVIEW r
JOIN LISTING l ON r.listing_id = l.listing_id
JOIN PLACE p ON p.place_id = l.place_id
    GROUP BY star;
```

star integer	avg numeric
3	4.0000000000000000
5	2.8333333333333333
4	2.5000000000000000
2	2.2500000000000000

SELECT Clause : It selects the star column and use the AVG aggregation function to evaluate the average maximum children.
FROM Clause : Specifies the tables (REVIEW, LISTING, PALCE) from which the data are retrieved, joining them on appropriate keys.
GROUP BY Clause : Groups the result set by the star column, enabling the calculation of average maximum children allowed for each star value.

TAGS

Store all the necessary information of tags associated with places within the system. This table is crucial for mapping tags to specific places, enhancing the categorization and searchability of listings.

```
CREATE TABLE TAGS(
    tags_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    tag_id INT NOT NULL,
    place_id INT NOT NULL,

    CONSTRAINT tag_fk FOREIGN KEY(tag_id) REFERENCES TAG(tag_id) ON DELETE CASCADE,
    CONSTRAINT place_fk FOREIGN KEY(place_id) REFERENCES PLACE(place_id) ON DELETE CASCADE
);
```

tags_id [PK] integer	tag_id integer	place_id integer
1	1	1
2	2	1
3	4	2
4	5	2
5	6	3
6	8	4
7	9	4
8	10	5
9	11	6
10	12	7
11	13	8
12	14	8
13	16	9
14	17	10
15	18	11
16	19	12
17	20	13
18	3	14
19	4	14
20	5	15
21	7	16
22	9	16

Using INSERT command to add all relevant information for each property.

```
INSERT INTO TAGS (tag_id, place_id)
VALUES (1, 1), (2, 1), (4, 2), (5, 2), (6, 3), (8, 4), (9, 4), (10, 5), (11, 6), (12, 7),
       (13, 8), (14, 8), (16, 9), (17, 10), (18, 11), (19, 12), (20, 13), (3, 14), (4, 14),
       (5, 15), (7, 16), (9, 16), (11, 17), (13, 18), (15, 18), (16, 19), (18, 20), (19, 20);
```

TAGS : Test Case : Count the number of places with exterior security cameras where security (person patrolling the place) is true.

The queries use the connection between the tags, tag, and place to count the number of places with exterior security cameras where security is true by using aggregation function and grouping. It also ensure the connection between tag and tags.

```
-- Count places with exterior security cameras and security enabled
SELECT exterior_security_camera, COUNT(*) FROM TAGS t
    JOIN TAG g ON g.tag_id = t.tag_id
    JOIN PLACE p ON p.place_id = t.place_id
WHERE p.security = true
    GROUP BY exterior_security_camera;
```

exterior_security_camera boolean	count bigint
false	4
true	13

SELECT Clause : It selects the exterior security camera column and the count the number of places with this feature.
FROM Clause : Specifies the tables (TAGS, TAG, PLACE) from which the data are retrieved, joining them on appropriate keys and filtering for places where security is true.
GROUP BY Clause : Groups the result set by the exterior_security_camera column, enabling the count of places in each subgroup.

TAG

Store all the necessary information of tags within the system. This table is crucial for defining different tags, including their names and descriptions.

```
CREATE TABLE TAG(
    tag_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    description TEXT NOT NULL
);
```

tag_id [PK] integer	name character varying (20)	description text
1	Beachfront	Properties located near or on the beach
2	City Center	Properties located in the heart of the city
3	Mountain View	Properties with stunning mountain views
4	Pet-Friendly	Properties that allow pets
5	Luxury	High-end, luxurious properties
6	Budget	Affordable, budget-friendly properties
7	Historic	Properties with historical significance
8	Secluded	Remote, secluded properties
9	Family-Friendly	Properties suitable for families with children
10	Romantic	Properties perfect for romantic getaways
11	Eco-Friendly	Environmentally-friendly, sustainable properties
12	Ski Resort	Properties located near or within a ski resort
13	Waterfront	Properties located near or on a body of water
14	Countryside	Properties situated in rural, countryside settings
15	Modern	Properties with contemporary, modern design
16	Rustic	Properties with a rustic, traditional style
17	Luxury Villa	High-end, luxurious villa properties
18	Cabin	Cozy, cabin-style properties
19	Apartment	Properties in apartment buildings or complexes
20	Hostel	Budget-friendly hostel accommodations

Using INSERT command to add all relevant information for each property.

```
INSERT INTO TAG (name, description)
VALUES ('Beachfront', 'Properties located near or on the beach'),
       ('City Center', 'Properties located in the heart of the city'),
       ('Mountain View', 'Properties with stunning mountain views'),
       ('Pet-Friendly', 'Properties that allow pets'),
       ('Luxury', 'High-end, luxurious properties'),
       ('Budget', 'Affordable, budget-friendly properties'),
       ('Historic', 'Properties with historical significance'),
       ('Secluded', 'Remote, secluded properties'),
       ('Family-Friendly', 'Properties suitable for families with children'),
       ('Romantic', 'Properties perfect for romantic getaways'),
       ('Eco-Friendly', 'Environmentally-friendly, sustainable properties'),
       ('Ski Resort', 'Properties located near or within a ski resort'),
       ('Waterfront', 'Properties located near or on a body of water'),
       ('Countryside', 'Properties situated in rural, countryside settings'),
       ('Modern', 'Properties with contemporary, modern design'),
       ('Rustic', 'Properties with a rustic, traditional style'),
       ('Luxury Villa', 'High-end, luxurious villa properties'),
       ('Cabin', 'Cozy, cabin-style properties'),
       ('Apartment', 'Properties in apartment buildings or complexes'),
       ('Hostel', 'Budget-friendly hostel accommodations');
```

TAG : Test Case : Count the number of places that allow pets where the maximum number of adult is greater than 2.

The queries use the connection between the tag, tags, and place to count the number of places that allow pets where the number of maximum adult is greater than 2 by using aggregation function and grouping.

```sql
-- Count places allowing pets with more than 2 adults
SELECT allowed_pet, COUNT(*) FROM TAG t
    JOIN TAGS g ON t.tag_id = g.tag_id
    JOIN PLACE p ON p.place_id = g.place_id
WHERE max_adult > 2
    GROUP BY allowed_pet;
```

allowed_pet boolean	count bigint
false	10
true	13

SELECT Clause : It selects the allowed pet column and the COUNT of places with this feature.
FROM Clause : Specifies the tables (TAG, TAGS, PLACE) from which the data are retrieved, joining them on appropriate keys and filtering for places where maximum number of adult is greater than 2.
GROUP BY Clause : Groups the result set by the allowed pet column, enabling the count of places in each subgroup.

USER

Store all the necessary information of administration that occurs within the system. This table is crucial for mapping administrators to hosts and guests, ensuring proper management and oversight.

```
CREATE TABLE "USER"(
    user_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    administrator_id INT NOT NULL,
    host_id INT NOT NULL,
    guest_id INT NOT NULL,

    CONSTRAINT administrator_fk FOREIGN KEY(administrator_id) REFERENCES ADMINISTRATOR(administrator_id) ON DELETE CASCADE,
    CONSTRAINT host_fk FOREIGN KEY(host_id) REFERENCES HOST(host_id) ON DELETE CASCADE,
    CONSTRAINT guest_fk FOREIGN KEY(guest_id) REFERENCES GUEST(guest_id) ON DELETE CASCADE

);
```

user_id [PK] integer	administrator_id integer	host_id integer	guest_id integer
1	1	1	1
2	1	2	2
3	1	3	3
4	2	4	4
5	2	5	5
6	3	6	6
7	3	7	7
8	3	8	8
9	4	9	9
10	4	10	10
11	5	11	11
12	5	12	12
13	5	13	13
14	6	14	14
15	6	15	15
16	7	16	16
17	7	17	17
18	8	18	18
19	8	19	19
20	8	20	20

Using INSERT command to add all relevant information for each administration.

```
INSERT INTO "USER" (administrator_id, host_id, guest_id)
VALUES (1, 1, 1),
       (1, 2, 2),
       (1, 3, 3),
       (2, 4, 4),
       (2, 5, 5),
       (3, 6, 6),
       (3, 7, 7),
       (3, 8, 8),
       (4, 9, 9),
       (4, 10, 10),
       (5, 11, 11),
       (5, 12, 12),
       (5, 13, 13),
       (6, 14, 14),
       (6, 15, 15),
       (7, 16, 16),
       (7, 17, 17),
       (8, 18, 18),
       (8, 19, 19),
       (8, 20, 20);
```

USER : Test Case : Count the number of user (administration) assigned to each administrators by their first name.

The queries use the connection between the user, host, and administrator to count the number of hosts managed by each administrator by first name by using aggregation function and grouping.

```
-- Count administrators by first name
SELECT ad.first_name, COUNT(*)
FROM "USER" u
    JOIN ADMINISTRATOR ad ON ad.administrator_id = u.administrator_id
    JOIN HOST h ON u.host_id = h.host_id
GROUP BY ad.first_name;
```

first_name character varying (30)	count bigint
Lala2	2
Lala1	3
Lala5	3
Lala3	3
Lala6	2
Lala4	2
Lala8	3
Lala7	2

SELECT Clause : It selects the first name column of the administrator and the count the number of hosts or administration managed by each administrator.
FROM Clause : Specifies the tables (USER, HOST, ADMINISTRATOR) from which the data are retrieved, joining them on appropriate keys.
GROUP BY Clause : Groups the result set by the first name column, enabling the count of hosts in each subgroup.

ADMINISTRATOR

Store all the necessary information of administrators within the system. This table is crucial for managing administrator details, including personal information and login credentials.

```
CREATE TABLE ADMINISTRATOR(
    administrator_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    email VARCHAR(75) NOT NULL,
    password VARCHAR(35) NOT NULL
);
```

administrator_id [PK] integer	first_name character varying (30)	last_name character varying (30)	email character varying (75)	password character varying (35)
1	Lala1	Nuna1	lala1@gmail.com	lala1
2	Lala2	Nuna2	lala2@gmail.com	lala2
3	Lala3	Nuna3	lala3@gmail.com	lala3
4	Lala4	Nuna4	lala4@gmail.com	lala4
5	Lala5	Nuna5	lala5@gmail.com	lala5
6	Lala6	Nuna6	lala6@gmail.com	lala6
7	Lala7	Nuna7	lala7@gmail.com	lala7
8	Lala8	Nuna8	lala8@gmail.com	lala8
9	Lala9	Nuna9	lala9@gmail.com	lala9
10	Lala10	Nuna10	lala10@gmail.com	lala10
11	Lala11	Nuna11	lala11@gmail.com	lala11
12	Lala12	Nuna12	lala12@gmail.com	lala12
13	Lala13	Nuna13	lala13@gmail.com	lala13
14	Lala14	Nuna14	lala14@gmail.com	lala14
15	Lala15	Nuna15	lala15@gmail.com	lala15
16	Lala16	Nuna16	lala16@gmail.com	lala16
17	Lala17	Nuna17	lala17@gmail.com	lala17
18	Lala18	Nuna18	lala18@gmail.com	lala18
19	Lala19	Nuna19	lala19@gmail.com	lala19
20	Lala20	Nuna20	lala20@gmail.com	lala20

Using INSERT command to add all relevant information for each administrator.

```
INSERT INTO ADMINISTRATOR (first_name, last_name, email, password) VALUES
    ('Lala1', 'Nuna1', 'lala1@gmail.com', 'lala1'),
    ('Lala2', 'Nuna2', 'lala2@gmail.com', 'lala2'),
    ('Lala3', 'Nuna3', 'lala3@gmail.com', 'lala3'),
    ('Lala4', 'Nuna4', 'lala4@gmail.com', 'lala4'),
    ('Lala5', 'Nuna5', 'lala5@gmail.com', 'lala5'),
    ('Lala6', 'Nuna6', 'lala6@gmail.com', 'lala6'),
    ('Lala7', 'Nuna7', 'lala7@gmail.com', 'lala7'),
    ('Lala8', 'Nuna8', 'lala8@gmail.com', 'lala8'),
    ('Lala9', 'Nuna9', 'lala9@gmail.com', 'lala9'),
    ('Lala10', 'Nuna10', 'lala10@gmail.com', 'lala10'),
    ('Lala11', 'Nuna11', 'lala11@gmail.com', 'lala11'),
    ('Lala12', 'Nuna12', 'lala12@gmail.com', 'lala12'),
    ('Lala13', 'Nuna13', 'lala13@gmail.com', 'lala13'),
    ('Lala14', 'Nuna14', 'lala14@gmail.com', 'lala14'),
    ('Lala15', 'Nuna15', 'lala15@gmail.com', 'lala15'),
    ('Lala16', 'Nuna16', 'lala16@gmail.com', 'lala16'),
    ('Lala17', 'Nuna17', 'lala17@gmail.com', 'lala17'),
    ('Lala18', 'Nuna18', 'lala18@gmail.com', 'lala18'),
    ('Lala19', 'Nuna19', 'lala19@gmail.com', 'lala19'),
    ('Lala20', 'Nuna20', 'lala20@gmail.com', 'lala20');
```

ADMINISTRATOR : Test Case : Count the number of user (administrations) assigned to each administrators by their last name.

The queries use the connection between the administrator, user, and host to count the number of administrators by their last name. by using aggregation function and grouping.

```
-- Count administrators by last name
SELECT a.last_name, COUNT(*)
FROM ADMINISTRATOR a
    JOIN "USER" u ON a.administrator_id = u.administrator_id
    JOIN HOST h ON u.host_id = h.host_id
GROUP BY a.last_name;
```

last_name character varying (30)	count bigint
Nuna7	2
Nuna4	2
Nuna8	3
Nuna5	3
Nuna6	2
Nuna1	3
Nuna3	3
Nuna2	2

SELECT Clause : It selects the last name column of the administrator and count the number of administrations assigned to each administrators with each last name.
FROM Clause : Specifies the tables (ADMINISTRATOR, USER, HOST) from which the data are retrieved, joining them on appropriate keys.
GROUP BY Clause : Groups the result set by the last name column, enabling the count of administrators in each subgroup.

BOOKING

Store all the necessary information of bookings within the system. This table is crucial for managing booking details, including guest information, listing details, and all other details.

```
CREATE TABLE BOOKING(
    booking_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    guest_id INT NOT NULL,
    listing_id INT NOT NULL,
    check_in DATE NOT NULL,
    check_out DATE NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    booking_time TIMESTAMP NOT NULL,
    paid BOOLEAN NOT NULL,
    num_adult INT NOT NULL,
    num_children INT NOT NULL,
    payment_id INT NOT NULL,

    CONSTRAINT guest_fk FOREIGN KEY(guest_id) REFERENCES GUEST(guest_id) ON DELETE CASCADE,
    CONSTRAINT listing_fk FOREIGN KEY(listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE,
    CONSTRAINT payment_fk FOREIGN KEY(payment_id) REFERENCES PAYMENT(payment_id) ON DELETE CASCADE
);
```

booking_id [PK] integer	guest_id integer	listing_id integer	check_in date	check_out date	total_price numeric (10,2)	booking_time timestamp without time zone	paid boolean	num_adult integer	num_children integer	payment_id integer
1	1	1	2023-07-01	2023-07-07	1200.50	2023-06-15 10:30:00	true	2	1	20
2	2	2	2023-08-10	2023-08-15	2500.75	2023-07-20 14:45:00	true	4	2	19
3	3	3	2023-09-01	2023-09-05	850.00	2023-08-18 09:15:00	false	2	0	18
4	4	4	2023-10-15	2023-10-20	1750.25	2023-09-28 16:20:00	true	3	1	17
5	5	5	2023-11-01	2023-11-07	2100.00	2023-10-10 11:35:00	false	4	0	16
6	6	6	2023-12-20	2023-12-27	1450.75	2023-11-25 18:10:00	true	2	2	15
7	7	7	2024-01-01	2024-01-08	2800.50	2023-12-15 08:25:00	true	4	1	14
8	8	8	2024-02-10	2024-02-15	1200.00	2024-01-20 13:40:00	false	2	0	13
9	9	9	2024-03-15	2024-03-20	1650.25	2024-02-28 10:55:00	true	3	1	12
10	10	10	2024-04-01	2024-04-07	2400.75	2024-03-15 14:20:00	true	4	2	11
11	11	11	2024-05-10	2024-05-15	1100.50	2024-04-25 09:35:00	false	2	0	10
12	12	12	2024-06-01	2024-06-08	2200.75	2024-05-15 16:50:00	true	4	1	9
13	13	13	2024-07-15	2024-07-20	1450.00	2024-06-28 12:10:00	false	2	2	8
14	14	14	2024-08-01	2024-08-07	2850.25	2024-07-10 18:30:00	true	4	0	7
15	15	15	2024-09-10	2024-09-15	1700.50	2024-08-25 11:45:00	true	3	1	6
16	16	16	2024-10-01	2024-10-07	2100.75	2024-09-15 14:20:00	false	4	2	5
17	17	17	2024-11-15	2024-11-20	1300.00	2024-10-30 09:55:00	true	2	0	4
18	18	18	2024-12-20	2024-12-27	2600.25	2024-11-30 16:40:00	true	4	1	3
19	19	19	2025-01-01	2025-01-08	2900.50	2024-12-15 10:15:00	false	6	2	2
20	20	20	2025-02-10	2025-02-15	1400.75	2025-01-25 13:30:00	true	2	1	1

Using INSERT command to add all relevant information for each booking.

```
INSERT INTO BOOKING (guest_id, listing_id, check_in, check_out, total_price, booking_time, paid, num_adult, num_children,
VALUES (1, 1, '2023-07-01', '2023-07-07', 1200.50, '2023-06-15 10:30:00', true, 2, 1, 20),
       (2, 2, '2023-08-10', '2023-08-15', 2500.75, '2023-07-20 14:45:00', true, 4, 2, 19),
       (3, 3, '2023-09-01', '2023-09-05', 850.00, '2023-08-18 09:15:00', false, 2, 0, 18),
       (4, 4, '2023-10-15', '2023-10-20', 1750.25, '2023-09-28 16:20:00', true, 3, 1, 17),
       (5, 5, '2023-11-01', '2023-11-07', 2100.00, '2023-10-10 11:35:00', false, 4, 0, 16),
       (6, 6, '2023-12-20', '2023-12-27', 1450.75, '2023-11-25 18:10:00', true, 2, 2, 15),
       (7, 7, '2024-01-01', '2024-01-08', 2800.50, '2023-12-15 08:25:00', true, 4, 1, 14),
       (8, 8, '2024-02-10', '2024-02-15', 1200.00, '2024-01-20 13:40:00', false, 2, 0, 13),
       (9, 9, '2024-03-15', '2024-03-20', 1650.25, '2024-02-28 10:55:00', true, 3, 1, 12),
       (10, 10, '2024-04-01', '2024-04-07', 2400.75, '2024-03-15 14:20:00', true, 4, 2, 11),
       (11, 11, '2024-05-10', '2024-05-15', 1100.50, '2024-04-25 09:35:00', false, 2, 0, 10),
       (12, 12, '2024-06-01', '2024-06-08', 2200.75, '2024-05-15 16:50:00', true, 4, 1, 9),
       (13, 13, '2024-07-15', '2024-07-20', 1450.00, '2024-06-28 12:10:00', false, 2, 2, 8),
       (14, 14, '2024-08-01', '2024-08-07', 2850.25, '2024-07-10 18:30:00', true, 4, 0, 7),
       (15, 15, '2024-09-10', '2024-09-15', 1700.50, '2024-08-25 11:45:00', true, 3, 1, 6),
       (16, 16, '2024-10-01', '2024-10-07', 2100.75, '2024-09-15 14:20:00', false, 4, 2, 5),
       (17, 17, '2024-11-15', '2024-11-20', 1300.00, '2024-10-30 09:55:00', true, 2, 0, 4),
       (18, 18, '2024-12-20', '2024-12-27', 2600.25, '2024-11-30 16:40:00', true, 4, 1, 3),
       (19, 19, '2025-01-01', '2025-01-08', 2900.50, '2024-12-15 10:15:00', false, 6, 2, 2),
       (20, 20, '2025-02-10', '2025-02-15', 1400.75, '2025-01-25 13:30:00', true, 2, 1, 1);
```

BOOKING : Test Case : Count the number of bookings which has been paid or not paid with a total price greater than 2100.

The queries use the connection between the booking, guest, and listing to count the number of bookings with a total price greater than 2100, grouped by paid status 2000 using aggregation function and grouping. This ensure the connection between guest and booking.

```
-- Count bookings with total price greater than 2100
SELECT paid, COUNT(*)
FROM BOOKING b
    JOIN LISTING l ON b.listing_id = l.listing_id
    JOIN GUEST g ON g.guest_id = b.guest_id
WHERE b.total_price > 2100
GROUP BY paid;
```

paid boolean	count bigint
false	2
true	6

SELECT Clause : It selects the paid column and the COUNT of bookings with a total price greater than 2100.
FROM Clause : Specifies the tables (BOOKING, GUEST, LISTING) from which the data are retrieved, joining them on appropriate keys and filtering for bookings with a total price greater than 2100.
GROUP BY Clause : Groups the result set by the paid column, enabling the count of bookings in each subgroup.

PAYMENT

Store all the necessary information of payments within the system. This table is crucial for recording payment details, including person involved, person type, and payment method.

```
CREATE TABLE PAYMENT(
    payment_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    person_id INT NOT NULL,
    person_type VARCHAR(10) NOT NULL CHECK (person_type IN ('host','guest')),
    payment_type VARCHAR(15) NOT NULL
);
```

booking_id [PK] integer	guest_id integer	listing_id integer	check_in date	check_out date	total_price numeric (10,2)	booking_time timestamp without time zone	paid boolean	num_adult integer	num_children integer	payment_id integer
1	1	1	2023-07-01	2023-07-07	1200.50	2023-06-15 10:30:00	true	2	1	20
2	2	2	2023-08-10	2023-08-15	2500.75	2023-07-20 14:45:00	true	4	2	19
3	3	3	2023-09-01	2023-09-05	850.00	2023-08-18 09:15:00	false	2	0	18
4	4	4	2023-10-15	2023-10-20	1750.25	2023-09-28 16:20:00	true	3	1	17
5	5	5	2023-11-01	2023-11-07	2100.00	2023-10-10 11:35:00	false	4	0	16
6	6	6	2023-12-20	2023-12-27	1450.75	2023-11-25 18:10:00	true	2	2	15
7	7	7	2024-01-01	2024-01-08	2800.50	2023-12-15 08:25:00	true	4	1	14
8	8	8	2024-02-10	2024-02-15	1200.00	2024-01-20 13:40:00	false	2	0	13
9	9	9	2024-03-15	2024-03-20	1650.25	2024-02-28 10:55:00	true	3	1	12
10	10	10	2024-04-01	2024-04-07	2400.75	2024-03-15 14:20:00	true	4	2	11
11	11	11	2024-05-10	2024-05-15	1100.50	2024-04-25 09:35:00	false	2	0	10
12	12	12	2024-06-01	2024-06-08	2200.75	2024-05-15 16:50:00	true	4	1	9
13	13	13	2024-07-15	2024-07-20	1450.00	2024-06-28 12:10:00	false	2	2	8
14	14	14	2024-08-01	2024-08-07	2850.25	2024-07-10 18:30:00	true	4	0	7
15	15	15	2024-09-10	2024-09-15	1700.50	2024-08-25 11:45:00	true	3	1	6
16	16	16	2024-10-01	2024-10-07	2100.75	2024-09-15 14:20:00	false	4	2	5
17	17	17	2024-11-15	2024-11-20	1300.00	2024-10-30 09:55:00	true	2	0	4
18	18	18	2024-12-20	2024-12-27	2600.25	2024-11-30 16:40:00	true	4	1	3
19	19	19	2025-01-01	2025-01-08	2900.50	2024-12-15 10:15:00	false	6	2	2
20	20	20	2025-02-10	2025-02-15	1400.75	2025-01-25 13:30:00	true	2	1	1

Using INSERT command to add all relevant information for each payment.

```
INSERT INTO PAYMENT (person_id, person_type, payment_type) VALUES
    (1, 'host', 'upi'),(2, 'guest', 'upi'),(3, 'host', 'upi'),
    (4, 'guest', 'upi'),
    (5, 'host', 'upi'),
    (6, 'guest', 'credit_debit'),
    (7, 'host', 'credit_debit'),
    (8, 'guest', 'credit_debit'),
    (9, 'host', 'credit_debit'),
    (10, 'guest', 'credit_debit'),
    (11, 'host', 'net_banking'),
    (12, 'guest', 'net_banking'),
    (13, 'host', 'net_banking'),
    (14, 'guest', 'net_banking'),
    (15, 'host', 'net_banking'),
    (16, 'guest', 'upi'),
    (17, 'host', 'upi'),
    (18, 'guest', 'upi'),
    (19, 'host', 'upi'),
    (20, 'guest', 'upi'),
    (21, 'host', 'credit_debit'),
    (22, 'guest', 'credit_debit'),
    (23, 'host', 'credit_debit'),
    (24, 'guest', 'credit_debit'),
    (25, 'host', 'credit_debit'),
    (26, 'guest', 'net_banking'),
    (27, 'host', 'net_banking'),
    (28, 'guest', 'net_banking'),
```

PAYMENT : Test Case : Among the entire payment, retrieve UPI IDs and associated guests ID.

The queries use the connection between the payment, guest, and UPI to retrieve UPI IDs and associated person IDs for guests.

```
-- List UPI IDs and person IDs for guest payments
SELECT person_id, upi
FROM PAYMENT p
    JOIN UPI u ON p.payment_id = u.payment_method_id
    JOIN GUEST g ON g.guest_id = p.person_id AND p.person_type = 'guest
WHERE p.person_type = 'guest';
```

person_id integer	upi character varying (30)
2	guest2@upi
4	guest4@upi
16	guest16@upi
18	guest18@upi
20	guest20@upi

SELECT Clause : It selects the person_id and upi_id columns.

FROM Clause : Specifies the tables (PAYMENT, GUEST, UPI) from which the data are retrieved, joining them on appropriate keys and filtering for guests.

WHERE Clause : Restrict the result set to only guest.

UPI

Store all the necessary information of UPI payments within the system. This table is crucial for managing UPI payment details, including payment method ID and UPI ID.

```
CREATE TABLE UPI(
    upi_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    payment_method_id INT NOT NULL,
    upi VARCHAR(30) NOT NULL,

    CONSTRAINT payment_method_fk FOREIGN KEY(payment_method_id) REFERENCES PAYMENT(payment_id) ON DELETE CASCADE
);
```

upi_id [PK] integer	payment_method_id integer	upi character varying (30)
1	1	user1@upi
2	2	guest2@upi
3	3	host3@upi
4	4	guest4@upi
5	5	host5@upi
6	16	guest16@upi
7	17	host17@upi
8	18	guest18@upi
9	19	host19@upi
10	20	guest20@upi
11	31	host31@upi
12	32	guest32@upi
13	33	host33@upi
14	34	guest34@upi
15	35	host35@upi
16	46	guest46@upi
17	47	host47@upi
18	48	guest48@upi
19	49	host49@upi
20	50	guest50@upi

Using INSERT command to add all relevant information for each UPI.

```
INSERT INTO UPI (payment_method_id, upi)
VALUES (1, 'user1@upi'), (2, 'guest2@upi'), (3, 'host3@upi'), (4, 'guest4@upi'), (5, 'host5@upi'),
    (16, 'guest16@upi'), (17, 'host17@upi'), (18, 'guest18@upi'), (19, 'host19@upi'), (20, 'guest20@upi'),
    (31, 'host31@upi'), (32, 'guest32@upi'), (33, 'host33@upi'), (34, 'guest34@upi'), (35, 'host35@upi'),
    (46, 'guest46@upi'), (47, 'host47@upi'), (48, 'guest48@upi'), (49, 'host49@upi'), (50, 'guest50@upi');
```

UPI : Test Case : Retrieve UPI IDs, person IDs, and location IDs for hosts.

The queries use the connection between the UPI, payment, and host to retrieve UPI IDs, person IDs, and location IDs for hosts. This also ensure the connection from upi to host to location.

```sql
-- List UPI IDs, person IDs, and location IDs for host payments
SELECT u.upi, p.person_id, h.location_id
FROM UPI u
    JOIN PAYMENT p ON u.payment_method_id = p.payment_id
    JOIN HOST h ON h.host_id = p.person_id
WHERE p.person_type = 'host';
```

upi character varying (30)	person_id integer	location_id integer
user1@upi	1	4
host3@upi	3	14
host5@upi	5	1
host17@upi	17	9
host19@upi	19	20

SELECT Clause : It selects the upi_id, person_id, and location_id columns.

FROM Clause : Specifies the tables (UPI, PAYMENT, HOST) from which the data are retrieved, joining them on appropriate keys and filtering for hosts.

WHERE Clause : Restricts the result set to only host.

CREDIT_DEBIT_CARD

Store all the necessary information of credit and debit card payments within the system. This table is crucial for managing card payment details, including card number, holder name, and expiration date.

```
CREATE TABLE CREDIT_DEBIT_CARD(
    credit_debit_card_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    payment_method_id INT NOT NULL,
    card_number VARCHAR(20) NOT NULL,
    card_holder_name VARCHAR(45) NOT NULL,
    expiration_date DATE NOT NULL,
    cvv INT NOT NULL,

    CONSTRAINT payment_method_fk FOREIGN KEY(payment_method_id) REFERENCES PAYMENT(payment_id) ON DELETE CASCADE
);
```

credit_debit_card_id [PK] integer	payment_method_id integer	card_number character varying (20)	card_holder_name character varying (45)	expiration_date date	cvv integer
1	6	1234567890123456	Guest 6	2025-12-31	123
2	7	2345678901234567	Host 7	2026-06-30	456
3	8	3456789012345678	Guest 8	2024-09-30	789
4	9	4567890123456789	Host 9	2027-03-31	101
5	10	5678901234567890	Guest 10	2025-07-31	112
6	21	6789012345678901	Host 21	2026-11-30	213
7	22	7890123456789012	Guest 22	2024-02-28	314
8	23	8901234567890123	Host 23	2027-08-31	415
9	24	9012345678901234	Guest 24	2025-04-30	516
10	25	0123456789012345	Host 25	2026-10-31	617
11	36	1234567890123456	Guest 36	2025-01-31	718
12	37	2345678901234567	Host 37	2027-05-31	819
13	38	3456789012345678	Guest 38	2024-12-31	920
14	39	4567890123456789	Host 39	2026-03-31	21
15	40	5678901234567890	Guest 40	2025-09-30	122
16	51	6789012345678901	Host 51	2027-02-28	223
17	52	7890123456789012	Guest 52	2024-07-31	324
18	53	8901234567890123	Host 53	2026-01-31	425
19	54	9012345678901234	Guest 54	2025-06-30	526
20	55	0123456789012345	Host 55	2027-12-31	627

Using INSERT command to add all relevant information for each credit_debit_card.

```
INSERT INTO CREDIT_DEBIT_CARD (payment_method_id, card_number, card_holder_name, expiration_date, cvv)
VALUES (6, '1234567890123456', 'Guest 6', '2025-12-31', 123), (7, '2345678901234567', 'Host 7', '2026-06-30', 456),
       (8, '3456789012345678', 'Guest 8', '2024-09-30', 789), (9, '4567890123456789', 'Host 9', '2027-03-31', 101),
       (10, '5678901234567890', 'Guest 10', '2025-07-31', 112), (21, '6789012345678901', 'Host 21', '2026-11-30', 213),
       (22, '7890123456789012', 'Guest 22', '2024-02-28', 314), (23, '8901234567890123', 'Host 23', '2027-08-31', 415),
       (24, '9012345678901234', 'Guest 24', '2025-04-30', 516), (25, '0123456789012345', 'Host 25', '2026-10-31', 617),
       (36, '1234567890123456', 'Guest 36', '2025-01-31', 718), (37, '2345678901234567', 'Host 37', '2027-05-31', 819),
       (38, '3456789012345678', 'Guest 38', '2024-12-31', 920), (39, '4567890123456789', 'Host 39', '2026-03-31', 021),
       (40, '5678901234567890', 'Guest 40', '2025-09-30', 122), (51, '6789012345678901', 'Host 51', '2027-02-28', 223),
       (52, '7890123456789012', 'Guest 52', '2024-07-31', 324), (53, '8901234567890123', 'Host 53', '2026-01-31', 425),
       (54, '9012345678901234', 'Guest 54', '2025-06-30', 526), (55, '0123456789012345', 'Host 55', '2027-12-31', 627);
```

CREDIT_DEBIT_CARD : Retrieve card numbers, person IDs for hosts, and location IDs.

The queries use the connection between the credit/debit card, payment, and host to retrieve card numbers, person IDs, and location IDs for hosts. This also ensure the connection from credit debit card to host to location.

```
-- List credit/debit card details for host payments
SELECT u.card_number, p.person_id, h.location_id
FROM CREDIT_DEBIT_CARD u
    JOIN PAYMENT p ON u.payment_method_id = p.payment_id
    JOIN HOST h ON h.host_id = p.person_id
WHERE p.person_type = 'host';
```

card_number character varying (20)	person_id integer	location_id integer
2345678901234567	7	2
4567890123456789	9	3

SELECT Clause : It selects the card_number, person_id, and location_id columns.

FROM Clause : Specifies the tables (CREDIT_DEBIT_CARD, PAYMENT, HOST) from which the data are retrieved, joining them on appropriate keys and filtering for hosts.

WHERE Clause : Restrict the result set to host.

NET_ BANKING

Store all the necessary information of net banking payments within the system. This table is crucial for managing net banking details, including bank name, account number, and IFSC code.

```
CREATE TABLE NET_BANKING(
    net_banking_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    payment_method_id INT NOT NULL,
    bank_name VARCHAR(30) NOT NULL,
    account_number VARCHAR(25) NOT NULL,
    ifsc_code VARCHAR(25) NOT NULL,

    CONSTRAINT payment_method_fk FOREIGN KEY(payment_method_id) REFERENCES PAYMENT(payment_id) ON DELETE CASCADE
);
```

net_banking_id [PK] integer	payment_method_id integer	bank_name character varying (30)	account_number character varying (25)	ifsc_code character varying (25)
1	11	Bank A	1234567890	BNKA0001234
2	12	Bank B	2345678901	BNKB0002345
3	13	Bank C	3456789012	BNKC0003456
4	14	Bank D	4567890123	BNKD0004567
5	15	Bank E	5678901234	BNKE00056124
6	26	Bank S	6789012345	BNKF0001231
7	41	Bank AS	5673201234	BNKE00051248
8	27	Bank S	67812123445	BNKF0001212
9	42	Bank CA	52348912234	BNKE0005621
10	28	Bank AS	8834012345	BNKF0001231
11	43	Bank QWE	1178901234	BNKE000567234
12	29	Bank FGA	97834012345	BNKF0004431
13	44	Bank QWE	8678901234	BNKE00051248
14	30	Bank ASD	2712332412345	BNKF0005531
15	45	Bank RG	9778901234	BNKE000523478
16	56	Bank SD	271233412345	BNKF000234531
17	57	Bank EI	3478901234	BNKE0001278
18	58	Bank AD	27812312345	BNKF00055234
19	59	Bank LE	7678901234	BNKE0005128
20	60	Bank A	27123412345	BNKF00055234

Using INSERT command to add all relevant information for each Net Banking.

```
INSERT INTO NET_BANKING (payment_method_id, bank_name, account_number, ifsc_code)
VALUES (11, 'Bank A', '1234567890', 'BNKA0001234'), (12, 'Bank B', '2345678901', 'BNKB0002345'),
       (13, 'Bank C', '3456789012', 'BNKC0003456'), (14, 'Bank D', '4567890123', 'BNKD0004567'),
       (15, 'Bank E', '5678901234', 'BNKE00056124'), (26, 'Bank S', '6789012345', 'BNKF0001231'),
       (41, 'Bank AS', '5673201234', 'BNKE00051248'), (27, 'Bank S', '67812123445', 'BNKF0001212'),
       (42, 'Bank CA', '52348912234', 'BNKE0005621'), (28, 'Bank AS', '8834012345', 'BNKF0001231'),
       (43, 'Bank QWE', '1178901234', 'BNKE000567234'), (29, 'Bank FGA', '97834012345', 'BNKF0004431'),
       (44, 'Bank QWE', '8678901234', 'BNKE00051248'), (30, 'Bank ASD', '2712332412345', 'BNKF0005531'),
       (45, 'Bank RG', '9778901234', 'BNKE000523478'), (56, 'Bank SD', '271233412345', 'BNKF000234531'),
       (57, 'Bank EI', '3478901234', 'BNKE0001278'), (58, 'Bank AD', '27812312345', 'BNKF00055234'),
       (59, 'Bank LE', '7678901234', 'BNKE0005128'), (60, 'Bank A', '27123412345', 'BNKF00055234');
```

NET BANKING : Test Case : Retrieve bank name, person IDs for hosts, and location IDs.

The queries use the connection between the net banking, location, and host to retrieve bank name, person IDs, and location IDs for hosts. This also person the connection from net banking card to host to location.

```
-- List net banking details for host payments
SELECT u.bank_name, p.person_id, h.location_id
FROM NET_BANKING u
    JOIN PAYMENT p ON u.payment_method_id = p.payment_id
    JOIN HOST h ON h.host_id = p.person_id
WHERE p.person_type = 'host';
```

bank_name character varying (30)	person_id integer	location_id integer
Bank A	11	6
Bank C	13	7
Bank E	15	8

SELECT Clause : It selects the bank_name, person_id, and location_id columns.

FROM Clause : Specifies the tables (NET_BANKING, PAYMENT, HOST) from which the data are retrieved, joining them on appropriate keys and filtering for hosts.

WHERE Clause : Restrict the result set to host.

COMPLAIN

Store all the necessary information of complaints within the system. This table is crucial for managing complaints, including user ID (administration ID), type of complaint, and description.

```
CREATE TABLE COMPLAIN(
    complain_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    user_id INT NOT NULL,
    complain_type VARCHAR(30) NOT NULL,
    description TEXT NOT NULL,

    CONSTRAINT user_fk FOREIGN KEY (user_id) REFERENCES "USER"(user_id) ON DELETE CASCADE
);
```

complain_id [PK] integer	user_id integer	complain_type character varying (30)	description text
1	1	maintenance	Air conditioner not working properly in room 101.
2	2	noise	Loud music coming from the bar area after midnight.
3	3	billing	Incorrect charges applied to my recent bill.
4	4	cleaning	Room service not cleaning the room properly.
5	5	amenities	Broken swimming pool needs repair.
6	6	maintenance	Leaking faucet in the bathroom sink.
7	7	staff	Unfriendly staff at the front desk.
8	8	other	Lost my phone in the hotel lobby.
9	9	food	Found a hair in my food at the restaurant.
10	10	internet	Wi-Fi connection is slow and unreliable.
11	11	maintenance	Light bulb burnt out in the hallway.
12	12	security	Security concerns about access to the hotel floors.
13	13	noise	Thin walls, can hear noise from neighboring rooms.
14	14	billing	Double charges for a service not used.
15	15	cleaning	Dirty towels left in the bathroom.
16	16	amenities	Gym equipment not functioning properly.
17	17	maintenance	Broken shower head in the bathroom.
18	18	staff	Rude behavior from a staff member at the restaurant.
19	19	other	Found suspicious activity happening near my room.
20	20	food	Wrong order delivered by room service.

Using INSERT command to add all relevant information for each complain.

```
INSERT INTO COMPLAIN (user_id, complain_type, description)
VALUES
  (1, 'maintenance', 'Air conditioner not working properly in room 101.'),
  (2, 'noise', 'Loud music coming from the bar area after midnight.'),
  (3, 'billing', 'Incorrect charges applied to my recent bill.'),
  (4, 'cleaning', 'Room service not cleaning the room properly.'),
  (5, 'amenities', 'Broken swimming pool needs repair.'),
  (6, 'maintenance', 'Leaking faucet in the bathroom sink.'),
  (7, 'staff', 'Unfriendly staff at the front desk.'),
  (8, 'other', 'Lost my phone in the hotel lobby.'),
  (9, 'food', 'Found a hair in my food at the restaurant.'),
  (10, 'internet', 'Wi-Fi connection is slow and unreliable.'),
  (11, 'maintenance', 'Light bulb burnt out in the hallway.'),
  (12, 'security', 'Security concerns about access to the hotel floors.'),
  (13, 'noise', 'Thin walls, can hear noise from neighboring rooms.'),
  (14, 'billing', 'Double charges for a service not used.'),
  (15, 'cleaning', 'Dirty towels left in the bathroom.'),
  (16, 'amenities', 'Gym equipment not functioning properly.'),
  (17, 'maintenance', 'Broken shower head in the bathroom.'),
  (18, 'staff', 'Rude behavior from a staff member at the restaurant.'),
  (19, 'other', 'Found suspicious activity happening near my room.'),
  (20, 'food', 'Wrong order delivered by room service.');
```

COMPLAIN : Test Case : Test Case : Count the number of complaints grouped by complain type.

The queries use the connection between the complain, user, and host to count the number of complaints grouped by complain type by using aggregation function and grouping. This also ensure the connection from complain to host to user.

```
-- Count complaints by type
SELECT complain_type, COUNT(*)
FROM COMPLAIN c
    JOIN "USER" u ON c.user_id = u.user_id
    JOIN HOST h ON h.host_id = u.host_id
GROUP BY complain_type;
```

complain_type character varying (30)	count bigint
security	1
food	2
noise	2
internet	1
staff	2
billing	2
amenities	2
cleaning	2
maintenance	4
other	2

SELECT Clause : It selects the complain_type column and the COUNT of complaints.

FROM Clause : Specifies the tables (COMPLAIN, USER, HOST) from which the data are retrieved, joining them on appropriate keys.

GROUP BY Clause : Groups the result set by the complain_type column, enabling the count of complaints in each subgroup.

LISTING_HOSTING

Link each host to their listing id where a host can have many listing. This table is crucial for managing each host individuals with their listing.

```
CREATE TABLE HOST_LISTING(
    host_listing_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    host_id INT NOT NULL,
    listing_id INT NOT NULL,

    CONSTRAINT listing_fk FOREIGN KEY (listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE,
    CONSTRAINT host_fk FOREIGN KEY (host_id) REFERENCES HOST(host_id) ON DELETE CASCADE
);
```

Using INSERT command to add all relevant information for each complain.

```
INSERT INTO HOST_LISTING(host_id,listing_id)
VALUES (1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10),
    (11,11),(12,12),(13,13),(14,14),(15,15),(16,16),(17,17),(18,18),(19,19),(20,20);
```

host_listing_id [PK] integer	host_id integer	listing_id integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

LISTING_HOSTING : Test Case : Count the number of listing each host have.

The queries use the connection between the host, listing, and host_listing to count the number of number of listing each host have by using aggregation function and grouping.

host_id integer	count bigint
11	1
9	1
15	1
19	1
3	1
17	1
5	1
4	1
10	1
6	1
14	1
13	1
2	1
16	1
7	1
12	1
20	1
1	1
18	1
8	1

```
SELECT hl.host_id, COUNT(*)
FROM HOST_LISTING hl
    JOIN HOST h ON hl.host_id = h.host_id
    JOIN LISTING l ON hl.listing_id = l.listing_id
GROUP BY hl.host_id;
```

SELECT Clause : It selects the host_id column and the COUNT of listing.

FROM Clause : Specifies the tables (HOST_LISTING, LISTING, HOST) from which the data are retrieved, joining them on appropriate keys.

GROUP BY Clause : Groups the result set by the host_id column, enabling the count of complaints in each subgroup.

LISTING_AMENITIES

Link each listing with their corresponding amenities within the system. This table is crucial for managing listing and amenities relationship.

```
CREATE TABLE LISTING_AMENITIES(
    listing_amenities_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    listing_id INT NOT NULL,
    amenities_id INT NOT NULL,

    CONSTRAINT listing_fk FOREIGN KEY (listing_id) REFERENCES LISTING(listing_id) ON DELETE CASCADE,
    CONSTRAINT amenities_fk FOREIGN KEY (amenities_id) REFERENCES AMENITIES(amenities_id) ON DELETE CASCAD
);
```

listing_amenities_id [PK] integer	listing_id integer	amenities_id integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Using INSERT command to add all relevant information for each complain.

```
INSERT INTO LISTING_AMENITIES (listing_id, amenities_id)
VALUES (1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10),
       (11,11),(12,12),(13,13),(14,14),(15,15),(16,16),(17,17),(18,18),(19,19),(20,20);
```

LISTING_AMENITIES : Test Case : Count the number of listing which has different number of bathrooms.

The queries use the connection between the listing, amenities, and listing_hosting to count the number of listing which has different number of bathrooms by using aggregation function and grouping.

```
SELECT a.num_bathrooms, COUNT(*)
FROM LISTING_AMENITIES la
    JOIN LISTING l ON l.listing_id = la.listing_id
    JOIN AMENITIES a ON a.amenities_id = la.amenities_id
GROUP BY a.num_bathrooms
```

num_bathrooms integer	count bigint
3	3
5	2
4	1
2	5
1	9

SELECT Clause : It selects the num_bathrooms column and the COUNT of listing with those number of bathrooms.

FROM Clause : Specifies the tables (AMENITIES, LISTING_AMENITIES, LISTING) from which the data are retrieved, joining them on appropriate keys.

GROUP BY Clause : Groups the result set by the num_bathrooms column, enabling the count of complaints in each subgroup.