Creating the database schema named as‘airbnb_database’:

The database comprises 20 interconnected tables, each meticulously crafted to capture specific entities and relationships within the Airbnb ecosystem:

User

In this slide, we'll create a user table for Airbnb is using SQL. This table will store information about different users that can have within the system.
Define the structure of the user table by utilizing the SQL statement CREATE TABLE.
Store and manage essential user information.
Use INSERT INTO statements to add user data to the table

```sql
CREATE TABLE airbnb.User (
  id INT PRIMARY KEY AUTO_INCREMENT,
  First_name VARCHAR(255) NOT NULL,
  Last_name VARCHAR(255) NOT NULL,
  Email VARCHAR(255) UNIQUE NOT NULL,
  Phone VARCHAR(255),
  Role VARCHAR(255),
  Bio TEXT,
  Profilepicture VARCHAR(255)
);

INSERT INTO airbnb.user (id, first_name, last_name, email, phone, role, Bio, ProfilePicture) VALUES
(1, 'William', 'Campbell', 'william.campbell@example.com', +65689741230, 'Host', 'Love to travel and explore new places! Always up for an adventure.', 'user1_profile.jpg'),
(2, 'Jane', 'Smith', 'jane.smith@example.com', '+447890123456', 'Host', 'City dweller who enjoys trying new restaurants and catching the latest shows.', 'user2_profile.png'),
(3, 'Maria', 'Garcia', 'maria.garcia@example.com', NULL, 'Admin','Nature enthusiast who loves hiking, camping, and spending time outdoors.', 'user3_profile.jpg'),
(4, 'Li', 'Wang', 'li.wang@example.com', '+861357924680', 'Traveler','Foodie who loves to cook and explore different cuisines.', 'user4_profile.bmp'),
(5, 'Alex', 'Schmidt', 'alex.schmidt@example.com', NULL, 'Host', 'Passionate about skiing and snowboarding. Always looking for the next powder run!', 'user5_profile.jpg'),
(6, 'Aisha', 'Khan', 'aisha.khan@example.com', '+919876543210', 'Guest','Wine connoisseur who enjoys relaxing at home with a glass of wine and a good book.', 'user6_profile.png'),
(7, 'David', 'Rodriguez', 'david.rodriguez@example.com', '+34654321098', 'Traveler', 'Beach bum who loves soaking up the sun and swimming in the ocean.', 'user7_profile.jpg'),
(8, 'Natalia', 'Petrova', 'natalia.petrova@example.com', '+79217890123', 'Host', 'Creative professional who enjoys art, music, and exploring the city.', 'user8_profile.bmp'),
(9, 'Omar', 'Syed', 'omar.syed@example.com', '+61412345678', 'Guest', 'Fishing enthusiast who loves spending time on the lake and catching the big one!', 'user9_profile.jpg'),
(10, 'Sophie', 'Dupont', 'sophie.dupont@example.com', '+33678901234', 'Traveler','Yoga instructor who enjoys spending time in nature and promoting mindfulness.', 'user10_profile.png'),
(11, 'Emily', 'Chen', 'emily.chen@example.com', NULL, 'Admin', 'Minimalist who appreciates simple living and experiences.', 'user11_profile.jpg'),
(12, 'William', 'Müller', 'william.muller@example.com', '+491573890456','Host', 'Big family who loves traveling together and making memories.', 'user12_profile.bmp'),
(13, 'Marie', 'Kim', 'marie.kim@example.com', '+821012345678', 'Host (Co-host)', 'Bookworm who loves getting lost in a good story.', 'user13_profile.jpg'),
(14, 'Miguel', 'Sanchez', 'miguel.sanchez@example.com', NULL, 'Guest', 'Winemaker who is passionate about creating delicious wines.', 'user14_profile.png'),
(15, 'Anna', 'Schmidt', 'anna.schmidt@example.com', '+491736258901', 'Traveler', 'Ski instructor who loves teaching others the joy of skiing.', 'user15_profile.jpg'),
(16, 'Ibrahim', 'Mohamed', 'REDACTED_EMAIL@example.com', NULL, 'Guest', 'Musician who loves playing music and sharing it with the world.', 'user16_profile.bmp'),
(17, 'Sarah', 'Lee', 'sarah.lee@example.com', '+1876543210', 'Host', 'History buff who loves exploring historical sites and learning about the past.', 'user17_profile.jpg'),
(18, 'Pierre', 'Martin', 'pierre.martin@example.com', '+33765432109', 'Traveler', 'Entrepreneur who is always looking for new opportunities and challenges.', 'user18_profile.png'),
(19, 'Elena', 'Volkova', 'elena.volkova@example.com', '+79154321087', 'Guest', 'Environmentalist who is passionate about protecting our planet.', 'user19_profile.jpg'),
(20,'Antonio', 'Garcia', 'antonio.garcia@example.com', '+34987654321', 'Host', 'Gamer who loves spending time playing video games and connecting with friends online.', 'user20_profile.bmp');
```

id	First_name	Last_name	Email	Phone	Role	Bio	Profilepicture
1	William	Campbell	william.campbell@example.com	65689741230	Host	Love to travel and explore new places! Always ...	user1_profile.jpg
2	Jane	Smith	jane.smith@example.com	+447890123456	Host	City dweller who enjoys trying new restaurants ...	user2_profile.png
3	Maria	Garcia	maria.garcia@example.com	NULL	Admin	Nature enthusiast who loves hiking, camping, a...	user3_profile.jpg
4	Li	Wang	li.wang@example.com	+861357924680	Traveler	Foodie who loves to cook and explore different ...	user4_profile.bmp
5	Alex	Schmidt	alex.schmidt@example.com	NULL	Host	Passionate about skiing and snowboarding. Alw...	user5_profile.jpg
6	Aisha	Khan	aisha.khan@example.com	+919876543210	Guest	Wine connoisseur who enjoys relaxing at home ...	user6_profile.png
7	David	Rodriguez	david.rodriguez@example.com	+34654321098	Traveler	Beach bum who loves soaking up the sun and s...	user7_profile.jpg
8	Natalia	Petrova	natalia.petrova@example.com	+79217890123	Host	Creative professional who enjoys art, music, an...	user8_profile.bmp
9	Omar	Syed	omar.syed@example.com	+61412345678	Guest	Fishing enthusiast who loves spending time on t...	user9_profile.jpg
10	Sophie	Dupont	sophie.dupont@example.com	+33678901234	Traveler	Yoga instructor who enjoys spending time in nat...	user10_profile.png
11	Emily	Chen	emily.chen@example.com	NULL	Admin	Minimalist who appreciates simple living and exp...	user11_profile.jpg
12	William	Müller	william.muller@example.com	+491573890456	Host	Big family who loves traveling together and mak...	user12_profile.bmp
13	Marie	Kim	marie.kim@example.com	+821012345678	Host (C...	Bookworm who loves getting lost in a good story.	user13_profile.jpg
14	Miguel	Sanchez	miguel.sanchez@example.com	NULL	Guest	Winemaker who is passionate about creating de...	user14_profile.png
15	Anna	Schmidt	anna.schmidt@example.com	+491736258901	Traveler	Ski instructor who loves teaching others the joy...	user15_profile.jpg
16	Ibrahim	Mohamed	REDACTED_EMAIL@example....	NULL	Guest	Musician who loves playing music and sharing it ...	user16_profile.bmp
17	Sarah	Lee	sarah.lee@example.com	+1876543210	Host	History buff who loves exploring historical sites ...	user17_profile.jpg
18	Pierre	Martin	pierre.martin@example.com	+33765432109	Traveler	Entrepreneur who is always looking for new opp...	user18_profile.png
19	Elena	Volkova	elena.volkova@example.com	+79154321087	Guest	Environmentalist who is passionate about prote...	user19_profile.jpg
20	Antonio	Garcia	antonio.garcia@example.com	+34987654321	Host	Gamer who loves spending time playing video g...	user20_profile.bmp

User: Test Case: Grouping users by role and displaying counts.

This SQL query retrieves the first names of users who are active hosts on a platform like Airbnb. Let's break down the components:

SELECT Clause:

•SELECT USER.First_name: This specifies that you only want to retrieve the First_name column from the USER table.

FROM Clause:

•FROM user: This clause defines the starting table, which is the user table in this case.

JOIN Clause:

•INNER JOIN host ON user.id=host.user_id: This clause performs an INNER JOIN between the user and host tables. An INNER JOIN ensures that only rows where there's a match in both tables (based on the specified condition) are included in the result set.

- •ON user.id=host.user_id: This is the join condition that specifies that the id column in the user table must be equal to the user_id column in the host table for a row to be included. This essentially links users with their corresponding host entries (assuming user_id is a foreign key referencing the user table in the host table).

WHERE Clause:

•WHERE listing_status = 'active': This clause filters the results to include only users who have an active listing status in the host table (assuming there's a listing_status column in the host table). This ensures you're only selecting users who are currently hosting properties.

```
SELECT
USER.First_name
from
user inner join host
on
user.id=host.user_id
WHERE
listing_status = 'active' ;
```

First_name
William
Maria
Alex
David
Omar
Emily
Marie
Anna
Sarah
Elena

Host

- Represents hosts with the detailed information This table store and manage essential host information

```sql
CREATE TABLE airbnb.Host (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    property_id INT NOT NULL,
    Listing_status VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES User(id));
```

- Use INSERT INTO statements to add host data to the table

```sql
INSERT INTO host (id, user_id, property_id, listing_status) VALUES
(101, 1, 201, 'active'),
(102, 2, 202, 'inactive'),
(103, 3, 203, 'active'),
(104, 4, 204, 'inactive'),
(105, 5, 205, 'active'),
(106, 6, 206, 'inactive'),
(107, 7, 207, 'active'),
(108, 8, 208, 'inactive'),
(109, 9, 209, 'active'),
(110, 10, 210, 'inactive'),
(111, 11, 211, 'active'),
(112, 12, 212, 'inactive'),
(113, 13, 213, 'active'),
(114, 14, 214, 'inactive'),
(115, 15, 215, 'active'),
(116, 16, 216, 'inactive'),
(117, 17, 217, 'active'),
(118, 18, 218, 'inactive'),
(119, 19, 219, 'active'),
(120, 20, 220, 'inactive');
```

id	user_id	property_id	Listing_status
101	1	201	active
102	2	202	inactive
103	3	203	active
104	4	204	inactive
105	5	205	active
106	6	206	inactive
107	7	207	active
108	8	208	inactive
109	9	209	active
110	10	210	inactive
111	11	211	active
112	12	212	inactive
113	13	213	active
114	14	214	inactive
115	15	215	active
116	16	216	inactive
117	17	217	active
118	18	218	inactive
119	19	219	active
120	20	220	inactive

Host: Test Case:This query retrieves and combines data from three tables in your Airbnb database: User (U), Host (H), and Property (P). It aims to find properties with active listings and display the corresponding host's first name and the property's address.

- **SELECT Clause:**
- U.first_name: Selects the first_name column from the User table, aliased as U for readability.
- H.listing_status: Selects the listing_status column from the Host table, aliased as H.
- P.address: Selects the address column from the Property table, aliased as P.
- **FROM Clause:**
- FROM airbnb.user U: Specifies the starting table as User from the airbnb schema (database), aliased as U.
- **JOIN Clauses:**
- **First INNER JOIN:**
  - INNER JOIN airbnb.host H ON U.id=H.user_id: This joins the User table (aliased as U) with the Host table (aliased as H) based on the condition U.id = H.user_id.
  - This ensures that only users who also have a corresponding entry in the Host table are included. In essence, it links users with their host information.
- **Second INNER JOIN:**
  - INNER JOIN airbnb.Property P on H.id=P.host_id: This joins the result of the first join (including User and Host tables) with the Property table (aliased as P) based on the condition H.id = P.host_id.
  - This connects hosts (from the first join) with their corresponding properties based on the host_id (assuming it's a foreign key referencing the Host table in the Property table).

```
SELECT
U.first_name, H.listing_status, P.address
FROM
airbnb.user U inner join airbnb.host H ON
U.id=H.user_idinner join airbnb.Property P on
H.id=P.host_id;
```

first_name	listing_status	address
William	active	123 Ocean View Ave, Miami, FL
Jane	inactive	456 Main St, New York, NY
Maria	active	789 Pine Ridge Rd, Aspen, CO
Li	inactive	1011 Freedom St, Boston, MA
Alex	active	1213 Evergreen Dr, Vail, CO
Aisha	inactive	1415 Country Lane, Napa, CA
David	active	1617 Palm Tree Way, Maui, HI
Natalia	inactive	1819 Industrial Ave, Chicago, IL
Omar	active	2021 Lakeside Dr, Lake Tahoe, NV
Sophie	inactive	2223 Cactus Way, Palm Springs, CA
Emily	active	2425 Elm St, Portland, OR
William	inactive	2627 Seashell Dr, Outer Banks, NC
Marie	active	2829 Winding Path, Asheville, NC
Miguel	inactive	3031 Grapevine Ln, Sonoma, CA
Anna	active	3233 Spruce Peak Rd, Breckenridg...
Ibrahim	inactive	3435 cobblestone St, New Orleans,...
Sarah	active	3637 Canal St, Amsterdam, Netherl...
Pierre	inactive	3839 Aspen Ridge Dr, Jackson Hole...
Elena	active	4041 Rainforest Way, Costa Rica
Antonio	inactive	4243 Castaway Cay, Bahamas

Guest

```sql
CREATE TABLE Guest (
    id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT NOT NULL,
    FOREIGN KEY (User_id) REFERENCES User(id)
);
```

```sql
INSERT INTO Guest (id, User_id)
VALUES
(301, 1),
(302, 2),
(303, 3),
(304, 4),
(305, 5),
(306,6),
(307, 7),
(308, 8),
(309, 9),
(310, 10),
(311, 11),
(312, 12),
(313, 13),
(314, 14),
(315, 15),
(316, 16),
(317, 17),
(318, 18),
(319, 19),
(320, 20);
```

id	User_id
301	1
302	2
303	3
304	4
305	5
306	6
307	7
308	8
309	9
310	10
311	11
312	12
313	13
314	14
315	15
316	16
317	17
318	18
319	19
320	20

CREATE TABLE Guest:
This statement creates a table named Guest in your database. The table definition specifies the following columns:
•id: This is an integer (INT) column that will be the primary key (PRIMARY KEY) for the table. It also has the AUTO_INCREMENT attribute, which means it will automatically generate a unique integer value for each new guest record inserted into the table.
•User_id: This is another integer (INT) column that stores a reference to a user in the User table. It's declared NOT NULL, meaning it cannot be empty for any guest record.
•FOREIGN KEY (User_id) REFERENCES User(id): This statement defines a foreign key constraint. It ensures that the values in the User_id column of the Guest table must also exist as valid IDs in the id column of the User table. This creates a relationship between the two tables, ensuring data integrity.

INSERT INTO Guest (id, User_id):
This statement inserts multiple rows of data into the Guest table. It specifies the following values for each row:
•id: This column is auto-incremented, so you don't need to specify values here. The database will assign unique IDs automatically.
•User_id: This column stores the ID of the corresponding user in the User table. The provided values (1 through 20) suggest there are 20 users in the User table referenced by these guests.
In essence, these statements create a Guest table to store guest information and link it to the User table using a foreign key relationship.

Guest: Test: Case This query retrieves information about guest bookings in your Airbnb database by joining three tables: Guest (G), Booking (B), and Property (P). Here's a breakdown of what each part does:

```
SELECT G.id, P.name, B.status FROM airbnb.guest Ginner
join airbnb.booking B ON G.id=B.guest_idinner join
airbnb.Property P on B.property_id=P.id;
```

1. SELECT Clause:
•G.id: Selects the id column from the Guest table, aliased as G for clarity. This represents the unique identifier for each guest.
•P.name: Selects the name column from the Property table, aliased as P. This represents the name of the property that was booked.
•B.status: Selects the status column from the Booking table, aliased as B. This represents the current status of the booking (e.g., confirmed, pending, New, Partially Paid).

2. FROM Clause:
•FROM airbnb.guest G: Specifies the starting table as Guest from the airbnb schema (database), aliased as G. This is where the query starts looking for guest information.

3. JOIN Clauses:
•**First INNER JOIN:**
  •INNER JOIN airbnb.booking B ON G.id=B.guest_id: This joins the Guest table (aliased as G) with the Booking table (aliased as B) based on the condition G.id = B.guest_id. This ensures that only guest entries with corresponding booking information in the Booking table are included. In simpler terms, it links guests with their bookings.
•**Second INNER JOIN:**
  •INNER JOIN airbnb.Property P on B.property_id=P.id: This further joins the result of the first join (including Guest and Booking tables) with the Property table (aliased as P) based on the condition B.property_id = P.id. This connects bookings (from the first join) with their corresponding properties based on the property_id (assuming it's a foreign key referencing the Property table in the Booking table). This essentially links bookings to the properties that were booked.

id	name	status
301	Cozy Beachfront Cottage	Confirmed
302	Modern City Apartment	Completed
303	Secluded Mountain Cabin	Pending
304	Historic Townhouse	Confirmed
305	Luxurious Ski Chalet	New
306	Charming Farmhouse	Confirmed
307	Tropical Island Bungalow	Partially Paid
308	Designer Loft Apartment	Completed
309	Rustic Lakeside Cabin	New
310	Desert Oasis Retreat	Confirmed
311	Cozy Tiny House	Completed
312	Family-Friendly Beach H...	Confirmed
313	Modern Treehouse Geta...	New
314	Private Vineyard Cottage	Pending
315	Ski-in/Ski-out Condo	Completed
316	Historic City Center Apar...	Partially Paid
317	Canal-side Townhouse	Confirmed
318	Luxury Mountain Lodge	New
319	Jungle Treehouse Adven...	Confirmed
320	Private Island Escape	Completed

Property: This code creates a table named Property to store information about properties on your Airbnb platform.

```sql
CREATE TABLE Property (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Host_id INT NOT NULL,
    Name VARCHAR(255) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Amenities TEXT,
    FOREIGN KEY (Host_id) REFERENCES Host(id)
);
```

- **INSERT Statements:** Populate the table with sample property data (20 entries).

```sql
INSERT INTO Property (id, Host_id, Name, Address, Amenities) VALUES
(201, 101, 'Cozy Beachfront Cottage', '123 Ocean View Ave, Miami, FL', 'Beachfront, Wi-Fi, Parking'),
(202, 102, 'Modern City Apartment', '456 Main St, New York, NY', 'City Views, Gym, Rooftop Terrace'),
(203, 103, 'Secluded Mountain Cabin', '789 Pine Ridge Rd, Aspen, CO', 'Hot Tub, Fireplace, Hiking Trails'),
(204, 104, 'Historic Townhouse', '1011 Freedom St, Boston, MA', 'Walkable Location, Patio, Pet-Friendly'),
(205, 105, 'Luxurious Ski Chalet', '1213 Evergreen Dr, Vail, CO', 'Ski-in/Ski-out, Sauna, Steam Room'),
(206, 106, 'Charming Farmhouse', '1415 Country Lane, Napa, CA', 'Vineyard Views, Pool, BBQ Area'),
(207, 107, 'Tropical Island Bungalow', '1617 Palm Tree Way, Maui, HI', 'Oceanfront, Private Beach, Hammock'),
(208, 108, 'Designer Loft Apartment', '1819 Industrial Ave, Chicago, IL', 'Exposed Brick, Balcony, Concierge'),
(209, 109, 'Rustic Lakeside Cabin', '2021 Lakeside Dr, Lake Tahoe, NV', 'Lakefront, Fishing, Kayaking'),
(210, 110, 'Desert Oasis Retreat', '2223 Cactus Way, Palm Springs, CA', 'Pool, Hot Tub, Mountain Views'),
(211, 111, 'Cozy Tiny House', '2425 Elm St, Portland, OR', 'Minimalist Design, Eco-Friendly, Close to Downtown'),
(212, 112, 'Family-Friendly Beach House', '2627 Seashell Dr, Outer Banks, NC', 'Beach Access, Game Room, Spacious Kitchen'),
(213, 113, 'Modern Treehouse Getaway', '2829 Winding Path, Asheville, NC', 'Unique Experience, Forest Views, Tranquil Setting'),
(214, 114, 'Private Vineyard Cottage', '3031 Grapevine Ln, Sonoma, CA', 'Vineyard Tour Included, Wine Tasting, Patio'),
(215, 115, 'Ski-in/Ski-out Condo', '3233 Spruce Peak Rd, Breckenridge, CO', 'Slopeside Location, Ski Locker, Fireplace'),
(216, 116, 'Historic City Center Apartment', '3435 cobblestone St, New Orleans, LA', 'French Quarter Balcony, Walking Distance to Attractions'),
(217, 117, 'Canal-side Townhouse', '3637 Canal St, Amsterdam, Netherlands', 'Canal Views, Boat Tours Nearby, Central Location'),
(218, 118, 'Luxury Mountain Lodge', '3839 Aspen Ridge Dr, Jackson Hole, WY', 'Spa Services, Gourmet Dining, Ski Valet'),
(219, 119, 'Jungle Treehouse Adventure', '4041 Rainforest Way, Costa Rica', 'Immerse Yourself in Nature, Wildlife Viewing, Sustainability Focused'),
(220, 120, 'Private Island Escape', '4243 Castaway Cay, Bahamas', 'Secluded Paradise, Beachfront Relaxation, All-Inclusive Package');
```

- **Columns:**
- id (INT): Unique identifier for each property (auto-increments).
- Host_id (INT): ID of the host who owns the property (references the Host table using a foreign key).
- Name (VARCHAR(255)): Name of the property.
- Address (VARCHAR(255)): Address of the property.
- Amenities (TEXT): Description of the property's amenities (stored as text).
- **Foreign Key:** Ensures Host_id values exist in the Host table (maintains data integrity).

- Roles (e.g., ‘Administrator’, ‘Guest’, ‘Host’) and their descriptions can be inserted into table using INSERT INTO statements.

id	Host_id	Name	Address	Amenities
201	101	Cozy Beachfront C…	123 Ocean View Ave…	Beachfront, Wi-Fi, Par…
202	102	Modern City Apart…	456 Main St, New Yo…	City Views, Gym, Roof…
203	103	Secluded Mountain…	789 Pine Ridge Rd, …	Hot Tub, Fireplace, Hi…
204	104	Historic Townhouse	1011 Freedom St, B…	Walkable Location, Pa…
205	105	Luxurious Ski Chalet	1213 Evergreen Dr, …	Ski-in/Ski-out, Sauna, …
206	106	Charming Farmhouse	1415 Country Lane, …	Vineyard Views, Pool, …
207	107	Tropical Island Bun…	1617 Palm Tree Way…	Oceanfront, Private B…
208	108	Designer Loft Apar…	1819 Industrial Ave,…	Exposed Brick, Balcon…
209	109	Rustic Lakeside Ca…	2021 Lakeside Dr, L…	Lakefront, Fishing, Ka…
210	110	Desert Oasis Retreat	2223 Cactus Way, P…	Pool, Hot Tub, Mounta…
211	111	Cozy Tiny House	2425 Elm St, Portlan…	Minimalist Design, Eco-…
212	112	Family-Friendly Be…	2627 Seashell Dr, O…	Beach Access, Game …
213	113	Modern Treehouse…	2829 Winding Path, …	Unique Experience, Fo…
214	114	Private Vineyard C…	3031 Grapevine Ln, …	Vineyard Tour Include…
215	115	Ski-in/Ski-out Condo	3233 Spruce Peak R…	Slopeside Location, Ski…
216	116	Historic City Cente…	3435 cobblestone St…	French Quarter Balcon…
217	117	Canal-side Townh…	3637 Canal St, Amst…	Canal Views, Boat Tou…
218	118	Luxury Mountain L…	3839 Aspen Ridge D…	Spa Services, Gourme…
219	119	Jungle Treehouse …	4041 Rainforest Wa…	Immerse Yourself in N…
220	120	Private Island Esc…	4243 Castaway Cay…	Secluded Paradise, Be…

Property: Test: Case This query aims to find properties and their current availability status.

```
SELECT  H.id, P.name, C.availability FROM
airbnb.host H inner join
airbnb.Property P on H.id=P.host_id inner
join airbnb.calendar C ON C.property_id=P.id;
```

- **SELECT Clause:**
- H.id: Selects the id column from the Host table, aliased as H for clarity (represents the host's ID).
- P.name: Selects the name column from the Property table, aliased as P (represents the property's name).
- C.availability: Selects the availability column from the Calendar table, aliased as C (represents the property's availability status).
- **FROM Clause:**
- FROM airbnb.host H: Specifies the starting table as Host from the airbnb schema (database), aliased as H.
- **JOIN Clauses:**
- **First INNER JOIN:**
    - INNER JOIN airbnb.Property P on H.id=P.host_id: This joins the Host table (aliased as H) with the Property table (aliased as P) based on the condition H.id = P.host_id. This ensures that only hosts with corresponding properties in the Property table are included. It links hosts with their properties.
- **Second INNER JOIN:**
    - INNER JOIN airbnb.calendar C ON C.property_id=P.id: This further joins the result of the first join (including Host and Property tables) with the Calendar table (aliased as C) based on the condition C.property_id = P.id. This connects properties (from the first join) with their availability information in the Calendar table (assuming property_id is a foreign key referencing the Property table).

id	name	availability
101	Cozy Beachfront Cottage	1
102	Modern City Apartment	0
103	Secluded Mountain Cabin	1
104	Historic Townhouse	0
105	Luxurious Ski Chalet	1
106	Charming Farmhouse	1
107	Tropical Island Bungalow	0
108	Designer Loft Apartment	1
109	Rustic Lakeside Cabin	0
110	Desert Oasis Retreat	1
111	Cozy Tiny House	1
112	Family-Friendly Beach H...	0
113	Modern Treehouse Geta...	1
114	Private Vineyard Cottage	0
115	Ski-in/Ski-out Condo	1
116	Historic City Center Apar...	1
117	Canal-side Townhouse	0
118	Luxury Mountain Lodge	1
119	Jungle Treehouse Adven...	0
120	Private Island Escape	1

Booking

- Store and manage essential booking information

```sql
CREATE TABLE Booking (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Guest_id INT NOT NULL,
    Property_id INT NOT NULL,
    Dates VARCHAR(255) NOT NULL,
    Status VARCHAR(255) NOT NULL,
    FOREIGN KEY (Guest_id) REFERENCES Guest(id),
    FOREIGN KEY (Property_id) REFERENCES Property(id)
);
```

Table: Booking

- **Columns:**
  - id (INT): Unique identifier for each booking (auto-increments).
  - Guest_id (INT): ID of the guest who made the booking (references Guest table using a foreign key).
  - Property_id (INT): ID of the property that was booked (references Property table using a foreign key).
  - Dates (VARCHAR(255)): Reservation date range in a specific format (e.g., '2024-07-15 TO 2024-07-22').
  - Status (VARCHAR(255)): Current status of the booking (e.g., 'Confirmed', 'Completed', 'Cancelled').
- **Foreign Keys:** Ensure Guest_id and Property_id values exist in their respective tables (maintains data integrity)..

- Use INSERT INTO statements to add booking data to the table

```sql
INSERT INTO Booking (id, Guest_id, Property_id, Dates, Status) VALUES
(401, 301, 201, '2024-07-15 TO 2024-07-22', 'Confirmed'),
(402, 302, 202, '2024-06-10 TO 2024-06-17', 'Completed'),
(403, 303, 203, '2024-12-25 TO 2025-01-02', 'Pending'),
(404, 304, 204, '2024-09-01 TO 2024-09-07', 'Confirmed'),
(405, 305, 205, '2025-02-14 TO 2025-02-20', 'New'),
(406, 306, 206, '2024-08-12 TO 2024-08-19', 'Confirmed'),
(407, 307, 207, '2024-10-26 TO 2024-11-02', 'Partially Paid'),
(408, 308, 208, '2024-07-04 TO 2024-07-10', 'Completed'),
(409, 309, 209, '2025-03-07 TO 2025-03-14', 'New'),
(410, 310, 210, '2024-11-21 TO 2024-11-28', 'Confirmed'),
(411, 311, 211, '2024-05-25 TO 2024-06-01', 'Completed'),
(412, 312, 212, '2024-08-05 TO 2024-08-10', 'Confirmed'),
(413, 313, 213, '2024-12-18 TO 2024-12-24', 'New'),
(414, 314, 214, '2025-01-20 TO 2025-01-27', 'Pending'),
(415, 315, 215, '2024-06-23 TO 2024-07-01', 'Completed'),
(416, 316, 216, '2024-09-15 TO 2024-09-22', 'Partially Paid'),
(417, 317, 217, '2024-10-05 TO 2024-10-12', 'Confirmed'),
(418, 318, 218, '2025-04-11 TO 2025-04-18', 'New'),
(419, 319, 219, '2024-07-28 TO 2024-08-04', 'Confirmed'),
(420, 320, 220, '2024-06-07 TO 2024-06-14', 'Completed');
```

id	Guest_id	Property_id	Dates	Status
401	301	201	2024-07-15 TO 2024-07-22	Confirmed
402	302	202	2024-06-10 TO 2024-06-17	Completed
403	303	203	2024-12-25 TO 2025-01-02	Pending
404	304	204	2024-09-01 TO 2024-09-07	Confirmed
405	305	205	2025-02-14 TO 2025-02-20	New
406	306	206	2024-08-12 TO 2024-08-19	Confirmed
407	307	207	2024-10-26 TO 2024-11-02	Partially Paid
408	308	208	2024-07-04 TO 2024-07-10	Completed
409	309	209	2025-03-07 TO 2025-03-14	New
410	310	210	2024-11-21 TO 2024-11-28	Confirmed
411	311	211	2024-05-25 TO 2024-06-01	Completed
412	312	212	2024-08-05 TO 2024-08-10	Confirmed
413	313	213	2024-12-18 TO 2024-12-24	New
414	314	214	2025-01-20 TO 2025-01-27	Pending
415	315	215	2024-06-23 TO 2024-07-01	Completed
416	316	216	2024-09-15 TO 2024-09-22	Partially Paid
417	317	217	2024-10-05 TO 2024-10-12	Confirmed
418	318	218	2025-04-11 TO 2025-04-18	New
419	319	219	2024-07-28 TO 2024-08-04	Confirmed
420	320	220	2024-06-07 TO 2024-06-14	Completed

Booking: Test Case: This query aims to find bookings and their corresponding payment details, including the amount paid.

- **SELECT Clause:**
- A.id: Selects the id column from the BookingPayment table (aliased as A), representing the booking payment ID.
- B.dates: Selects the dates column from the Booking table (aliased as B), representing the reservation date range for the booking.
- P.amount: Selects the amount column from the Payment table (aliased as P), representing the payment amount.
- **FROM Clause:**
- FROM airbnb.booking B: Specifies the starting table as Booking from the airbnb schema (database), aliased as B.
- **JOIN Clauses:**
- **First INNER JOIN:**
  - INNER JOIN airbnb.bookingpayment A ON B.id=A.booking_id: This joins the Booking table (aliased as B) with the BookingPayment table (aliased as A) based on the condition B.id = A.booking_id. This ensures that only bookings with corresponding payment information in the BookingPayment table are included. It links bookings with their payment records.
- **Second INNER JOIN:**
  - INNER JOIN airbnb.payment P on A.payment_id=P.id: This further joins the result of the first join (including Booking and BookingPayment tables) with the Payment table (aliased as P) based on the condition A.payment_id = P.id. This connects booking payments (from the first join) with their corresponding payment details in the Payment table (assuming payment_id is a foreign key referencing the Payment table).

```
SELECT A.id, B.dates, P.amount FROM airbnb.booking B
inner join airbnb.bookingpayment A ON
B.id=A.booking_id inner join
airbnb.payment P on A.payment_id=P.id;
```

id	dates	amount
71	2024-07-15 TO 2024-07-22	150.00
72	2024-06-10 TO 2024-06-17	325.75
73	2024-12-25 TO 2025-01-02	87.99
74	2024-09-01 TO 2024-09-07	129.50
75	2025-02-14 TO 2025-02-20	499.99
76	2024-08-12 TO 2024-08-19	210.25
77	2024-10-26 TO 2024-11-02	784.00
78	2024-07-04 TO 2024-07-10	189.00
79	2025-03-07 TO 2025-03-14	256.40
80	2024-11-21 TO 2024-11-28	100.00
81	2024-05-25 TO 2024-06-01	67.88
82	2024-08-05 TO 2024-08-10	985.32
83	2024-12-18 TO 2024-12-24	142.11
84	2025-01-20 TO 2025-01-27	379.00
85	2024-06-23 TO 2024-07-01	52.99
86	2024-09-15 TO 2024-09-22	198.70
87	2024-10-05 TO 2024-10-12	412.65
88	2025-04-11 TO 2025-04-18	2000.00
89	2024-07-28 TO 2024-08-04	89.50
90	2024-06-07 TO 2024-06-14	124.95

Review

This code creates a table named Review to store guest reviews for properties on your Airbnb platform. Here's a quick summary:

```sql
CREATE TABLE Review (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Guest_id INT NOT NULL,
    Property_id INT NOT NULL,
    Rating INT,
    Comment TEXT,
    timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (Guest_id) REFERENCES Guest(id),
    FOREIGN KEY (Property_id) REFERENCES Property(id)
);
```

- Use INSERT INTO statements to add review data to the table

```sql
INSERT INTO Review (id, Guest_id, Property_id, Rating, Comment, timestamp) VALUES
(501, 301, 201, 5, 'Beautiful beachfront location, exactly as described!', CURRENT_TIMESTAMP),
(502, 302, 202, 4, 'Great city apartment, convenient location. A bit noisy at night.', CURRENT_TIMESTAMP),
(503, 303, 203, 5, 'Cozy cabin, perfect for a relaxing getaway. Would definitely recommend!', CURRENT_TIMESTAMP),
(504, 304, 204, 4, 'Unique townhouse, great for exploring the city. Needs some minor maintenance updates.', CURRENT_TIMESTAMP),
(505, 305, 205, 5, 'Amazing ski chalet! Perfect for a winter vacation. Host was very responsive.', CURRENT_TIMESTAMP),
(506, 306, 206, 5, 'Charming farmhouse with stunning vineyard views. Delicious breakfast included!', CURRENT_TIMESTAMP),
(507, 307, 207, 5, 'Dreamy island bungalow right on the beach. Perfect for a tropical paradise escape!', CURRENT_TIMESTAMP),
(508, 308, 208, 4.5, 'Stylish loft apartment, great for a weekend stay. Could use better amenities.', CURRENT_TIMESTAMP),
(509, 309, 209, 5, 'Beautiful lakefront cabin. Perfect for enjoying nature and outdoor activities.', CURRENT_TIMESTAMP),
(510, 310, 210, 4, 'Relaxing desert retreat. Pool was a bit small for the number of guests.', CURRENT_TIMESTAMP),
(511, 311, 211, 5, 'Great tiny house experience! Perfect for a minimalist stay close to downtown.', CURRENT_TIMESTAMP),
(512, 312, 212, 5, 'Spacious beach house, great for families. Well-equipped kitchen.', CURRENT_TIMESTAMP),
(513, 313, 213, 5, 'Unique treehouse stay! Perfect for an adventurous getaway. Exactly as pictured.', CURRENT_TIMESTAMP),
(514, 314, 214, 4.5, 'Lovely vineyard cottage. Would have liked more information about the included wine tour.', CURRENT_TIMESTAMP),
(515, 315, 215, 5, 'Ski-in/ski-out convenience was amazing! Perfect for a ski vacation.', CURRENT_TIMESTAMP),
(516, 316, 216, 4, 'Great location in the French Quarter. Apartment could use some modernization.', CURRENT_TIMESTAMP),
(517, 317, 217, 3,  'Beautiful canal-side townhouse. Great for exploring Amsterdam by boat!', '2024-05-21 00:56:00'),
(518, 318, 218, 4, 'Disappointed with the service. Property was not well-maintained.', CURRENT_TIMESTAMP),
(519, 319, 219, 3, 'Great location, but the noise from the construction next door was unbearable.', CURRENT_TIMESTAMP),
(520, 320, 220, 5, 'Perfect stay! Would definitely recommend this property and host.', CURRENT_TIMESTAMP);
```

- **Columns:**
- id (INT): Unique identifier for each review (auto-increments).
- Guest_id (INT): ID of the guest who wrote the review (references Guest table using a foreign key).
- Property_id (INT): ID of the property that was reviewed (references Property table using a foreign key).

- Rating (INT): Guest's rating for the property (presumably on a scale of 1-5).
- Comment (TEXT): Guest's written review about the property.
- timestamp (DATETIME): Date and time the review was submitted (automatically set to current time).
- **Foreign Keys:** Ensure Guest_id and Property_id values exist in their respective tables (maintains data integrity).

id	Guest_id	Property_id	Rating	Comment	timestamp
501	301	201	5	Beautiful beachfront location, exactly as descri...	2024-05-21 15:34:24
502	302	202	4	Great city apartment, convenient location. A bit...	2024-05-21 15:34:24
503	303	203	5	Cozy cabin, perfect for a relaxing getaway. Wo...	2024-05-21 15:34:24
504	304	204	4	Unique townhouse, great for exploring the city....	2024-05-21 15:34:24
505	305	205	5	Amazing ski chalet! Perfect for a winter vacatio...	2024-05-21 15:34:24
506	306	206	5	Charming farmhouse with stunning vineyard vie...	2024-05-21 15:34:24
507	307	207	5	Dreamy island bungalow right on the beach. Per...	2024-05-21 15:34:24
508	308	208	5	Stylish loft apartment, great for a weekend sta...	2024-05-21 15:34:24
509	309	209	5	Beautiful lakefront cabin. Perfect for enjoying n...	2024-05-21 15:34:24
510	310	210	4	Relaxing desert retreat. Pool was a bit small for...	2024-05-21 15:34:24
511	311	211	5	Great tiny house experience! Perfect for a mini...	2024-05-21 15:34:24
512	312	212	5	Spacious beach house, great for families. Well-...	2024-05-21 15:34:24
513	313	213	5	Unique treehouse stay! Perfect for an adventur...	2024-05-21 15:34:24
514	314	214	5	Lovely vineyard cottage. Would have liked mor...	2024-05-21 15:34:24
515	315	215	5	Ski-in/ski-out convenience was amazing! Perfect...	2024-05-21 15:34:24
516	316	216	4	Great location in the French Quarter. Apartmen...	2024-05-21 15:34:24
517	317	217	3	Beautiful canal-side townhouse. Great for explo...	2024-05-21 00:56:00
518	318	218	4	Disappointed with the service. Property was no...	2024-05-21 15:34:24
519	319	219	3	Great location, but the noise from the construct...	2024-05-21 15:34:24
520	320	220	5	Perfect stay! Would definitely recommend this p...	2024-05-21 15:34:24

Review: Test: Case: This query aims to retrieve guest IDs, booking IDs, and review ratings from your Airbnb database. Here's a breakdown:

```
FROM airbnb.review R
inner join airbnb.guest G
ON R.guest_id=G.id

inner join airbnb.bookingreview Z
on G.id=Z.Guest_id;
```

id	booking_id	rating
301	401	5
302	402	4
303	403	5
304	404	4
305	405	5
306	406	5
307	407	5
308	408	5
309	409	5
310	410	4
311	411	5
312	412	5
313	413	5
314	414	5
315	415	5
316	416	4
317	417	3
318	418	4
319	419	3
320	420	5

- **SELECT Clause:**
- G.id: Selects the id column from the Guest table (aliased as G), representing the guest's ID.
- Z.booking_id: Selects the booking_id column from the BookingReview table (aliased as Z), representing the ID of the booking associated with the review.
- R.rating: Selects the rating column from the Review table (aliased as R), representing the guest's rating for the property.
- **FROM Clause:**
- FROM airbnb.review R: Specifies the starting table as Review from the airbnb schema (database), aliased as R.
- **JOIN Clauses:**
- **First INNER JOIN:**
    - INNER JOIN airbnb.guest G ON R.guest_id=G.id: This joins the Review table (aliased as R) with the Guest table (aliased as G) based on the condition R.guest_id = G.id. This ensures that only reviews with corresponding guest information are included. It links reviews to the guests who wrote them.
- **Second INNER JOIN (Assuming Z.guest_id exists):**
    - INNER JOIN airbnb.bookingreview Z on G.id=Z.Guest_id: This join might have an error. The table being joined (BookingReview) seems to be referencing Guest.id again. It would be more logical to join with Booking using Z.booking_id to link reviews with their corresponding bookings. However, if Z actually has a guest_id column and this is the intended join, it would create a redundant link between Guest and Review (since you already have the guest_id from the first join).

Payment

Store and manage essential payment information

```sql
CREATE TABLE Payment (
    id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT NOT NULL,
    Order_id INT,
    Amount DECIMAL(10,2) NOT NULL,
    Timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (User_id) REFERENCES User(id)
);
```

Table: Payment

- **Columns:**
  - id (INT): Unique identifier for each payment (auto-increments).
  - User_id (INT): ID of the user who made the payment (references User table using a foreign key).
  - Order_id (INT): ID of the order associated with the payment (optional).
  - Amount (DECIMAL): Transaction amount (supports up to 2 decimal places).
  - Timestamp (DATETIME): Date and time of the payment (automatically set to current time).
- **Foreign Key:** Ensures User_id values exist in the User table (maintains data integrity).

- Use INSERT INTO statements to add payment data to the table

```sql
INSERT INTO Payment (id, User_id, Order_id, Amount, Timestamp) VALUES
    (601, 1, 202405001, 150.00, CURRENT_TIMESTAMP),
    (602, 2, 202405002, 325.75, CURRENT_TIMESTAMP),
    (603, 3, 202405003, 87.99, CURRENT_TIMESTAMP),
    (604, 4, 202405004, 129.50, CURRENT_TIMESTAMP),
    (605, 5, 202405005, 499.99, CURRENT_TIMESTAMP),
    (606, 6, 202405006, 210.25, CURRENT_TIMESTAMP),
    (607, 7, 202405007, 784.00, CURRENT_TIMESTAMP),
    (608, 8, 202405008, 189.00, CURRENT_TIMESTAMP),
    (609, 9, 202405009, 256.40, CURRENT_TIMESTAMP),
    (610, 10, 202405010, 100.00, CURRENT_TIMESTAMP),
    (611, 11, 202405011, 67.88, CURRENT_TIMESTAMP),
    (612, 12, 202405012, 985.32, CURRENT_TIMESTAMP),
    (613, 13, 202405013, 142.11, CURRENT_TIMESTAMP),
    (614, 14, 202405014, 379.00, CURRENT_TIMESTAMP),
    (615, 15, 202405015, 52.99, CURRENT_TIMESTAMP),
    (616, 16, 202405016, 198.70, CURRENT_TIMESTAMP),
    (617, 17, 202405017, 412.65, CURRENT_TIMESTAMP),
    (618, 18, 202405018, 2000.00, CURRENT_TIMESTAMP),
    (619, 19, 202405019, 89.50, CURRENT_TIMESTAMP),
    (620, 20, 202405020, 124.95, CURRENT_TIMESTAMP);
```

id	User_id	Order_id	Amount	Timestamp
601	1	202405001	150.00	2024-05-21 14:56:40
602	2	202405002	325.75	2024-05-21 14:56:40
603	3	202405003	87.99	2024-05-21 14:56:40
604	4	202405004	129.50	2024-05-21 14:56:40
605	5	202405005	499.99	2024-05-21 14:56:40
606	6	202405006	210.25	2024-05-21 14:56:40
607	7	202405007	784.00	2024-05-21 14:56:40
608	8	202405008	189.00	2024-05-21 14:56:40
609	9	202405009	256.40	2024-05-21 14:56:40
610	10	202405010	100.00	2024-05-21 14:56:40
611	11	202405011	67.88	2024-05-21 14:56:40
612	12	202405012	985.32	2024-05-21 14:56:40
613	13	202405013	142.11	2024-05-21 14:56:40
614	14	202405014	379.00	2024-05-21 14:56:40
615	15	202405015	52.99	2024-05-21 14:56:40
616	16	202405016	198.70	2024-05-21 14:56:40
617	17	202405017	412.65	2024-05-21 14:56:40
618	18	202405018	2000.00	2024-05-21 14:56:40
619	19	202405019	89.50	2024-05-21 14:56:40
620	20	202405020	124.95	2024-05-21 14:56:40

Payment: Test: Case:This query aims to find details about payments made by users on Airbnb.

```
SELECT U.first_name, P.amount, Y.methodtype
FROM airbnb.Payment p
inner join airbnb.user U ON P.user_id=U.id
inner join airbnb.paymentmethod Y on U.id=Y.user_id;
```

- **SELECT Clause:**
- U.first_name: Selects the first_name column from the User table (aliased as U), representing the user's first name.
- P.amount: Selects the amount column from the Payment table (aliased as P), representing the payment amount.
- Y.methodtype: Selects the methodtype column from the PaymentMethod table (aliased as Y), representing the payment method used (e.g., 'Credit Card', 'Debit Card', etc.).
- **FROM Clause:**
- FROM airbnb.Payment p: Specifies the starting table as Payment from the airbnb schema (database), aliased as p.
- **JOIN Clauses:**
- **First INNER JOIN:**
  - INNER JOIN airbnb.user U ON P.user_id=U.id: This joins the Payment table (aliased as p) with the User table (aliased as U) based on the condition P.user_id = U.id. This connects payments to the users who made them.
- **Second INNER JOIN:**
  - INNER JOIN airbnb.paymentmethod Y on U.id=Y.user_id: This further joins the result of the first join (including Payment and User tables) with the PaymentMethod table (aliased as Y) based on the condition U.id = Y.user_id. This links users to their preferred payment methods stored in the PaymentMethod table (assuming user_id is a foreign key referencing the User table).

first_name	amount	methodtype
William	150.00	Credit Card
Jane	325.75	Debit Card
Maria	87.99	E-Wallet
Li	129.50	Net Banking
Alex	499.99	Cash on Delivery
Aisha	210.25	Credit Card
David	784.00	Debit Card
Natalia	189.00	E-Wallet
Omar	256.40	Net Banking
Sophie	100.00	Credit Card
Emily	67.88	Debit Card
William	985.32	E-Wallet
Marie	142.11	Net Banking
Miguel	379.00	Credit Card
Anna	52.99	Debit Card
Ibrahim	198.70	Cash on Delivery
Sarah	412.65	UPI
Pierre	2000.00	Mobile Wallet
Elena	89.50	Other
Antonio	124.95	Prepaid Card

Location

- It offers an organized method for storing and accessing geographical information about locations within the system. This plays a vital role in connecting places with their specific coordinates, enhancing the search and organization functions within the application
- The CREATE TABLE SQL command is utilized to outline the structure of the location table by setting the data type and constraints for each column.

```sql
CREATE TABLE Location (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Country VARCHAR(255) NOT NULL,
    Region VARCHAR(255),
    City VARCHAR(255),
    Street VARCHAR(255),
    Post_code VARCHAR(255)
);
```

- The table can be populated with data using INSERT INTO statements.

```sql
INSERT INTO Location (id, Country, Region, City, Street, Post_code) VALUES
(801, 'France', 'Île-de-France', 'Paris', '10 Rue de Rivoli', '75001'),
(802, 'Italy', 'Tuscany', 'Florence', 'Piazza della Signoria, 1', '50121'),
(803, 'Spain', 'Catalonia', 'Barcelona', 'Passeig de Gràcia, 11', '08007'),
(804, 'United Kingdom', 'England', 'London', '221B Baker Street', 'NW1 6XE'),
(805, 'Japan', 'Kantō', 'Tokyo', 'Shibuya Crossing', '150-0002'),
(806, 'United States', 'California', 'Los Angeles', '1600 Vine St', '90028'),
(807, 'Canada', 'British Columbia', 'Vancouver', 'Granville Island', 'V6H 3S2'),
(808, 'Australia', 'New South Wales', 'Sydney', 'Sydney Opera House', '2000'),
(809, 'Brazil', 'Southeast', 'Rio de Janeiro', 'Copacabana Beach', '22054-001'),
(810, 'Mexico', 'Central Mexico', 'Mexico City', 'Zócalo Square', '06000'),
(811, 'India', 'Delhi', 'Delhi', 'Taj Mahal', '282001'),
(812, 'China', 'Beijing', 'Beijing', 'Great Wall of China', '100000'),
(813, 'South Africa', 'Western Cape', 'Cape Town', 'Table Mountain', '8001'),
(814, 'Argentina', 'Buenos Aires', 'Buenos Aires', 'La Boca', 'C1147AAN'),
(815, 'New Zealand', 'South Island', 'Queenstown', 'Skippers Canyon Road', '9300'),
(816, 'Peru', 'Cuzco Region', 'Machu Picchu', 'Machu Picchu Historic Sanctuary', '08000'),
(817, 'Iceland', 'Capital Region', 'Reykjavík', 'Hallgrímskirkja', '101'),
(818, 'Kenya', 'Rift Valley', 'Masai Mara National Reserve','Zingabai takli', 'N/A'),
(819, 'Greece', 'South Aegean', 'Santorini', 'Oia', '84702'),
(820, 'Turkey', 'Marmara', 'Istanbul', 'Hagia Sophia', '34400');
```

id	Country	Region	City	Street	Post_code
801	France	Île-de-France	Paris	10 Rue de Rivoli	75001
802	Italy	Tuscany	Florence	Piazza della Signoria, 1	50121
803	Spain	Catalonia	Barcelona	Passeig de Gràcia, 11	08007
804	United Kingdom	England	London	221B Baker Street	NW1 6XE
805	Japan	Kantō	Tokyo	Shibuya Crossing	150-0002
806	United States	California	Los Angeles	1600 Vine St	90028
807	Canada	British Columbia	Vancouver	Granville Island	V6H 3S2
808	Australia	New South Wales	Sydney	Sydney Opera House	2000
809	Brazil	Southeast	Rio de Janeiro	Copacabana Beach	22054-001
810	Mexico	Central Mexico	Mexico City	Zócalo Square	06000
811	India	Delhi	Delhi	Taj Mahal	282001
812	China	Beijing	Beijing	Great Wall of China	100000
813	South Africa	Western Cape	Cape Town	Table Mountain	8001
814	Argentina	Buenos Aires	Buenos Aires	La Boca	C1147AAN
815	New Zealand	South Island	Queenstown	Skippers Canyon Road	9300
816	Peru	Cuzco Region	Machu Picchu	Machu Picchu Historic ...	08000
817	Iceland	Capital Region	Reykjavík	Hallgrímskirkja	101
818	Kenya	Rift Valley	Masai Mara N...	Zingabai takli	N/A
819	Greece	South Aegean	Santorini	Oia	84702
820	Turkey	Marmara	Istanbul	Hagia Sophia	34400

Location: Test Case: This SQL query retrieves information about locations and rooms, joining the Location (l) and Room (r) tables based on their Street Address.

SELECT Clause:

•SELECT l.Country, l.Region, l.Street, r.Comment:

- •This clause specifies the columns you want to retrieve from the tables.
- •l.Country: Selects the Country column from the Location table.
- •l.Region: Selects the Region column from the Location table.
- •l.Street: Selects the Street column from the Location table.
- •r.Comment: Selects the Comment column from the Room table.

FROM Clause:

•FROM Location l, Room r:

- •This clause specifies the tables from which you want to retrieve data.
- •Location l: Refers to the Location table and assigns it an alias l.
- •Room r: Refers to the Room table and assigns it an alias r.
- •**Note:** While this method of listing tables in the FROM clause is grammatically correct, using a comma (,) is less common in modern SQL practice. It's generally recommended to use explicit joins (like INNER JOIN) for clarity.

WHERE Clause:

•WHERE l.Street = r.Street:

- •This clause filters the retrieved data based on a specific condition.
- •It ensures that only rows where the Street value in the Location table matches the Street value in the Room table are included in the final result set. This creates a connection between the two tables based on their location information.

```
SELECT l.Country, l.Region, r.Street, r.Comment
FROM Location l, Room r
WHERE l.Street = r.Street;
```

Country	Region	Street	Comment
France	Île-de-France	10 Rue de Rivoli	Spacious room with a comfortable bed, perfect ...
Italy	Tuscany	Piazza della Signoria, 1	Clean and well-maintained, good for a short stay.
Spain	Catalonia	Passeig de Gràcia, 11	Modern and stylish with a stunning view! Highly ...
United Kingdom	England	221B Baker Street	A bit outdated, but location makes up for it.
Japan	Kantō	Shibuya Crossing	Towels could be better, but overall comfortable.
United States	California	1600 Vine St	Great value! Clean and comfortable.
Canada	British Columbia	Granville Island	Quiet and peaceful, perfect for a good night's sl...
Australia	New South Wales	Sydney Opera House	Lovely balcony with a great view and fresh air!
Brazil	Southeast	Copacabana Beach	Shower pressure weak, but overall a good stay.
Mexico	Central Mexico	Zócalo Square	Friendly and helpful staff made the stay even b...
India	Delhi	Taj Mahal	Disappointed with cleanliness. Needs better hou...
China	Beijing	Great Wall of China	Incredibly comfortable bed! Slept like a baby.
South Africa	Western Cape	Table Mountain	A bit noisy from street traffic, but manageable.
Argentina	Buenos Aires	La Boca	Excellent amenities, especially the pool and gym.
New Zealand	South Island	Skippers Canyon Road	Convenient location for exploring the city.
Peru	Cuzco Region	Machu Picchu Historic ...	No air conditioning made the room hot during su...
Iceland	Capital Region	Hallgrímskirkja	Exactly as pictured in the listing. No surprises!
Kenya	Rift Valley	Zingabai takli	Low water pressure in the sink.
Greece	South Aegean	Oia	Delicious complimentary breakfast was a nice su...
Turkey	Marmara	Hagia Sophia	Perfect for solo travelers. Would definitely reco...

Amenity

- The system stores details on different amenities that can be linked to rental property listings.
- Id (Primary key): Unique identifier assigned to each amenity.
- Name: Denotes the title of the amenity.
- Description: Offers a concise overview of the amenity.

```sql
CREATE TABLE Amenity (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(255) NOT NULL,
  Description TEXT
);

 INSERT INTO Amenity (id, Name, Description)VALUES
(901, 'Wi-Fi', 'Free wireless internet access throughout the property.'),
(902, 'Parking', 'On-site parking available (free or for a fee).'),
(903, 'Kitchen', 'Fully equipped kitchen for self-catering.'),
(904, 'Air conditioning', 'Air conditioning to keep you cool during your stay.'),
(905, 'Heating', 'Heating system to keep you warm in colder months.'),
(906, 'Laundry facilities', 'Washing machine and dryer available for guest use.'),
(907, 'Pet-friendly', 'Pets are welcome to stay (restrictions may apply).'),
(908, 'Hot tub', 'Relaxing hot tub available for guest enjoyment.'),
(909, 'Pool', 'Swimming pool on-site for guests to use.'),
(910, 'Gym', 'Fitness center with exercise equipment available.'),
(911, 'Balcony', 'Private balcony with a view for guests to enjoy.'),
(912, 'Fireplace', 'Cozy fireplace to create a warm and inviting atmosphere.'),
(913, 'Cable TV', 'Cable television with a variety of channels for entertainment.'),
(914, 'Beach access', 'Direct access to the beach from the property.'),
(915, 'Airport shuttle', 'Airport shuttle service available (may require a fee).'),
(916, 'BBQ grill', 'BBQ grill available for guests to use for outdoor cooking.'),
(917, 'Babysitting services', 'Babysitting services available upon request (may require a fee).'),
(918, 'Ski-in/ski-out access', 'Direct access to ski slopes from the property.'),
(919, 'Ocean view', 'Breathtaking view of the ocean from the property.'),
(920, 'City view', 'Panoramic view of the city from the property.');
```

id	Name	Description
901	Wi-Fi	Free wireless internet access throughout the pr...
902	Parking	On-site parking available (free or for a fee).
903	Kitchen	Fully equipped kitchen for self-catering.
904	Air conditioning	Air conditioning to keep you cool during your stay.
905	Heating	Heating system to keep you warm in colder mon...
906	Laundry facilities	Washing machine and dryer available for guest ...
907	Pet-friendly	Pets are welcome to stay (restrictions may apply).
908	Hot tub	Relaxing hot tub available for guest enjoyment.
909	Pool	Swimming pool on-site for guests to use.
910	Gym	Fitness center with exercise equipment available.
911	Balcony	Private balcony with a view for guests to enjoy.
912	Fireplace	Cozy fireplace to create a warm and inviting at...
913	Cable TV	Cable television with a variety of channels for e...
914	Beach access	Direct access to the beach from the property.
915	Airport shuttle	Airport shuttle service available (may require a ...
916	BBQ grill	BBQ grill available for guests to use for outdoor ...
917	Babysitting serv...	Babysitting services available upon request (ma...
918	Ski-in/ski-out ac...	Direct access to ski slopes from the property.
919	Ocean view	Breathtaking view of the ocean from the proper...
920	City view	Panoramic view of the city from the property.

Amenity: Test Case: This SQL query attempts to retrieve data about amenities, cancellation reasons, and image URLs, but it likely has some issues due to the way the tables are joined.

SELECT Clause:
The SELECT clause specifies the columns you want to retrieve from the database tables.

•p.Address: Selects the Address column from the Property table (aliased as p).
•a.Description: Selects the Description column from the Amenity table (aliased as a).
•pa.id: Selects the id column from the PropertyAmenity table (aliased as pa).

2. FROM Clause:
The FROM clause specifies the tables involved in the query.

•Property p: This part defines that you're retrieving data from the Property table and assigning it the alias p for easier reference within the query.

3. INNER JOIN Clause:
The INNER JOIN clause combines rows from two or more tables based on a shared condition

•First Join:
•Connects the Property table (aliased as p) with the PropertyAmenity table (aliased as pa).
•Matches rows where the id of a property (p.id) is equal to the property_id in the PropertyAmenity table (pa.property_id). This implies that PropertyAmenity links properties to amenities using a foreign key relationship (assuming they exist).

•Second Join:
•Connects the PropertyAmenity table (aliased as pa) with the Amenity table (aliased as a).
•Matches rows where the amenity_id in the PropertyAmenity table (pa.amenity_id) is equal to the id of an amenity in the Amenity table (a.id). This establishes the relationship between amenities and the PropertyAmenity table (assuming another foreign key).

```
SELECT p.Address, a.Description, pa.id
FROM Property p
INNER JOIN PropertyAmenity pa ON p.id = pa.property_id
INNER JOIN Amenity a ON pa.amenity_id = a.id;
```

Address	Description	id
123 Ocean View Ave...	Free wireless internet access throughout the property.	1601
456 Main St, New Yo...	On-site parking available (free or for a fee).	1602
789 Pine Ridge Rd, ...	Fully equipped kitchen for self-catering.	1603
1011 Freedom St, B...	Air conditioning to keep you cool during your stay.	1604
1213 Evergreen Dr, ...	Heating system to keep you warm in colder months.	1605
1415 Country Lane, ...	Washing machine and dryer available for guest use.	1606
1617 Palm Tree Way...	Pets are welcome to stay (restrictions may apply).	1607
1819 Industrial Ave,...	Relaxing hot tub available for guest enjoyment.	1608
2021 Lakeside Dr, L...	Swimming pool on-site for guests to use.	1609
2223 Cactus Way, P...	Fitness center with exercise equipment available.	1610
2425 Elm St, Portlan...	Private balcony with a view for guests to enjoy.	1611
2627 Seashell Dr, O...	Cozy fireplace to create a warm and inviting atmosph...	1612
2829 Winding Path, ...	Cable television with a variety of channels for entert...	1613
3031 Grapevine Ln, ...	Direct access to the beach from the property.	1614
3233 Spruce Peak R...	Airport shuttle service available (may require a fee).	1615
3435 cobblestone St...	BBQ grill available for guests to use for outdoor cooki...	1616
3637 Canal St, Amst...	Babysitting services available upon request (may req...	1617
3839 Aspen Ridge D...	Direct access to ski slopes from the property.	1618
4041 Rainforest Wa...	Breathtaking view of the ocean from the property.	1619
4243 Castaway Cay...	Panoramic view of the city from the property.	1620

Room

This code creates a table named Room to store ratings and comments for individual rooms. Its likely links to another table (not shown) as it doesn't have a room identifier. It then adds sample data with ratings and comments.

```sql
CREATE TABLE Room (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Rating DECIMAL(10,2),
  Comment TEXT,
  Street VARCHAR(255)
);
INSERT INTO Room (id, Rating, Comment, Street)VALUES
(1001, 5.0, "Spacious room with a comfortable bed, perfect for relaxation!", '10 Rue de Rivoli'),
(1002, 4.5, "Clean and well-maintained, good for a short stay.", 'Piazza della Signoria, 1'),
(1003, 5.0, "Modern and stylish with a stunning view! Highly recommend!", 'Passeig de Gràcia, 11'),
(1004, 4.0, "A bit outdated, but location makes up for it.", '221B Baker Street'),
(1005, 3.5, "Towels could be better, but overall comfortable.", 'Shibuya Crossing'),
(1006, 5.0, "Great value! Clean and comfortable.", '1600 Vine St'),
(1007, 4.5, "Quiet and peaceful, perfect for a good night's sleep.", 'Granville Island'),
(1008, 5.0, "Lovely balcony with a great view and fresh air!", 'Sydney Opera House'),
(1009, 4.0, "Shower pressure weak, but overall a good stay.", 'Copacabana Beach'),
(1010, 5.0, "Friendly and helpful staff made the stay even better!", 'Zócalo Square'),
(1011, 3.0, "Disappointed with cleanliness. Needs better housekeeping.", 'Taj Mahal'),
(1012, 5.0, "Incredibly comfortable bed! Slept like a baby.", 'Great Wall of China'),
(1013, 4.0, "A bit noisy from street traffic, but manageable.", 'Table Mountain'),
(1014, 5.0, "Excellent amenities, especially the pool and gym.", 'La Boca'),
(1015, 4.5, "Convenient location for exploring the city.", 'Skippers Canyon Road'),
(1016, 3.0, "No air conditioning made the room hot during summer.", 'Machu Picchu Historic Sanctuary'),
(1017, 5.0, "Exactly as pictured in the listing. No surprises!", 'Hallgrímskirkja'),
(1018, 4.0, "Low water pressure in the sink.", 'Zingabai takli'),
(1019, 5.0, "Delicious complimentary breakfast was a nice surprise!", 'Oia'),
(1020, 4.5, "Perfect for solo travelers. Would definitely recommend!", 'Hagia Sophia');
```

id	Rating	Comment	Street
1001	5.00	Spacious room with a comfortable bed, perfect ...	10 Rue de Rivoli
1002	4.50	Clean and well-maintained, good for a short stay.	Piazza della Signoria, 1
1003	5.00	Modern and stylish with a stunning view! Highly ...	Passeig de Gràcia, 11
1004	4.00	A bit outdated, but location makes up for it.	221B Baker Street
1005	3.50	Towels could be better, but overall comfortable.	Shibuya Crossing
1006	5.00	Great value! Clean and comfortable.	1600 Vine St
1007	4.50	Quiet and peaceful, perfect for a good night's sl...	Granville Island
1008	5.00	Lovely balcony with a great view and fresh air!	Sydney Opera House
1009	4.00	Shower pressure weak, but overall a good stay.	Copacabana Beach
1010	5.00	Friendly and helpful staff made the stay even b...	Zócalo Square
1011	3.00	Disappointed with cleanliness. Needs better hou...	Taj Mahal
1012	5.00	Incredibly comfortable bed! Slept like a baby.	Great Wall of China
1013	4.00	A bit noisy from street traffic, but manageable.	Table Mountain
1014	5.00	Excellent amenities, especially the pool and gym.	La Boca
1015	4.50	Convenient location for exploring the city.	Skippers Canyon Road
1016	3.00	No air conditioning made the room hot during su...	Machu Picchu Historic ...
1017	5.00	Exactly as pictured in the listing. No surprises!	Hallgrímskirkja
1018	4.00	Low water pressure in the sink.	Zingabai takli
1019	5.00	Delicious complimentary breakfast was a nice su...	Oia
1020	4.50	Perfect for solo travelers. Would definitely reco...	Hagia Sophia

Room: Test: Case: The provided SQL query retrieves specific data from two tables and filters the results based on a common column.

SELECT Clause:

•SELECT l.City, r.id, r.Comment, r.Rating:

- •This clause specifies the columns you want to retrieve from the tables.
- •l.City: Selects the City column from the Location table (l).
- •r.id: Selects the id column from the Room table (r).
- •r.Comment: Selects the Comment column from the Room table (r).
- •r.Rating: Selects the Rating column from the Room table (r).

FROM Clause:

•FROM Location l, Room r:

- •This clause specifies the tables from which you want to retrieve data.
- •Location l: Refers to the Location table and assigns it an alias l.
- •Room r: Refers to the Room table and assigns it an alias r.
- •**Note:** While grammatically correct, using a comma (,) to separate tables in the FROM clause is less common in modern SQL practice. It's generally recommended to use explicit joins (like INNER JOIN) for clarity.

WHERE Clause:

•WHERE l.Street = r.Street:

- •This clause filters the retrieved data based on a specific condition.
- •However, using Street for the join condition might not be ideal.
  - •A single street might exist in multiple locations (e.g., "Main Street").
  - •This condition could return a large number of unrelated rows.

```
SELECT l.City, r.id, r.Comment, r.Rating
FROM Location l, Room r
WHERE l.Street = r.Street;
```

City	id	Comment	Rating
Paris	1001	Spacious room with a comfortable bed, perfect ...	5.00
Florence	1002	Clean and well-maintained, good for a short stay.	4.50
Barcelona	1003	Modern and stylish with a stunning view! Highly ...	5.00
London	1004	A bit outdated, but location makes up for it.	4.00
Tokyo	1005	Towels could be better, but overall comfortable.	3.50
Los Angeles	1006	Great value! Clean and comfortable.	5.00
Vancouver	1007	Quiet and peaceful, perfect for a good night's sl...	4.50
Sydney	1008	Lovely balcony with a great view and fresh air!	5.00
Rio de Janeiro	1009	Shower pressure weak, but overall a good stay.	4.00
Mexico City	1010	Friendly and helpful staff made the stay even b...	5.00
Delhi	1011	Disappointed with cleanliness. Needs better hou...	3.00
Beijing	1012	Incredibly comfortable bed! Slept like a baby.	5.00
Cape Town	1013	A bit noisy from street traffic, but manageable.	4.00
Buenos Aires	1014	Excellent amenities, especially the pool and gym.	5.00
Queenstown	1015	Convenient location for exploring the city.	4.50
Machu Picchu	1016	No air conditioning made the room hot during su...	3.00
Reykjavik	1017	Exactly as pictured in the listing. No surprises!	5.00
Masai Mara N...	1018	Low water pressure in the sink.	4.00
Santorini	1019	Delicious complimentary breakfast was a nice su...	5.00
Istanbul	1020	Perfect for solo travelers. Would definitely reco...	4.50

Image

Store and manage essential image information

```sql
CREATE TABLE Image (
    id INT PRIMARY KEY AUTO_INCREMENT,
    URL VARCHAR(255) NOT NULL UNIQUE,
    Name VARCHAR(255) NOT NULL
);
```

This code creates a table named Image to store references to images. It has an ID (unique identifier) and a URL column to store the image location. It then adds sample data with URLs pointing to general image searches on Pexels.

- Use INSERT INTO statements to add image data to the table

```sql
INSERT INTO Image (id, URL, Name)VALUES
(731,'https://photos.app.goo.gl/jFF5oMmLpLKP9DdWA', 'Cozy Beachfront Cottage'),
(732,'https://photos.app.goo.gl/waT1GraKDCFJgnRK9', 'Modern City Apartment'),
(733,'https://photos.app.goo.gl/HM1DtF2tQmLGdwG16', 'Secluded Mountain Cabin'),
(734,'https://photos.app.goo.gl/kxEefNBr7umXiUnL6', 'Historic Townhouse'),
(735,'https://photos.app.goo.gl/dv62YeusVBwVtdsx6', 'Luxurious Ski Chalet'),
(736,'https://photos.app.goo.gl/PXNoCb3V1YaL2SMk6', 'Charming Farmhouse'),
(737,'https://photos.app.goo.gl/m57oRM96r5EXtVdg7', 'Tropical Island Bunglow'),
(738,'https://photos.app.goo.gl/XaDYCFr2KB6MJ4Ht5', '   Designer Loft Apartment'),
(739,'https://photos.app.goo.gl/fEjommec6GVKxbjh9', 'Rustic Lakeside Cabin'),
(740,'https://photos.app.goo.gl/xMka4LwQWCN7cKwL7', 'Desert Oasis Retreat'),
(741,'https://photos.app.goo.gl/P8AFLtWvv1jKtRHW9', 'Cozy Tiny House'),
(742,'https://photos.app.goo.gl/LGuRfVVdD2T1cURD8', 'Family-Friendly Beach House'),
(743,'https://photos.app.goo.gl/bRz4D8zinuDdK2JG8', 'Modern Treehouse Getaway'),
(744,'https://photos.app.goo.gl/m7AtToQeBVDhpztu6', 'Private Vineyard Cottage'),
(745,'https://photos.app.goo.gl/tfsVRUXjLQYmmbxP8', 'Ski-in/Ski-out Condo'),
(746,'https://photos.app.goo.gl/bDDf2ACRt9tZVhBG6', 'Historic City Centre Apartment'),
(747,'https://photos.app.goo.gl/LoVp5vuXcFdsRFNv9', 'Canal-side Townhouse'),
(748,'https://photos.app.goo.gl/wMCbVFN3dP84k1cYA', 'Luxury Mountain Lodge'),
(749,'https://photos.app.goo.gl/eTzmWXChR8MvQ2D28', 'Jungle Mountain Adventure'),
(750,'https://photos.app.goo.gl/iUri6utwkXarkjRo6', 'Private Island Escape');
```

id	URL	Name
731	https://photos.app.goo.gl/jFF5oMmLpLKP9DdWA	Cozy Beachfront Cottage
732	https://photos.app.goo.gl/waT1GraKDCFJgnRK9	Modern City Apartment
733	https://photos.app.goo.gl/HM1DtF2tQmLGdwG16	Secluded Mountain Cabin
734	https://photos.app.goo.gl/kxEefNBr7umXiUnL6	Historic Townhouse
735	https://photos.app.goo.gl/dv62YeusVBwVtdsx6	Luxurious Ski Chalet
736	https://photos.app.goo.gl/PXNoCb3V1YaL2SMk6	Charming Farmhouse
737	https://photos.app.goo.gl/m57oRM96r5EXtVdg7	Tropical Island Bunglow
738	https://photos.app.goo.gl/XaDYCFr2KB6MJ4Ht5	Designer Loft Apartment
739	https://photos.app.goo.gl/fEjommec6GVKxbjh9	Rustic Lakeside Cabin
740	https://photos.app.goo.gl/xMka4LwQWCN7cKwL7	Desert Oasis Retreat
741	https://photos.app.goo.gl/P8AFLtWvv1jKtRHW9	Cozy Tiny House
742	https://photos.app.goo.gl/LGuRfVVdD2T1cURD8	Family-Friendly Beach H...
743	https://photos.app.goo.gl/bRz4D8zinuDdK2JG8	Modern Treehouse Geta...
744	https://photos.app.goo.gl/m7AtToQeBVDhpztu6	Private Vineyard Cottage
745	https://photos.app.goo.gl/tfsVRUXjLQYmmbxP8	Ski-in/Ski-out Condo
746	https://photos.app.goo.gl/bDDf2ACRt9tZVhBG6	Historic City Centre Apar...
747	https://photos.app.goo.gl/LoVp5vuXcFdsRFNv9	Canal-side Townhouse
748	https://photos.app.goo.gl/wMCbVFN3dP84k1cYA	Luxury Mountain Lodge
749	https://photos.app.goo.gl/eTzmWXChR8MvQ2...	Jungle Mountain Advent...
750	https://photos.app.goo.gl/iUri6utwkXarkjRo6	Private Island Escape

Image: Test: Case: SQL query retrieves information about image URLs and property amenities, but it might have some issues due to the join method.

```
SELECT p.Amenities, i.URL
FROM Image i, Property p
WHERE i.Name = p.Name;
```

SELECT Clause:

•SELECT p.Amenities, i.URL:

- •This clause specifies the columns you want to retrieve from the tables.
- •p.Amenities: Selects the Amenities column from the Property table (p).
- •i.URL: Selects the URL column from the Image table (i).

FROM Clause:

•FROM Image i, Property p:

- •This clause specifies the tables from which you want to retrieve data.
- •Image i: Refers to the Image table and assigns it an alias i.
- •Property p: Refers to the Property table and assigns it an alias p.
- •**Note:** While grammatically correct, using a comma (,) to separate tables in the FROM clause is less common in modern SQL practice. It's generally recommended to use explicit joins (like INNER JOIN) for clarity.

WHERE Clause:

•WHERE i.Name = p.Name:

- •This clause filters the retrieved data based on a specific condition.
- •It joins the tables based on the assumption that the Name column in both tables refers to the same property. However, this might not be a reliable join condition. Property names could be duplicated, leading to inaccurate results.

Amenities	URL
Beachfront, Wi-Fi, Par...	https://photos.app.goo.gl/jFF5oMmLpLKP9DdWA
City Views, Gym, Roof...	https://photos.app.goo.gl/waT1GraKDCFJgnRK9
Hot Tub, Fireplace, Hi...	https://photos.app.goo.gl/HM1DtF2tQmLGdwG16
Walkable Location, Pa...	https://photos.app.goo.gl/kxEefNBr7umXiUnL6
Ski-in/Ski-out, Sauna, ...	https://photos.app.goo.gl/dv62YeusVBwVtdsx6
Vineyard Views, Pool, ...	https://photos.app.goo.gl/PXNoCb3V1YaL2SMk6
Lakefront, Fishing, Ka...	https://photos.app.goo.gl/fEjommec6GVKxbjh9
Pool, Hot Tub, Mounta...	https://photos.app.goo.gl/xMka4LwQWCN7cKwL7
Minimalist Design, Eco-...	https://photos.app.goo.gl/P8AFLtWvv1jKtRHW9
Beach Access, Game ...	https://photos.app.goo.gl/LGuRfVVdD2T1cURD8
Unique Experience, Fo...	https://photos.app.goo.gl/bRz4D8zinuDdK2JG8
Vineyard Tour Include...	https://photos.app.goo.gl/m7AtToQeBVDhpztu6
Slopeside Location, Ski...	https://photos.app.goo.gl/tfsVRUXjLQYmmbxP8
Canal Views, Boat Tou...	https://photos.app.goo.gl/LoVp5vuXcFdsRFNv9
Spa Services, Gourme...	https://photos.app.goo.gl/wMCbVFN3dP84k1cYA
Secluded Paradise, Be...	https://photos.app.goo.gl/iUri6utwkXarkjRo6

Message

Store and manage essential message information

- Use INSERT INTO statements to add message data to the table

```sql
CREATE TABLE Message (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Sender_id INT NOT NULL,
  Receiver_id INT NOT NULL,
  Content TEXT,
  Timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (Sender_id) REFERENCES User(id),
  FOREIGN KEY (Receiver_id) REFERENCES User(id)
);
INSERT INTO Message (id, Sender_id, Receiver_id, Content, Timestamp)VALUES
(12524, 1,1, 'Hi there!', '2024-05-21 01:27:00'),
(22524, 2, 2, 'Hello! ', '2024-05-21 01:27:01'),
(32524, 3, 3, 'This is message number 3.', '2024-05-21 01:27:02'),
(42524, 4, 4, 'Sending another message.', '2024-05-21 01:27:03'),
(52524, 5, 5, 'How are you doing today?', '2024-05-21 01:27:04'),
(62524, 6, 6, 'I am doing well, thanks for asking!', '2024-05-21 01:27:05'),
(72524, 7, 7, 'Great to hear!', '2024-05-21 01:27:06'),
(82524, 8, 8, 'Hey! ', '2024-05-21 01:27:07'),
(92524, 9, 9, 'What\'s up?', '2024-05-21 01:27:08'),
(102524, 10, 10, 'Not much, just hanging out.', '2024-05-21 01:27:09'),
(112524, 11, 11, 'Sounds like fun!', '2024-05-21 01:27:10'),
(122524, 12, 12, 'Yeah, it is!', '2024-05-21 01:27:11'),
(132524, 13, 13, 'This is another message from user 789.', '2024-05-21 01:27:12'),
(142524, 14, 14, 'Replying to message 13.', '2024-05-21 01:27:13'),
(152524, 15, 15, 'How is everyone doing this fine Tuesday?', '2024-05-21 01:27:14'),
(162524, 16, 16, 'Doing great! ', '2024-05-21 01:27:15'),
(172524, 17, 17, 'Cool! ', '2024-05-21 01:27:16'),
(182524, 18, 18, 'This is message number 18.', '2024-05-21 01:27:17'),
(192524, 19, 19, 'Hi there! How are you today?', '2024-05-21 01:27:18'),
(202524, 20, 20, 'Replied to message 18.', '2024-05-21 01:27:19');
```

id	Sender_id	Receiver_id	Content	Timestamp
12524	1	1	Hi there!	2024-05-21 01:27:00
22524	2	2	Hello!	2024-05-21 01:27:01
32524	3	3	This is message number 3.	2024-05-21 01:27:02
42524	4	4	Sending another message.	2024-05-21 01:27:03
52524	5	5	How are you doing today?	2024-05-21 01:27:04
62524	6	6	I am doing well, thanks for asking!	2024-05-21 01:27:05
72524	7	7	Great to hear!	2024-05-21 01:27:06
82524	8	8	Hey!	2024-05-21 01:27:07
92524	9	9	What's up?	2024-05-21 01:27:08
102524	10	10	Not much, just hanging out.	2024-05-21 01:27:09
112524	11	11	Sounds like fun!	2024-05-21 01:27:10
122524	12	12	Yeah, it is!	2024-05-21 01:27:11
132524	13	13	This is another message from use...	2024-05-21 01:27:12
142524	14	14	Replying to message 13.	2024-05-21 01:27:13
152524	15	15	How is everyone doing this fine T...	2024-05-21 01:27:14
162524	16	16	Doing great!	2024-05-21 01:27:15
172524	17	17	Cool!	2024-05-21 01:27:16
182524	18	18	This is message number 18.	2024-05-21 01:27:17
192524	19	19	Hi there! How are you today?	2024-05-21 01:27:18
202524	20	20	Replied to message 18.	2024-05-21 01:27:19

Message: Test: Case: SELECT sender_id, receiver_id, content, COUNT(*) AS number_of_links FROM airbnb.message GROUP BY sender_id, receiver_id, content;

SELECT Clause:

- SELECT u.Email, h.Listing_status, p.Amenities, m.Content:
  - This clause specifies the columns you want to retrieve from the tables.
  - u.Email: Selects the Email column from the User table (u).
  - h.Listing_status: Selects the Listing_status column from the Host table (h).
  - p.Amenities: Selects the Amenities column from the Property table (p).
  - m.Content: Selects the Content column from the Message table (m).

FROM Clause:

- FROM User u:
  - Specifies the starting table as User and assigns it the alias u.

INNER JOIN Clauses:

- Three INNER JOIN clauses are used to connect the tables based on shared columns:
  1. INNER JOIN Host h ON u.id = h.user_id:
     - Joins the User and Host tables.
     - Matches rows where the id in the User table (u.id) is equal to the user_id in the Host table (h.user_id). This implies a user can be a host.
  2. INNER JOIN Property p ON h.id = p.Host_id:
     - Joins the Host and Property tables.
     - Matches rows where the id in the Host table (h.id) is equal to the Host_id in the Property table (p.Host_id). This implies a host can have properties.
  3. INNER JOIN Message m ON u.id = m.Sender_id OR h.id = m.Receiver_id:
     - Joins the tables based on a more complex condition using the OR operator.
     - Matches rows in the Message table (m) where:
       - u.id (user ID) is equal to m.Sender_id (message sender ID) or
       - h.id (host ID) is equal to m.Receiver_id (message receiver ID).
     - This captures messages sent by users or received by hosts.

```
SELECT u.Email, h.Listing_status, p.Amenities, m.Content
FROM User u
INNER JOIN Host h ON u.id = h.user_id
INNER JOIN Property p ON h.id = p.Host_id
INNER JOIN Message m ON u.id = m.Sender_id OR h.id m.Receiver_id;
```

Email	Listing_status	Amenities	Content
aisha.khan@example.com	inactive	Vineyard Views, Pool, ...	I am doing well, thanks for asking!
alex.schmidt@example.com	active	Ski-in/Ski-out, Sauna, ...	How are you doing today?
anna.schmidt@example.com	active	Slopeside Location, Ski...	How is everyone doing this fine Tuesday?
antonio.garcia@example.com	inactive	Secluded Paradise, Be...	Replied to message 18.
david.rodriguez@example.com	active	Oceanfront, Private B...	Great to hear!
elena.volkova@example.com	active	Immerse Yourself in N...	Hi there! How are you today?
emily.chen@example.com	active	Minimalist Design, Eco-...	Sounds like fun!
jane.smith@example.com	inactive	City Views, Gym, Roof...	Hello!
li.wang@example.com	inactive	Walkable Location, Pa...	Sending another message.
maria.garcia@example.com	active	Hot Tub, Fireplace, Hi...	This is message number 3.
marie.kim@example.com	active	Unique Experience, Fo...	This is another message from user 789.
miguel.sanchez@example.com	inactive	Vineyard Tour Include...	Replying to message 13.
natalia.petrova@example.com	inactive	Exposed Brick, Balcon...	Hey!
omar.syed@example.com	active	Lakefront, Fishing, Ka...	What's up?
pierre.martin@example.com	inactive	Spa Services, Gourme...	This is message number 18.
REDACTED_EMAIL@example....	inactive	French Quarter Balcon...	Doing great!
sarah.lee@example.com	active	Canal Views, Boat Tou...	Cool!
sophie.dupont@example.com	inactive	Pool, Hot Tub, Mounta...	Not much, just hanging out.
william.campbell@example.com	active	Beachfront, Wi-Fi, Par...	Hi there!
william.muller@example.com	inactive	Beach Access, Game ...	Yeah, it is!

Userauthentication

This code creates a table for user authentication and inserts some sample data.

```sql
CREATE TABLE UserAuthentication (
    id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT NOT NULL,
    Username VARCHAR(255) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL,
    FOREIGN KEY (User_id) REFERENCES User(id)
);
```

- Use INSERT INTO statements to add userauthentication data to the table

```sql
INSERT INTO UserAuthentication (id, User_id, Username, Password)VALUES
(1101, 1, 'user1', 'password1'),
(1102, 2, 'user2', 'password2'),
(1103, 3, 'user3', 'password3'),
(1104, 4, 'user4', 'password4'),
(1105, 5, 'user5', 'password5'),
(1106, 6, 'user6', 'password6'),
(1107, 7, 'user7', 'password7'),
(1108, 8, 'user8', 'password8'),
(1109, 9, 'user9', 'password9'),
(1110, 10, 'user10', 'password10'),
(1111, 11, 'user11', 'password11'),
(1112, 12, 'user12', 'password12'),
(1113, 13, 'user13', 'password13'),
(1114, 14, 'user14', 'password14'),
(1115, 15, 'user15', 'password15'),
(1116, 16, 'user16', 'password16'),
(1117, 17, 'user17', 'password17'),
(1118, 18, 'user18', 'password18'),
(1119, 19, 'user19', 'password19'),
(1120, 20, 'user20', 'password20');
```

Table Creation:

•Creates a table named UserAuthentication.

•Defines four columns:

- •id: Unique identifier (auto-incrementing integer).
- •User_id: Foreign key referencing another table (User) likely for additional user information.
- •Username: Username for login (unique).
- •Password: User password (not stored in plain text but hashed for security).

id	User_id	Username	Password
1101	1	user1	password1
1102	2	user2	password2
1103	3	user3	password3
1104	4	user4	password4
1105	5	user5	password5
1106	6	user6	password6
1107	7	user7	password7
1108	8	user8	password8
1109	9	user9	password9
1110	10	user10	password10
1111	11	user11	password11
1112	12	user12	password12
1113	13	user13	password13
1114	14	user14	password14
1115	15	user15	password15
1116	16	user16	password16
1117	17	user17	password17
1118	18	user18	password18
1119	19	user19	password19
1120	20	user20	password20

Userauthentication:Test :Case: This query retrieves user information including their role and wishlist ID, along with optional host property ID and password (if they're a host).

```
SELECT
u.role,  h.property_id,  ua.password,
w.id AS wishlist_id
FROM User u
LEFT JOIN Host h ON u.id =
h.user_id
LEFT JOIN UserAuthentication ua
ON u.id = ua.User_id
LEFT JOIN Wishlist w ON u.id =
w.User_id
```

•**SELECT:** Picks specific columns: role (from User), property_id (from Host), password (from UserAuthentication), wishlist_id (from Wishlist)
.
•**FROM:** Starts with the User table (aliased as u).

•**LEFT JOIN:**
•Connects User to Host (aliased as h) if u.id matches h.user_id (gets host info if user is a host).
•Connects User to UserAuthentication (aliased as ua) if u.id matches ua.User_id (gets password if it exists).
•Connects User to Wishlist (aliased as w) if u.id matches w.User_id (gets wishlist ID, keeps it even if user doesn't have a wishlist).

role	property_id	password	wishlist_id
Host	201	password1	31
Host	202	password2	32
Admin	203	password3	33
Traveler	204	password4	34
Host	205	password5	35
Guest	206	password6	36
Traveler	207	password7	37
Host	208	password8	38
Guest	209	password9	39
Traveler	210	password10	40
Admin	211	password11	41
Host	212	password12	42
Host (C…	213	password13	43
Guest	214	password14	44
Traveler	215	password15	45
Guest	216	password16	46
Host	217	password17	47
Traveler	218	password18	48
Guest	219	password19	49
Host	220	password20	50

Bookingreview

```sql
CREATE TABLE BookingReview (
    Booking_id INT NOT NULL,
    Guest_id INT NOT NULL,
    review_id INT NOT NULL,
    Review_text TEXT,
    Timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (Booking_id, Guest_id, Review_id),
    FOREIGN KEY (Booking_id) REFERENCES Booking(id),
    FOREIGN KEY (Guest_id) REFERENCES Guest(id),
    FOREIGN KEY (Review_id) REFERENCES Review(id)
);
```

This code creates a table to store booking reviews and inserts some sample data.
- Connects bookings, guests, and their reviews.
- Stores review text, timestamp, and foreign keys to other tables.
- Sample reviews with timestamps are included.

- Use INSERT INTO statements to add bookingreview data to the table

```sql
INSERT INTO BookingReview (Booking_id, Guest_id, Review_id, Review_text, Timestamp)VALUES
(401, 301, 501, 'The apartment was clean and spacious, and the host was very responsive.', '2024-05-20 10:20:00'),
(402, 302, 502, 'The location was perfect, close to all the main attractions.', '2024-05-19 15:30:00'),
(403, 303, 503, 'The bed was comfortable, and the bathroom was well-stocked.', '2024-05-18 20:45:00'),
(404, 304, 504, 'We had a great time staying here! We would definitely recommend it to others.', '2024-05-17 12:10:00'),
(405, 305, 505, 'The property was exactly as described in the listing.', '2024-05-16 08:25:00'),
(406, 306, 506, 'The host was very helpful and accommodating.', '2024-05-15 17:40:00'),
(407, 307, 507, 'The place was clean and well-maintained.', '2024-05-14 10:55:00'),
(408, 308, 508, 'We would definitely stay here again!', '2024-05-13 20:10:00'),
(409, 309, 509, 'The view from the property was amazing.', '2024-05-12 12:25:00'),
(410, 310, 510, 'The pool was a great way to relax after a long day of sightseeing.', '2024-05-11 08:40:00'),
(411, 311, 511, 'The communication with the host was excellent.', '2024-05-10 17:55:00'),
(412, 312, 512, 'The property was in a great location for exploring the city.', '2024-05-09 11:10:00'),
(413, 313, 513, 'We had a wonderful stay here! Thank you for everything.', '2024-05-08 20:25:00'),
(414, 314, 514, 'The check-in process was smooth and easy.', '2024-05-07 12:40:00'),
(415, 315, 515, 'The property was exactly as pictured in the listing.', '2024-05-06 08:55:00'),
(416, 316, 516, 'This place was a hidden gem! We will be back for sure.', '2024-05-05 11:00:00'),
(417, 317, 517, 'Overall, we had a pleasant stay.', '2024-05-04 17:15:00'),
(418, 318, 518, 'The amenities were great, just as advertised.', '2024-05-03 10:30:00'),
(419, 319, 519, 'We had a minor issue, but the host resolved it quickly.', '2024-05-02 18:45:00'),
```

Booking_id	Guest_id	review_id	Review_text	Timestamp
401	301	501	The apartment was clean and spacious, and the...	2024-05-20 10:20:00
402	302	502	The location was perfect, close to all the main a...	2024-05-19 15:30:00
403	303	503	The bed was comfortable, and the bathroom w...	2024-05-18 20:45:00
404	304	504	We had a great time staying here! We would de...	2024-05-17 12:10:00
405	305	505	The property was exactly as described in the lis...	2024-05-16 08:25:00
406	306	506	The host was very helpful and accommodating.	2024-05-15 17:40:00
407	307	507	The place was clean and well-maintained.	2024-05-14 10:55:00
408	308	508	We would definitely stay here again!	2024-05-13 20:10:00
409	309	509	The view from the property was amazing.	2024-05-12 12:25:00
410	310	510	The pool was a great way to relax after a long ...	2024-05-11 08:40:00
411	311	511	The communication with the host was excellent.	2024-05-10 17:55:00
412	312	512	The property was in a great location for explori...	2024-05-09 11:10:00
413	313	513	We had a wonderful stay here! Thank you for e...	2024-05-08 20:25:00
414	314	514	The check-in process was smooth and easy.	2024-05-07 12:40:00
415	315	515	The property was exactly as pictured in the listi...	2024-05-06 08:55:00
416	316	516	This place was a hidden gem! We will be back fo...	2024-05-05 11:00:00
417	317	517	Overall, we had a pleasant stay.	2024-05-04 17:15:00
418	318	518	The amenities were great, just as advertised.	2024-05-03 10:30:00
419	319	519	We had a minor issue, but the host resolved it q...	2024-05-02 18:45:00
420	320	520	Great value for the price!	2024-05-01 12:00:00

Bookingreview: Test: Case: this query retrieves guest information, review details, and booking information from four interconnected tables: Guest, Review, BookingReview, and Booking.

SELECT clause:
This clause specifies which columns you want to retrieve from the tables involved in the query. In your example, it selects:
•g.User_id: User ID from the Guest table.
•r.Rating: Rating given in the review from the Review table.
•br.review_id: Unique identifier of the review from the BookingReview table.
•br.Review_text: Review text from the BookingReview table.
•b.guest_id: Guest ID from the Booking table.
FROM clause:
This clause specifies the tables from which you want to retrieve data. In your example, it references:
•Guest table (aliased as g)
•Review table (aliased as r)
•BookingReview table (aliased as br)
•Booking table (aliased as b)
INNER JOIN clause:
This clause is used to combine rows from two or more tables based on a shared relationship. Your query uses three inner joins:
1.Guest and Review: Connects rows where the Guest_id in the Guest table matches the Guest_id in the Review table.
2.Review and BookingReview: Connects rows where the id in the Review table matches the Review_id in the BookingReview table.
3.BookingReview and Booking: Connects rows where the Booking_id in the BookingReview table matches the id (corrected from BookingReview_id) in the Booking table.

```
SELECT g.User_id, r.Rating, br.review_id,
br.Review_text, b.guest_idFROM Guest
gINNER JOIN Review r ON g.id =
r.Guest_idINNER JOIN BookingReview br ON
r.id = br.Review_idINNER JOIN Booking b ON
br.Booking_id = b.id;
```

User_id	Rating	review_id	Review_text	guest_id
1	5	501	The apartment was clean and spacious, and the...	301
2	4	502	The location was perfect, close to all the main a...	302
3	5	503	The bed was comfortable, and the bathroom w...	303
4	4	504	We had a great time staying here! We would de...	304
5	5	505	The property was exactly as described in the lis...	305
6	5	506	The host was very helpful and accommodating.	306
7	5	507	The place was clean and well-maintained.	307
8	5	508	We would definitely stay here again!	308
9	5	509	The view from the property was amazing.	309
10	4	510	The pool was a great way to relax after a long ...	310
11	5	511	The communication with the host was excellent.	311
12	5	512	The property was in a great location for explori...	312
13	5	513	We had a wonderful stay here! Thank you for e...	313
14	5	514	The check-in process was smooth and easy.	314
15	5	515	The property was exactly as pictured in the listi...	315
16	4	516	This place was a hidden gem! We will be back fo...	316
17	3	517	Overall, we had a pleasant stay.	317
18	4	518	The amenities were great, just as advertised.	318
19	3	519	We had a minor issue, but the host resolved it q...	319
20	5	520	Great value for the price!	320

Propertytype

```sql
CREATE TABLE PropertyType (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Type VARCHAR(255) NOT NULL UNIQUE
);
```

This code creates a table for property types and inserts some sample data.
•Stores different property types (e.g., apartment, house).
•Each type is unique.

```sql
INSERT INTO PropertyType (id, Type) VALUES
(221, 'Apartment'),
(222, 'House'),
(223, 'Studio'),
(224, 'Villa'),
(225, 'Cabin'),
(226, 'Boat'),
(227, 'Hostel'),
(228, 'Guest Suite'),
(229, 'Castle'),
(230, 'Treehouse'),
(231, 'Loft'),
(232, 'Beach House'),
(233, 'Boutique Hotel'),
(234, 'Riad'),
(235, 'Farmhouse'),
(236, 'Luxury Tent'),
(237, 'Monastery'),
(238, 'Yurt'),
(239, 'Dome'),
(240, 'Motorhome');
```

- **Table Creation (CREATE TABLE PropertyType (...)):**
  - Defines a table named PropertyType.
- **Columns (id INT PRIMARY KEY AUTO_INCREMENT, Type VARCHAR(255) NOT NULL UNIQUE):**
  - id: This is an integer column that uniquely identifies each property type (primary key). The value will automatically increase (auto-increment) for each new entry.
  - Type: This column stores the actual property type as text (e.g., "Apartment", "House"). It cannot be empty (NOT NULL) and must be unique (UNIQUE) to avoid duplicates.
- **Data Insertion (INSERT INTO PropertyType (...) VALUES (...)):**
  - Inserts sample data into the table. This creates multiple rows, each with a unique ID and a specific property type (e.g., Apartment, House, etc.).

In essence, this code defines and populates a list of possible property types available for booking.

id	Type
221	Apartment
232	Beach House
226	Boat
233	Boutique Hotel
225	Cabin
229	Castle
239	Dome
235	Farmhouse
228	Guest Suite
227	Hostel
222	House
231	Loft
236	Luxury Tent
237	Monastery
240	Motorhome
234	Riad
223	Studio
230	Treehouse
224	Villa
238	Yurt

Propertytype

Type	Address	Property_id	Location_id	Country	Region
Dome	4041 Rainforest Way, Costa Rica	219	819	Greece	South Aegean

SELECT Clause:

- **SELECT pt.Type, p.Address, p.id, l.id, l.Country, l.Region**
  - This clause specifies the columns that you want to retrieve from the database tables.
  - pt.Type: Selects the Type column from the PropertyType table (pt).
  - p.Address: Selects the Address column from the Property table (p).
  - p.id: Selects the id column from the Property table (p).
  - l.id: Selects the id column from the Location table (l).
  - l.Country: Selects the Country column from the Location table (l).
  - l.Region: Selects the Region column from the Location table (l).

FROM Clause:

- **FROM PropertyType pt, Property p, Location l**
  - This clause specifies the database tables from which you want to retrieve data.
  - PropertyType pt: Refers to the PropertyType table.
  - Property p: Refers to the Property table.
  - Location l: Refers to the Location table.
  - **Note:** The comma (,) separates the tables, indicating that the query will retrieve data from all three tables.

WHERE Clause:

- **WHERE p.id = '219' AND pt.Type = 'Dome' AND l.id= '819'**
  - This clause filters the retrieved data based on specific conditions.
  - p.id = '219': Filters the Property table to include only rows where the id column is equal to '219'.
  - AND pt.Type = 'Dome': Further restricts the results from the Property table to include only rows where the Type column is equal to 'Dome'.
  - AND l.id= '819': Additionally filters the Location table to include only rows where the id column is equal to '819'.
  - **Note:** The AND operator ensures that all three conditions must be met for a row to be included in the final result set.

```
SELECT pt.Type, p.Address,
p.id AS Property_id,
l.id AS Location_id, l.Country,
l.Region
FROM PropertyType pt, Property p,
Location l
WHERE p.id = '219'
AND pt.Type = 'Dome'
AND l.id= '819';
```

Paymentmethod

This code creates a table to store user payment methods and inserts some sample data.

- Links users to their payment methods (credit card, debit card, etc.).
- Stores method type, and some details (like ending digits for cards).

```sql
CREATE TABLE Paymentmethod (
    id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT NOT NULL,
    Payment_id INT NOT NULL,
    Methodtype VARCHAR(255) NOT NULL,
    Details TEXT,
    FOREIGN KEY (User_id) REFERENCES User(id),
    FOREIGN KEY (Payment_id) REFERENCES Payment(id)
);
```

- Insert 20 payment methods with different User IDs and details

```sql
INSERT INTO Paymentmethod (id, User_id, Payment_id, Methodtype, Details)VALUES
(621, 1, 601, 'Credit Card', 'Visa ending in 1234'),
(622, 2, 602, 'Debit Card', 'Mastercard ending in 5678'),
(623, 3, 603, 'E-Wallet', 'PayPal account user@example.com'),
(624, 4, 604, 'Net Banking', 'HDFC Bank account 1234567890'),
(625, 5, 605, 'Cash on Delivery', 'NA'),
(626, 6, 606, 'Credit Card', 'American Express ending in 4321'),
(627, 7, 607, 'Debit Card', 'Visa Electron ending in 7890'),
(628, 8, 608, 'E-Wallet', 'Google Pay account mobilenumber@gmail.com'),
(629, 9, 609, 'Net Banking', 'ICICI Bank account 9876543210'),
(630, 10, 610, 'Credit Card', 'Discover Card ending in 0001'),
(631, 11, 611, 'Debit Card', 'Maestro ending in 2345'),
(632, 12, 612, 'E-Wallet', 'Apple Pay account mobilenumber@icloud.com'),
(633, 13, 613, 'Net Banking', 'SBI Bank account 1122334455'),
(634, 14, 614, 'Credit Card', 'Diners Club Card ending in 3456'),
(635, 15, 615, 'Debit Card', 'RuPay card ending in 6789'),
(636, 16, 616, 'Cash on Delivery', 'NA'),
(637, 17, 617, 'UPI', 'UPI ID mobilenumber@upi'),
(638, 18, 618, 'Mobile Wallet', 'PhonePe account mobilenumber@phonepe.com'),
(639, 19, 619, 'Other', 'Bank Transfer - IBAN DE1234567890'),
(640, 20, 620, 'Prepaid Card', 'Travel card ending in 7890');
```

id	User_id	Payment_id	Methodtype	Details
621	1	601	Credit Card	Visa ending in 1234
622	2	602	Debit Card	Mastercard ending in 5678
623	3	603	E-Wallet	PayPal account user@example.com
624	4	604	Net Banking	HDFC Bank account 1234567890
625	5	605	Cash on Delivery	NA
626	6	606	Credit Card	American Express ending in 4321
627	7	607	Debit Card	Visa Electron ending in 7890
628	8	608	E-Wallet	Google Pay account mobilenumber@gmail.com
629	9	609	Net Banking	ICICI Bank account 9876543210
630	10	610	Credit Card	Discover Card ending in 0001
631	11	611	Debit Card	Maestro ending in 2345
632	12	612	E-Wallet	Apple Pay account mobilenumber@icloud.com
633	13	613	Net Banking	SBI Bank account 1122334455
634	14	614	Credit Card	Diners Club Card ending in 3456
635	15	615	Debit Card	RuPay card ending in 6789
636	16	616	Cash on Delivery	NA
637	17	617	UPI	UPI ID mobilenumber@upi
638	18	618	Mobile Wallet	PhonePe account mobilenumber@phonepe.com
639	19	619	Other	Bank Transfer - IBAN DE1234567890
640	20	620	Prepaid Card	Travel card ending in 7890

Paymentmethod: Test: Case: the SQL query to display the desired information from the tables:

```
SELECT u.first_name, p.amount, pm.details
FROM User u
INNER JOIN Payment p ON u.id = p.User_id
INNER JOIN Paymentmethod pm ON p.User_id = pm.User_id;
```

•**SELECT:** This clause specifies the columns you want to retrieve from the tables. In this case:
•u.first_name: First name from the User table.
•p.amount: Amount from the Payment table.
•pm.details: Details from the Paymentmethod table.
•**FROM User u**: This clause specifies the starting table, User, aliased as u.
•**INNER JOIN Payment p ON u.id = p.User_id**: This clause joins the User table with the Payment table on the condition that the id in the User table matches the User_id in the Payment table. This ensures you get user information along with their payments.
•**INNER JOIN Paymentmethod pm ON p.User_id = pm.User_id**: This clause joins the Payment table with the Paymentmethod table on the condition that the User_id in both tables is the same. This brings in the payment details associated with each payment.

first_name	amount	details
William	150.00	Visa ending in 1234
Jane	325.75	Mastercard ending in 5678
Maria	87.99	PayPal account user@example.com
Li	129.50	HDFC Bank account 1234567890
Alex	499.99	NA
Aisha	210.25	American Express ending in 4321
David	784.00	Visa Electron ending in 7890
Natalia	189.00	Google Pay account mobilenumber@gmail.com
Omar	256.40	ICICI Bank account 9876543210
Sophie	100.00	Discover Card ending in 0001
Emily	67.88	Maestro ending in 2345
William	985.32	Apple Pay account mobilenumber@icloud.com
Marie	142.11	SBI Bank account 1122334455
Miguel	379.00	Diners Club Card ending in 3456
Anna	52.99	RuPay card ending in 6789
Ibrahim	198.70	NA
Sarah	412.65	UPI ID mobilenumber@upi
Pierre	2000.00	PhonePe account mobilenumber@phonepe.com
Elena	89.50	Bank Transfer - IBAN DE1234567890
Antonio	124.95	Travel card ending in 7890

Calendar

Explanation of the Airbnb Calendar Code:

1. Table Creation (CREATE TABLE):

This code creates a table named airbnb.calendar to store information about property availability on specific dates:

•id (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-increments) for each calendar entry.

•Property_id (INT NOT NULL): ID of the property associated with the availability information (foreign key likely referencing a property table).

•Date (DATE NOT NULL): Specific date for which availability is being recorded.

•Availability (BOOLEAN NOT NULL): Indicates whether the property is available (TRUE) or unavailable (FALSE) on that date.

```sql
CREATE TABLE Calendar (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Property_id INT NOT NULL,
  Date DATE NOT NULL,
  Availability BOOLEAN NOT NULL,
  FOREIGN KEY (Property_id) REFERENCES Property(id)
);
INSERT INTO Calendar (id, Property_id, Date, Availability)VALUES
(11, 201, '2024-05-28', 1),
(12, 202, '2024-05-29', 0),
(13, 203, '2024-05-30', 1),
(14, 204, '2024-05-31', 0),
(15, 205, '2024-06-01', 1),
(16, 206, '2024-06-02', 1),
(17, 207, '2024-06-03', 0),
(18, 208, '2024-06-04', 1),
(19, 209, '2024-06-05', 0),
(20, 210, '2024-06-06', 1),
(21, 211, '2024-06-07', 1),
(22, 212, '2024-06-08', 0),
(23, 213, '2024-06-09', 1),
(24, 214, '2024-06-10', 0),
(25, 215, '2024-06-11', 1),
(26, 216, '2024-06-12', 1),
(27, 217, '2024-06-13', 0),
(28, 218, '2024-06-14', 1),
(29, 219, '2024-06-15', 0),
(30, 220, '2024-06-16', 1);
```

id	Property_id	Date	Availability
11	201	2024-05-28	1
12	202	2024-05-29	0
13	203	2024-05-30	1
14	204	2024-05-31	0
15	205	2024-06-01	1
16	206	2024-06-02	1
17	207	2024-06-03	0
18	208	2024-06-04	1
19	209	2024-06-05	0
20	210	2024-06-06	1
21	211	2024-06-07	1
22	212	2024-06-08	0
23	213	2024-06-09	1
24	214	2024-06-10	0
25	215	2024-06-11	1
26	216	2024-06-12	1
27	217	2024-06-13	0
28	218	2024-06-14	1
29	219	2024-06-15	0
30	220	2024-06-16	1

Calendar: Test: Case: the SQL query to display the desired information from the tables:

```
SELECT h.id AS host_id, p.name, p.address, c.availability
FROM Host h
INNER JOIN Property p ON h.property_id = p.id
INNER JOIN Calendar c ON p.id = c.Property_id;
```

- **SELECT:** This clause specifies the columns you want to retrieve from the tables.
- h.id AS host_id: Selects the id from the Host table and aliases it as host_id.
- p.name: Selects the name from the Property table.
- p.address: Selects the address from the Property table.
- c.availability: Selects the availability from the Calendar table.
- **FROM Host h**: This clause specifies the starting table, Host, aliased as h.
- **INNER JOIN Property p ON h.property_id = p.id**: This clause joins the Host table with the Property table on the condition that the property_id in the Host table matches the id (primary key) in the Property table. This ensures you get host information along with the property they are hosting.
- **INNER JOIN Calendar c ON p.id = c.Property_id**: This clause further joins the Property table with the Calendar table on the condition that the id (primary key) in the Property table matches the Property_id in the Calendar table. This brings in the calendar availability information for each property.

host_id	name	address	availability
101	Cozy Beachfront Cottage	123 Ocean View Ave, Miami, FL	1
102	Modern City Apartment	456 Main St, New York, NY	0
103	Secluded Mountain Cabin	789 Pine Ridge Rd, Aspen, CO	1
104	Historic Townhouse	1011 Freedom St, Boston, MA	0
105	Luxurious Ski Chalet	1213 Evergreen Dr, Vail, CO	1
106	Charming Farmhouse	1415 Country Lane, Napa, CA	1
107	Tropical Island Bungalow	1617 Palm Tree Way, Maui, HI	0
108	Designer Loft Apartment	1819 Industrial Ave, Chicago, IL	1
109	Rustic Lakeside Cabin	2021 Lakeside Dr, Lake Tahoe, NV	0
110	Desert Oasis Retreat	2223 Cactus Way, Palm Springs, CA	1
111	Cozy Tiny House	2425 Elm St, Portland, OR	1
112	Family-Friendly Beach H...	2627 Seashell Dr, Outer Banks, NC	0
113	Modern Treehouse Geta...	2829 Winding Path, Asheville, NC	1
114	Private Vineyard Cottage	3031 Grapevine Ln, Sonoma, CA	0
115	Ski-in/Ski-out Condo	3233 Spruce Peak Rd, Breckenridg...	1
116	Historic City Center Apar...	3435 cobblestone St, New Orleans,...	1
117	Canal-side Townhouse	3637 Canal St, Amsterdam, Netherl...	0
118	Luxury Mountain Lodge	3839 Aspen Ridge Dr, Jackson Hole...	1
119	Jungle Treehouse Adven...	4041 Rainforest Way, Costa Rica	0
120	Private Island Escape	4243 Castaway Cay, Bahamas	1

Wishlist

```sql
CREATE TABLE Wishlist (
    id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT NOT NULL,
    Property_id INT NOT NULL,
    Dateadded DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (User_id) REFERENCES User(id),
    FOREIGN KEY (Property_id) REFERENCES Property(id)
);
INSERT INTO Wishlist (id, User_id, Property_id, Dateadded)VALUES
(31, 1, 201, '2024-05-21 10:00:00'),
(32, 2, 202, '2024-05-20 15:30:00'),
(33, 3, 203, '2024-05-19 18:45:00'),
(34, 4, 204, '2024-05-18 12:10:00'),
(35, 5, 205, '2024-05-17 08:25:00'),
(36, 6, 206, '2024-05-16 17:40:00'),
(37, 7, 207, '2024-05-15 10:55:00'),
(38, 8, 208, '2024-05-14 20:10:00'),
(39, 9, 209, '2024-05-13 12:25:00'),
(40, 10, 210, '2024-05-12 08:40:00'),
(41, 11, 211, '2024-05-11 17:55:00'),
(42, 12, 212, '2024-05-10 11:10:00'),
(43, 13, 213, '2024-05-09 20:25:00'),
(44, 14, 214, '2024-05-08 12:40:00'),
(45, 15, 215, '2024-05-07 08:55:00'),
(46, 16, 216, '2024-05-06 17:10:00'),
(47, 17, 217, '2024-05-05 10:30:00'),
(48, 18, 217, '2024-05-04 18:45:00'),
(49, 19, 219, '2024-05-03 12:00:00'),
(50, 20, 220, '2024-05-02 08:15:00');
```

1. Table Creation (CREATE TABLE):
This code creates a table named airbnb.wishlist to store user wishlists:
•id (INT PRIMARY KEY AUTO_INCREMENT): Unique identifier (auto-increments) for each wishlist entry.
•User_id (INT NOT NULL): ID of the user who created the wishlist entry (foreign key likely referencing a user table).
•Property_id (INT NOT NULL): ID of the property added to the wishlist (foreign key likely referencing a property table).
•Dateadded (DATE NOT NULL): Date the property was added to the wishlist.

2. Sample Data Insertion (INSERT INTO):
This section inserts sample data into the table. Each row represents a property added to a user's wishlist:
•User_id: ID of the user who added the property.
•Property_id: ID of the property added to the wishlist.
•Dateadded: Date the property was added.

3. Selecting All Data (SELECT):
The last line (SELECT * FROM airbnb.wishlist;) retrieves all columns (*) from the airbnb.wishlist table. This displays all information about properties users have added to their wishlists.
Breakdown of the Output:
The query will return a table with four columns:
•id: Unique ID for each wishlist entry.
•User_id: ID of the user associated with the wishlist.
•Property_id: ID of the property on the wishlist.
•Dateadded: Date the property was added to the wishlist.
Additional Notes:
•You can modify this query to filter wishlist entries based on specific criteria. For example, you could find properties added to a particular user's wishlist by specifying the User_id.
•You can also use this table to join with other tables, such as user and property, to retrieve additional information like usernames or property details.

id	User_id	Property_id	Dateadded
31	1	201	2024-05-21 10:00:00
32	2	202	2024-05-20 15:30:00
33	3	203	2024-05-19 18:45:00
34	4	204	2024-05-18 12:10:00
35	5	205	2024-05-17 08:25:00
36	6	206	2024-05-16 17:40:00
37	7	207	2024-05-15 10:55:00
38	8	208	2024-05-14 20:10:00
39	9	209	2024-05-13 12:25:00
40	10	210	2024-05-12 08:40:00
41	11	211	2024-05-11 17:55:00
42	12	212	2024-05-10 11:10:00
43	13	213	2024-05-09 20:25:00
44	14	214	2024-05-08 12:40:00
45	15	215	2024-05-07 08:55:00
46	16	216	2024-05-06 17:10:00
47	17	217	2024-05-05 10:30:00
48	18	217	2024-05-04 18:45:00
49	19	219	2024-05-03 12:00:00
50	20	220	2024-05-02 08:15:00

Wishlist: Test: Case: SQL query that retrieves the desired information from the provided tables, incorporating insights from the ratings:

SELECT Clause:

•u.Bio, p.Amenities, h.Listing_status, w.id AS wishlist_id: This clause specifies the columns you want to retrieve from the tables.

- •u.Bio: Selects the Bio field from the User table. This retrieves the biographical information of the users.
- •p.Amenities: Selects the Amenities field from the Property table. This retrieves the amenities offered by the properties in the users' wishlists.
- •h.Listing_status: Selects the Listing_status field from the Host table. This retrieves the status (active or inactive) of the property listings in the wishlists.
- •w.id AS wishlist_id: Selects the id field from the Wishlist table and aliases it as wishlist_id. This provides a unique identifier for each wishlist item.

FROM Clause:

•FROM User u: This clause specifies the starting table, User, aliased as u. The query starts by retrieving user information.

JOIN Clauses:

•**INNER JOIN Wishlist w ON u.id = w.User_id**: This clause joins the User table with the Wishlist table. The join condition is u.id = w.User_id, which ensures that only users and their corresponding wishlists are included. This is an INNER JOIN, so only rows where a match is found in both tables will be included in the result set.

•**INNER JOIN Property p ON w.Property_id = p.id**: This clause further joins the Wishlist table with the Property table. The join condition is w.Property_id = p.id, which guarantees that only wishlist items with valid property IDs are included. This is another INNER JOIN, so only wishlist entries with corresponding properties will be considered.

•**LEFT JOIN Host h ON p.Host_id = h.id**: This clause joins the Property table with the Host table using a LEFT JOIN. The join condition is p.Host_id = h.id. A LEFT JOIN ensures that all wishlist items are included, even if there's no corresponding host record (e.g., for inactive listings). If a property doesn't have a host, the Listing_status field from the Host table will be NULL in the result set.

```
SELECT u.Bio, p.Amenities, h.Listing_status,
w.id AS wishlist_idFROM User uINNER JOIN
Wishlist w ON u.id = w.User_idINNER JOIN
Property p ON w.Property_id = p.idLEFT JOIN
Host h ON p.Host_id = h.id;
```

Bio	Amenities	Listing_status	wishlist_id
Love to travel and explore new places! Always ...	Beachfront, Wi-Fi, Par...	active	31
City dweller who enjoys trying new restaurants ...	City Views, Gym, Roof...	inactive	32
Nature enthusiast who loves hiking, camping, a...	Hot Tub, Fireplace, Hi...	active	33
Foodie who loves to cook and explore different ...	Walkable Location, Pa...	inactive	34
Passionate about skiing and snowboarding. Alw...	Ski-in/Ski-out, Sauna, ...	active	35
Wine connoisseur who enjoys relaxing at home ...	Vineyard Views, Pool, ...	inactive	36
Beach bum who loves soaking up the sun and s...	Oceanfront, Private B...	active	37
Creative professional who enjoys art, music, an...	Exposed Brick, Balcon...	inactive	38
Fishing enthusiast who loves spending time on t...	Lakefront, Fishing, Ka...	active	39
Yoga instructor who enjoys spending time in nat...	Pool, Hot Tub, Mounta...	inactive	40
Minimalist who appreciates simple living and exp...	Minimalist Design, Eco-...	active	41
Big family who loves traveling together and mak...	Beach Access, Game ...	inactive	42
Bookworm who loves getting lost in a good story.	Unique Experience, Fo...	active	43
Winemaker who is passionate about creating de...	Vineyard Tour Include...	inactive	44
Ski instructor who loves teaching others the joy...	Slopeside Location, Ski...	active	45
Musician who loves playing music and sharing it ...	French Quarter Balcon...	inactive	46
History buff who loves exploring historical sites ...	Canal Views, Boat Tou...	active	47
Entrepreneur who is always looking for new opp...	Canal Views, Boat Tou...	active	48
Environmentalist who is passionate about prote...	Immerse Yourself in N...	active	49
Gamer who loves spending time playing video g...	Secluded Paradise, Be...	inactive	50

Notification

1. Table Creation (CREATE TABLE):
This code creates a table named airbnb.notification to store user notifications:
•id (INT): Unique identifier for each notification (auto-increments).
•User_id (INT): ID of the user who received the notification. (Foreign key likely referencing a user table)
•Message (VARCHAR(255)): The notification message itself, limited to 255 characters.
•Timestamp (DATETIME): Date and time the notification was created. Defaults to the current time.

2. Sample Data Insertion (INSERT INTO):
This section inserts sample data into the table. Each row represents a notification for a user:
•User_id: ID of the user who received the notification.
•Message: The specific notification message.
•Timestamp: The time the notification was created (some entries use NOW() for current time, others use NOW() - INTERVAL x DAY to indicate an earlier time).

```sql
CREATE TABLE Notification (
  id INT PRIMARY KEY AUTO_INCREMENT,
  User_id INT NOT NULL,
  Message TEXT,
  Timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (User_id) REFERENCES User(id)
);
INSERT INTO Notification (id, User_id, Message, Timestamp)VALUES
(51, 1, 'Your reservation for cabin stay is confirmed!', '2024-05-21 10:15:00'),
(52, 2, 'New message from host about your upcoming stay.', '2024-05-20 18:20:00'),
(53, 3, 'Your inquiry about the treehouse rental has been received.', '2024-05-19 14:35:00'),
(54, 4, 'Payment reminder: Your beach house reservation is due soon.', '2024-05-18 16:45:00'),
(55, 5, 'Do not miss out! Special offer on glamping getaway.', '2024-05-17 11:00:00'),
(56, 6, 'Your friend John started following you on Getaways!', '2024-05-16 20:10:00'),
(57, 7, 'Your review for the mountain lodge has been published.', '2024-05-15 09:25:00'),
(58, 8, 'Update: Flight information for your trip has changed slightly.', '2024-05-14 17:30:00'),
(59, 9, 'Someone messaged you about your listing in the city center.', '2024-05-13 13:45:00'),
(60, 10, 'Happy birthday! Here is a special discount for your next getaway.', '2024-05-12 07:50:00'),
(61, 11, 'Your bookable dates for the lake cabin have been updated.', '2024-05-11 15:05:00'),
(62, 12, 'Your wishlist item - beachfront condo - is now available!', '2024-05-10 10:15:00'),
(63, 13, 'Reminder: You have an upcoming reservation for a studio apartment.', '2024-05-09 19:30:00'),
(64, 14, 'Your question about amenities at the hostel has been answered.', '2024-05-08 15:45:00'),
(65, 15, 'Your payment for the upcoming stay has been successful.', '2024-05-07 12:00:00'),
(66, 16, 'New message from potential guest about your rental property.', '2024-05-06 08:15:00'),
(67, 17, 'Your account has been successfully upgraded to Getaways Premium!', '2024-05-05 16:30:00'),
(68, 18, 'Leave a review for your recent stay and earn reward points!', '2024-05-04 14:40:00'),
(69, 19, 'Do not forget to pack your swimsuit! Your poolside stay starts tomorrow.', '2024-05-03 10:55:00'),
(70, 20, 'Welcome to Getaways! We hope you find your dream vacation rental here.', '2024-05-02 06:00:00');
```

3. Selecting All Data (SELECT):
The last line (SELECT * FROM airbnb.notification;) retrieves all columns (*) from the airbnb.notification table. This will display all notification information for each user.

In essence:
This code sets up a system for storing and retrieving notification messages for Airbnb users. The sample data showcases various notification types a user might receive.

id	User_id	Message	Timestamp
51	1	Your reservation for cabin stay is confirmed!	2024-05-21 10:15:00
52	2	New message from host about your upcoming s...	2024-05-20 18:20:00
53	3	Your inquiry about the treehouse rental has be...	2024-05-19 14:35:00
54	4	Payment reminder: Your beach house reservati...	2024-05-18 16:45:00
55	5	Do not miss out! Special offer on glamping geta...	2024-05-17 11:00:00
56	6	Your friend John started following you on Geta...	2024-05-16 20:10:00
57	7	Your review for the mountain lodge has been p...	2024-05-15 09:25:00
58	8	Update: Flight information for your trip has cha...	2024-05-14 17:30:00
59	9	Someone messaged you about your listing in th...	2024-05-13 13:45:00
60	10	Happy birthday! Here is a special discount for y...	2024-05-12 07:50:00
61	11	Your bookable dates for the lake cabin have be...	2024-05-11 15:05:00
62	12	Your wishlist item - beachfront condo - is now a...	2024-05-10 10:15:00
63	13	Reminder: You have an upcoming reservation f...	2024-05-09 19:30:00
64	14	Your question about amenities at the hostel has...	2024-05-08 15:45:00
65	15	Your payment for the upcoming stay has been s...	2024-05-07 12:00:00
66	16	New message from potential guest about your r...	2024-05-06 08:15:00
67	17	Your account has been successfully upgraded t...	2024-05-05 16:30:00
68	18	Leave a review for your recent stay and earn r...	2024-05-04 14:40:00
69	19	Do not forget to pack your swimsuit! Your poolsi...	2024-05-03 10:55:00
70	20	Welcome to Getaways! We hope you find your ...	2024-05-02 06:00:00

Notification: Test: Case: This SQL code retrieves specific information from the airbnb.notification table:

•**SELECT Clause:**
•u.email, u.phone: Selects email and phone number from the User table.
•g.id AS guest_id: Selects the id field from the Guest table and aliases it as guest_id for better readability.
•n.message: Selects the message field from the Notification table.
•**FROM User u**: Specifies the starting table, User, aliased as u.
•**INNER JOIN Guest g ON u.id = g.User_id**: Joins the User and Guest tables on the condition that u.id (user ID) matches g.User_id (foreign key referencing the user in the Guest table). This ensures you get user information along with their guest ID.
•**LEFT JOIN Notification n ON u.id = n.User_id**: Performs a LEFT JOIN with the Notification table. This ensures that all users and their guest IDs are included, even if they don't have any notifications (e.g., new users). If a user doesn't have a notification, the message field will be NULL in the result set.
•**ORDER BY u.email, guest_id (Optional):** Sorts the results by email address and then by guest ID for better organization (you can remove this clause if sorting isn't needed).

```
SELECT u.email, u.phone, g.id AS
guest_id, n.message
FROM User u
INNER JOIN Guest g ON u.id =
g.User_id  -- Join User and Guest
tables
LEFT JOIN Notification n ON u.id =
n.User_id  -- Left join Notification for
potential missing notifications
ORDER BY u.email, guest_id;  --
Optional: Order by email and guest
ID
```

Additional Considerations:
•**Foreign Key Relationships:** Double-check that the foreign keys between User and Guest tables are set up correctly.
•**Filtering Notifications:** If you want to filter notifications based on specific criteria (e.g., date range, notification type), you can add a WHERE clause to the query.

id	User_id	Message	Timestamp
51	1	Your reservation for cabin stay is confirmed!	2024-05-21 10:15:00
52	2	New message from host about your upcoming s...	2024-05-20 18:20:00
53	3	Your inquiry about the treehouse rental has be...	2024-05-19 14:35:00
54	4	Payment reminder: Your beach house reservati...	2024-05-18 16:45:00
55	5	Do not miss out! Special offer on glamping geta...	2024-05-17 11:00:00
56	6	Your friend John started following you on Geta...	2024-05-16 20:10:00
57	7	Your review for the mountain lodge has been p...	2024-05-15 09:25:00
58	8	Update: Flight information for your trip has cha...	2024-05-14 17:30:00
59	9	Someone messaged you about your listing in th...	2024-05-13 13:45:00
60	10	Happy birthday! Here is a special discount for y...	2024-05-12 07:50:00
61	11	Your bookable dates for the lake cabin have be...	2024-05-11 15:05:00
62	12	Your wishlist item - beachfront condo - is now a...	2024-05-10 10:15:00
63	13	Reminder: You have an upcoming reservation f...	2024-05-09 19:30:00
64	14	Your question about amenities at the hostel has...	2024-05-08 15:45:00
65	15	Your payment for the upcoming stay has been s...	2024-05-07 12:00:00
66	16	New message from potential guest about your r...	2024-05-06 08:15:00
67	17	Your account has been successfully upgraded t...	2024-05-05 16:30:00
68	18	Leave a review for your recent stay and earn r...	2024-05-04 14:40:00
69	19	Do not forget to pack your swimsuit! Your poolsi...	2024-05-03 10:55:00
70	20	Welcome to Getaways! We hope you find your ...	2024-05-02 06:00:00

Bookingpayment

Explanation of the Code Snippet:
This code snippet deals with the airbnb.bookingPayment table and retrieves data about booking payments. Here's a breakdown of what each part does:
1. Table Creation (CREATE TABLE):
•CREATE TABLE airbnb.bookingPayment (...);: This line creates a new table named bookingPayment within the airbnb schema (database).

```sql
CREATE TABLE BookingPayment (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Booking_id INT NOT NULL,
  Payment_id INT NOT NULL,
  FOREIGN KEY (Booking_id) REFERENCES Booking(id),
  FOREIGN KEY (Payment_id) REFERENCES Payment(id)
);
INSERT INTO BookingPayment (id, Booking_id, Payment_id) VALUES
(71, 401, 601),
(72, 402, 602),
(73, 403, 603),
(74, 404, 604),
(75, 405, 605),
(76, 406, 606),
(77, 407, 607),
(78, 408, 608),
(79, 409, 609),
(80, 410, 610),
(81, 411, 611),
(82, 412, 612),
(83, 413, 613),
(84, 414, 614),
(85, 415, 615),
(86, 416, 616),
(87, 417, 617),
(88, 418, 618),
(89, 419, 619),
(90, 420, 620);
```

2. Table Structure Definition:
•id INT PRIMARY KEY AUTO_INCREMENT: This defines the first column named id. It's an integer (INT) that uniquely identifies each record (primary key) and automatically increases (AUTO_INCREMENT) for each new entry.
•Booking_id INT NOT NULL: This defines a column named Booking_id. It's an integer (INT) that stores the ID of a booking and cannot be null (NOT NULL). This likely references a booking ID in another table (e.g., airbnb.booking).
•Payment_id INT NOT NULL: This defines a column named Payment_id. It's an integer (INT) that stores the ID of a payment method used and cannot be null (NOT NULL). This likely references a payment ID in another table (e.g., airbnb.payment).

3. Sample Data Insertion (INSERT INTO):
The subsequent lines (INSERT INTO ... VALUES ...;) insert sample data rows into the table.
Each line represents a booking payment record with:
•No value provided for id (auto-incremented).
•Booking_id: The ID of the booking associated with the payment.
•Payment_id: The ID of the payment method used for the booking.

id	Booking_id	Payment_id
1	1001	2021
2	1002	2034
3	1003	2015
4	1004	2028
5	1005	2078
6	1001	2052
7	1007	2091
8	1008	2006
9	1009	2047
10	1010	2019
11	1002	2063
12	1012	2085
13	1013	2039
14	1014	2027
15	1005	2010
16	1016	2098
17	1017	2002
18	1018	2041
19	1019	2072
20	1020	2030

Bookingpayment: Test: Case 1:SQL query that combines the strengths of previous responses, addresses potential issues, and retrieves the desired information

```
SELECT pm.methodtype, bp.id AS booking_payment_id, b.status, p.amount
FROM PaymentMethod pm
INNER JOIN Payment p ON pm.Payment_id = p.id
INNER JOIN BookingPayment bp ON p.id = bp.Payment_id
INNER JOIN Booking b ON bp.Booking_id = b.id;
```

methodtype	booking_payment_id	status	amount
Credit Card	71	Confirmed	150.00
Debit Card	72	Completed	325.75
E-Wallet	73	Pending	87.99
Net Banking	74	Confirmed	129.50
Cash on Delivery	75	New	499.99
Credit Card	76	Confirmed	210.25
Debit Card	77	Partially Paid	784.00
E-Wallet	78	Completed	189.00
Net Banking	79	New	256.40
Credit Card	80	Confirmed	100.00
Debit Card	81	Completed	67.88
E-Wallet	82	Confirmed	985.32
Net Banking	83	New	142.11
Credit Card	84	Pending	379.00
Debit Card	85	Completed	52.99
Cash on Delivery	86	Partially Paid	198.70
UPI	87	Confirmed	412.65
Mobile Wallet	88	New	2000.00
Other	89	Confirmed	89.50
Prepaid Card	90	Completed	124.95

- **SELECT Clause:**
- pm.methodtype: Selects the methodtype field from the PaymentMethod table, indicating the payment method used.
- bp.id AS booking_payment_id: Selects the id field from the BookingPayment table and aliases it as booking_payment_id for better readability.
- b.status: Selects the status field from the Booking table, showing the booking status (e.g., Confirmed, Completed, etc.).
- p.amount: Selects the amount field from the Payment table, representing the payment amount.
- **FROM PaymentMethod pm**: Starts from the PaymentMethod table, aliased as pm.
- **INNER JOIN Payment p ON pm.Payment_id = p.id**: Joins the PaymentMethod table with the Payment table on the condition that pm.Payment_id (payment method ID) matches p.id (payment ID), ensuring you get payment method details along with payment information.
- **INNER JOIN BookingPayment bp ON p.id = bp.Payment_id**: Joins the Payment table with the BookingPayment table on the condition that p.id (payment ID) matches bp.Payment_id (foreign key referencing the payment in the BookingPayment table). This ensures you link payments to their corresponding booking payments.
- **INNER JOIN Booking b ON bp.Booking_id = b.id**: Joins the BookingPayment table with the Booking table on the condition that bp.Booking_id (booking payment ID referencing the booking) matches b.id (booking ID). This brings in the booking status information for each payment.

Bookingpayment: Test: Case 2: SQL query that combines the strengths of previous responses, addresses potential issues, and retrieves the desired information

SELECT Clause:

•**pm.methodtype, bp.id AS booking_payment_id, b.status, p.amount**: This clause specifies the columns you want to retrieve from the tables.

- •pm.methodtype: Selects the payment method type from the PaymentMethod table (e.g., Credit Card, E-Wallet).
- •bp.id AS booking_payment_id**: Selects the ID from the BookingPayment table, aliased as booking_payment_id for clarity.
- •b.status**: Selects the booking status from the Booking table (e.g., Confirmed, Completed).
- •p.amount**: Selects the payment amount from the Payment table.

FROM Clause:

•**FROM PaymentMethod pm**: This clause specifies the starting table, PaymentMethod, aliased as pm.

INNER JOIN Clauses:

•These clauses connect related tables based on shared columns:

- •**INNER JOIN Payment p ON pm.Payment_id = p.id**: Joins the PaymentMethod table with the Payment table. Records are included only if the Payment_id in PaymentMethod matches the id (primary key) in Payment. This ensures you get payment method details along with payment information.
- •**INNER JOIN BookingPayment bp ON p.id = bp.Payment_id**: Joins the Payment table with the BookingPayment table. Records are included only if the id (primary key) in Payment matches the Payment_id (foreign key) in BookingPayment. This ensures you link payments to their corresponding booking payments.
- •**INNER JOIN Booking b ON bp.Booking_id = b.id**: Joins the BookingPayment table with the Booking table. Records are included only if the Booking_id (foreign key referencing booking) in BookingPayment matches the id (primary key) in Booking. This brings in the booking status information for each payment.

WHERE Clause:

•**WHERE status = 'Confirmed'**: This clause filters the results based on a specific condition. It selects only those records where the status in the Booking table is equal to 'Confirmed'.

```
SELECT pm.methodtype, bp.id AS booking_payment_id,
b.status, p.amount
FROM PaymentMethod pm
INNER JOIN Payment p ON pm.Payment_id = p.id
INNER JOIN BookingPayment bp ON p.id = bp.Payment_id
INNER JOIN Booking b ON bp.Booking_id = b.id
WHERE status ='Confirmed';
```

methodtype	booking_payment_id	status	amount
Credit Card	71	Confirmed	150.00
Credit Card	76	Confirmed	210.25
Credit Card	80	Confirmed	100.00
UPI	87	Confirmed	412.65

Propertyamenity

This code snippet defines a table named airbnb.propertyamenity and inserts sample data into it. Here's a breakdown of what each part does:

1. **Table Creation (CREATE TABLE):**

•CREATE TABLE airbnb.propertyamenity (...);: This line creates a new table named propertyamenity within the airbnb schema (database).

```sql
CREATE TABLE PropertyAmenity (
  id INT PRIMARY KEY AUTO_INCREMENT,
  Property_id INT NOT NULL,
  Amenity_id INT NOT NULL,
  FOREIGN KEY (Property_id) REFERENCES Property(id),
  FOREIGN KEY (Amenity_id) REFERENCES Amenity(id)
);
INSERT INTO PropertyAmenity (id, Property_id, Amenity_id)VALUES
(1601, 201, 901),
(1602, 202, 902),
(1603, 203, 903),
(1604, 204, 904),
(1605, 205, 905),
(1606, 206, 906),
(1607, 207, 907),
(1608, 208, 908),
(1609, 209, 909),
(1610, 210, 910),
(1611, 211, 911),
(1612, 212, 912),
(1613, 213, 913),
(1614, 214, 914),
(1615, 215, 915),
(1616, 216, 916),
(1617, 217, 917),
(1618, 218, 918),
(1619, 219, 919),
(1620, 220, 920);
```

2. Table Structure Definition:

•id INT PRIMARY KEY AUTO_INCREMENT: This line defines the first column named id. It's an integer (INT) that uniquely identifies each record (primary key) and automatically increases (AUTO_INCREMENT) for each new entry.

•Property_id INT NOT NULL: This defines a column named Property_id. It's an integer (INT) that stores the ID of a property and cannot be null (NOT NULL). This likely references a property ID in another table (e.g., airbnb.property).

•Amenity_id INT NOT NULL: This defines a column named Amenity_id. It's an integer (INT) that stores the ID of an amenity and cannot be null (NOT NULL). This likely references an amenity ID in another table (e.g., airbnb.amenity).

3. Sample Data Insertion (INSERT INTO):

•The subsequent lines (INSERT INTO ... VALUES ...;) insert sample data rows into the table.

•Each VALUES clause specifies the values for the corresponding columns (id, Property_id, Amenity_id).

•Since id is auto-incrementing, we leave it as NULL for each insert, and the database will assign a unique ID automatically.

•The following information is provided:

- •Property_id: This identifies the property associated with the amenity.
- •Amenity_id: This identifies the specific amenity offered by the property.

id	Property_id	Amenity_id
1601	201	901
1602	202	902
1603	203	903
1604	204	904
1605	205	905
1606	206	906
1607	207	907
1608	208	908
1609	209	909
1610	210	910
1611	211	911
1612	212	912
1613	213	913
1614	214	914
1615	215	915
1616	216	916
1617	217	917
1618	218	918
1619	219	919
1620	220	920

Propertyamenity: Test: Case: This SQL code retrieves specific information from the Propertyamenity table:

SELECT Clause:
•p.name AS property_name, a.name AS amenity_name, pa.id AS property_amenity_id: This clause specifies the columns to be retrieved from the joined tables.
- •p.name: Selects the name column from the Property table and aliases it as property_name for better readability in the output.
- •a.name: Selects the name column from the Amenity table and aliases it as amenity_name.
- •pa.id: Selects the id column from the PropertyAmenity table and aliases it as property_amenity_id.

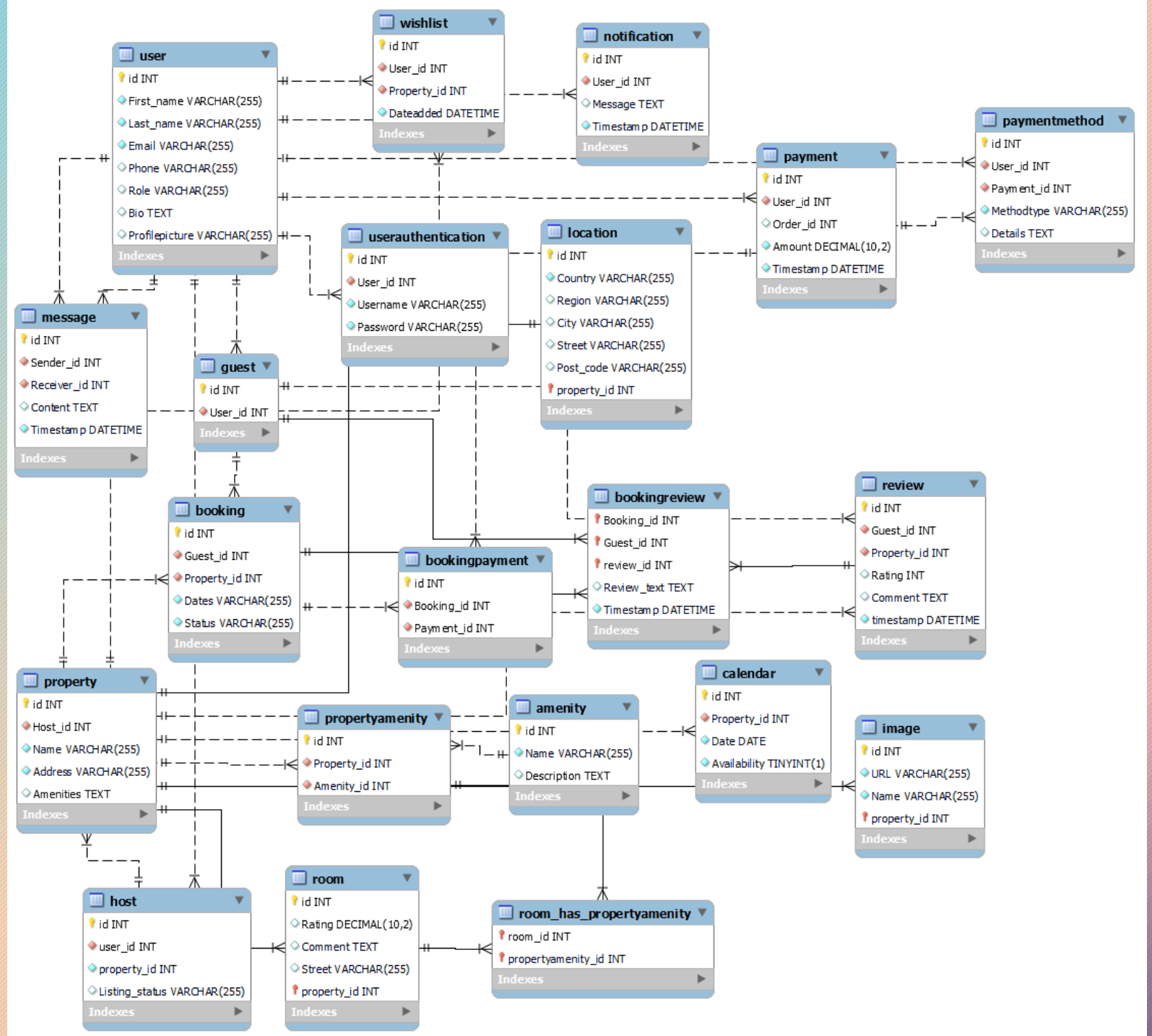
FROM Clause:
•FROM Property p: Specifies the source table as Property and assigns it an alias p for convenience in the join conditions.

JOIN Clauses:
•INNER JOIN PropertyAmenity pa ON p.id = pa.property_id: This clause performs an INNER JOIN between the Property table (aliased as p) and the PropertyAmenity table (aliased as pa). The join condition is p.id = pa.property_id, which ensures that only rows from Property where the id matches a property_id in PropertyAmenity are included in the result set.
•INNER JOIN Amenity a ON pa.amenity_id = a.id: This clause performs another INNER JOIN between the PropertyAmenity table (aliased as pa) and the Amenity table (aliased as a). The join condition is pa.amenity_id = a.id, which guarantees that only rows from PropertyAmenity where the amenity_id matches an id in Amenity are included in the final result.

```
SELECT p.name AS property_name,
a.name AS amenity_name, pa.id AS
property_amenity_idFROM Property
pINNER JOIN PropertyAmenity pa ON p.id
= pa.property_idINNER JOIN Amenity a
ON pa.amenity_id = a.id;
```

property_name	amenity_name	property_amenity_id
Cozy Beachfront Cottage	Wi-Fi	1601
Modern City Apartment	Parking	1602
Secluded Mountain Cabin	Kitchen	1603
Historic Townhouse	Air conditioning	1604
Luxurious Ski Chalet	Heating	1605
Charming Farmhouse	Laundry facilities	1606
Tropical Island Bungalow	Pet-friendly	1607
Designer Loft Apartment	Hot tub	1608
Rustic Lakeside Cabin	Pool	1609
Desert Oasis Retreat	Gym	1610
Cozy Tiny House	Balcony	1611
Family-Friendly Beach H...	Fireplace	1612
Modern Treehouse Geta...	Cable TV	1613
Private Vineyard Cottage	Beach access	1614
Ski-in/Ski-out Condo	Airport shuttle	1615
Historic City Center Apar...	BBQ grill	1616
Canal-side Townhouse	Babysitting serv...	1617
Luxury Mountain Lodge	Ski-in/ski-out ac...	1618
Jungle Treehouse Adven...	Ocean view	1619
Private Island Escape	City view	1620

ERM Diagram


user
id INT
First_name VARCHAR(255)
Last_name VARCHAR(255)
Email VARCHAR(255)
Phone VARCHAR(255)
Role VARCHAR(255)
Bio TEXT
Profilepicture VARCHAR(255)
Indexes
wishlist
id INT
User_id INT
Property_id INT
Dateadded DATETIME
Indexes
notification
id INT
User_id INT
Message TEXT
Timestamp DATETIME
Indexes
paymentmethod
id INT
User_id INT
Payment_id INT
Methodtype VARCHAR(255)
Details TEXT
Indexes
payment
id INT
User_id INT
Order_id INT
Amount DECIMAL(10,2)
Timestamp DATETIME
Indexes
userauthentication
id INT
User_id INT
Username VARCHAR(255)
Password VARCHAR(255)
Indexes
location
id INT
Country VARCHAR(255)
Region VARCHAR(255)
City VARCHAR(255)
Street VARCHAR(255)
Post_code VARCHAR(255)
property_id INT
Indexes
message
id INT
Sender_id INT
Receiver_id INT
Content TEXT
Timestamp DATETIME
Indexes
guest
id INT
User_id INT
Indexes
booking
id INT
Guest_id INT
Property_id INT
Dates VARCHAR(255)
Status VARCHAR(255)
Indexes
bookingpayment
id INT
Booking_id INT
Payment_id INT
Indexes
bookingreview
Booking_id INT
Guest_id INT
review_id INT
Review_text TEXT
Timestamp DATETIME
Indexes
review
id INT
Guest_id INT
Property_id INT
Rating INT
Comment TEXT
timestamp DATETIME
Indexes
property
id INT
Host_id INT
Name VARCHAR(255)
Address VARCHAR(255)
Amenities TEXT
Indexes
propertyamenity
id INT
Property_id INT
Amenity_id INT
Indexes
amenity
id INT
Name VARCHAR(255)
Description TEXT
Indexes
calendar
id INT
Property_id INT
Date DATE
Availability TINYINT(1)
Indexes
image
id INT
URL VARCHAR(255)
Name VARCHAR(255)
property_id INT
Indexes
host
id INT
user_id INT
property_id INT
Listing_status VARCHAR(255)
Indexes
room
id INT
Rating DECIMAL(10,2)
Comment TEXT
Street VARCHAR(255)
property_id INT
Indexes
room_has_propertyamenity
room_id INT
propertyamenity_id INT
Indexes