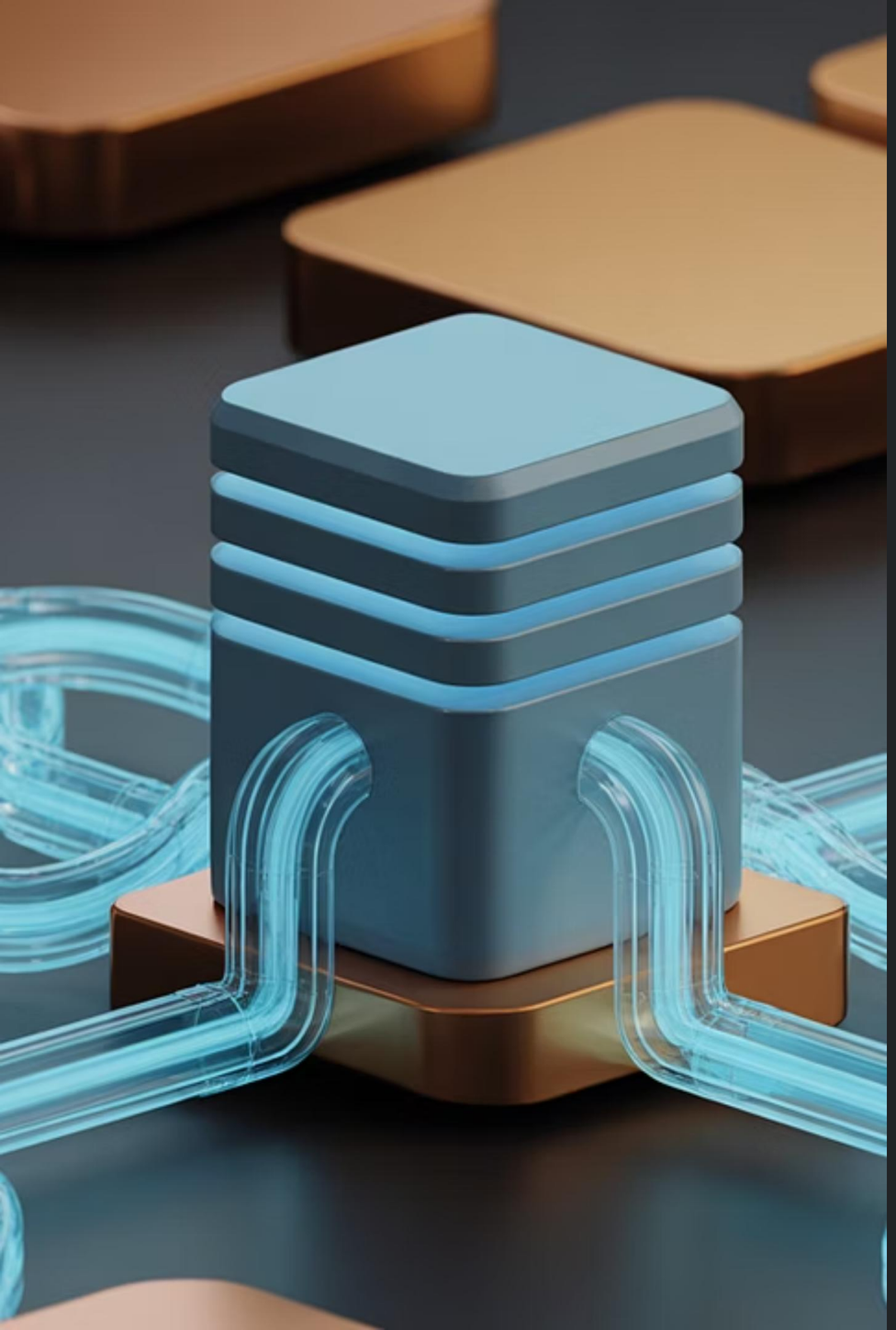


Phase 2: Airbnb Data Mart Implementation

By: Gehad Awadh Yahya Al-Nami
Matriculation Nr: 32106098

Development Phase Overview

1. ## Implement SQL schema & dummy data

   Create comprehensive database structure with properly defined tables, relationships, constraints, and populate with realistic test data

2. ## Document every DDL & INSERT statement

   Provide detailed documentation of all Data Definition Language statements and data insertion queries for future reference

3. ## Provide at least 20 entries per table

   Ensure sufficient data volume to test relationships, constraints, and query performance across all tables

4. ## Run test queries & screenshot results

   Execute verification queries to demonstrate proper functionality and document the results visually

5. ## Reflect on implementation steps

   Analyze challenges, solutions, and lessons learned throughout the implementation process

Database Design Process

1 Requirements Analysis

Captured key user roles (guest, host, admin) and their actions (search, book, review), along with the data required for each action.

2 Entity-Relationship Modeling

Crafted a comprehensive ER model with 21 core entities and their relationships, using Chen notation to show cardinalities.

3 Data Modeling

Assigned atomic attributes to each entity and produced a data dictionary documenting data types and key roles (PK, FK).

4 Normalization

Eliminated redundancy and ensured data integrity by normalizing the model to Third Normal Form, splitting out lookup tables, join tables, and sub-entities.

5 SQL Implementation

The logical schema directly shaped the SQL implementation.

Entity-Relationship Model

21 entities with 1:1, 1:N and M:N relationships representing the comprehensive Airbnb data model

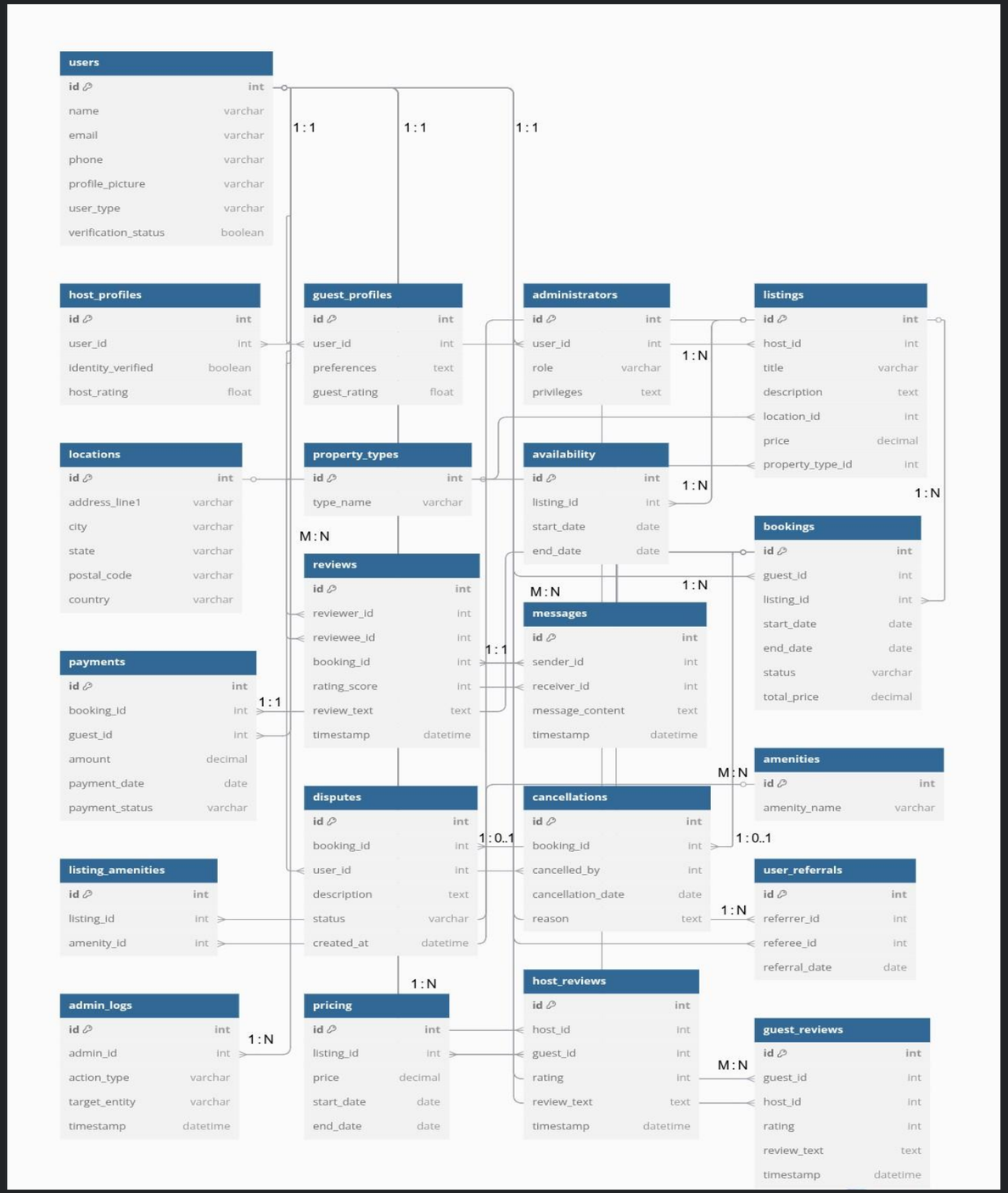


Table: users

```sql
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
  phone VARCHAR(20),
  user_type VARCHAR(10),
  verification_status BOOLEAN
);

INSERT INTO users (id, name, email, phone,
user_type, verification_status) VALUES
(1, 'Alice Smith', 'alice@gmail.com', '123-111-
1111', 'guest', TRUE),

(2, 'Bob Johnson', 'bob@gmail.com', '123-222-
2222', 'host', TRUE),


...
```

Query #1 **Execution time: 0.74ms**

id	name	email	phone	user_type	verification_status
1	Alice Smith	alice@gmail.com	123-111-1111	guest	1
2	Bob Johnson	bob@gmail.com	123-222-2222	host	1
3	Clara Lee	clara@gmail.com	123-333-3333	guest	0
4	David Kim	david@gmail.com	123-444-4444	host	1
5	Ella Brown	ella@gmail.com	123-555-5555	admin	1
6	Frank Liu	frank@gmail.com	123-666-6666	guest	1
7	Grace Hall	grace@gmail.com	123-777-7777	host	0
8	Henry Green	henry@gmail.com	123-888-8888	guest	1
9	Isla Patel	isla@gmail.com	123-999-9999	guest	1
10	Jake Moore	jake@gmail.com	123-101-0101	host	1
11	Karen White	karen@gmail.com	123-121-2121	guest	1

SELECT * FROM users;

Table: host_profiles

```
CREATE TABLE host_profiles (
  user_id INT PRIMARY KEY,
  identity_verified TINYINT(1),
  host_rating DECIMAL(2,1),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
INSERT INTO host_profiles (user_id,
identity_verified, host_rating) VALUES
(2,1,4.8),
(4,1,4.5),
(7,0,0.0),

...
```

Query #1 **Execution time: 0.68ms**

user_id	identity_verified	host_rating
1	0	0.0
2	1	4.8
3	0	0.0
4	1	4.5
5	1	4.9
6	1	4.5
7	0	0.0
8	1	4.4

SELECT * FROM host_profiles;

Table: locations

```sql
CREATE TABLE locations (
  id INT PRIMARY KEY,
  address_line1 VARCHAR(100),
  city VARCHAR(50),
  state VARCHAR(50),
  postal_code VARCHAR(20),
  country VARCHAR(50)
);

INSERT INTO locations (id, address_line1, city,
state, postal_code, country) VALUES
(1, 'Hauptstraße 1', 'Berlin', 'Berlin',
'10115', 'Germany'),

(2, 'Goethestraße 12', 'Frankfurt', 'Hesse',
'60313', 'Germany'),

...
```

Query #1 **Execution time: 0.99ms**

id	address_line1	city	state	postal_code	country
4	Marienplatz 8	Munich	Bavaria	80331	Germany
13	Altstadtstraße 22	Nuremberg	Bavaria	90403	Germany
14	Zugspitzstraße 6	Garmisch	Bavaria	82467	Germany
17	Ludwigstraße 9	Regensburg	Bavaria	93047	Germany

SELECT * FROM locations WHERE state = 'Bavaria';

Table: payments

```
CREATE TABLE payments (
  id INT PRIMARY KEY,
  booking_id INT,
  guest_id INT,
  amount DECIMAL(10,2),
  payment_date DATE,
  payment_status VARCHAR(20),
  payment_method VARCHAR(20),
  FOREIGN KEY (booking_id) REFERENCES
bookings(id),
  FOREIGN KEY (guest_id) REFERENCES users(id)
);

INSERT INTO payments (id, booking_id, guest_id,
amount, payment_date, payment_status,
payment_method) VALUES
(1, 1, 1, 380.00, '2024-07-30', 'paid',
'credit_card'),

(2, 2, 3, 482.00, '2024-08-08', 'paid',
'paypal'),

...
```

Query #1 **Execution time: 0.4ms**

id	booking_id	guest_id	amount	payment_date	payment_status	payment_method
1	1	1	380.00	2024-07-30	paid	credit_card
2	2	3	482.00	2024-08-08	paid	paypal
4	4	8	140.00	2024-08-30	paid	debit_card
5	5	9	750.00	2024-09-02	paid	paypal
7	7	13	340.00	2024-09-10	paid	credit_card
8	8	16	180.00	2024-09-18	paid	bank_transfer
9	9	18	650.00	2024-09-30	paid	credit_card

SELECT * FROM payments WHERE payment_status='paid';

Table: listing_amenities

```
CREATE TABLE listing_amenities (
  id INT PRIMARY KEY,
  listing_id INT,
  amenity_id INT,
  FOREIGN KEY (listing_id) REFERENCES
listings(id),
  FOREIGN KEY (amenity_id) REFERENCES
amenities(id)
);
INSERT INTO listing_amenities (id, listing_id,
amenity_id) VALUES
(1,1,1),
(2,1,3),
...
```

Query #1 **Execution time: 0.34ms**

id	listing_id	amenity_id
1	1	1
2	1	3
3	2	2
4	2	7
5	3	6
6	3	5
7	4	9

SELECT * FROM listing_amenities;

Table: admin_logs

```
CREATE TABLE admin_logs (
  id INT PRIMARY KEY,
  admin_id INT,
  action_type VARCHAR(50),
  target_entity VARCHAR(50),
  timestamp DATETIME,
  FOREIGN KEY (admin_id) REFERENCES users(id)
);
INSERT INTO admin_logs (id, admin_id,
action_type, target_entity, timestamp) VALUES

(1,5,'create_user','users','2024-07-01
09:00:00'),

(2,5,'delete_user','users','2024-07-02
10:15:00'),

...
```

Query #1 **Execution time: 0.5ms**

id	admin_id	action_type	target_entity	timestamp
1	5	create_user	users	2024-07-01 09:00:00
2	5	delete_user	users	2024-07-02 10:15:00
3	14	approve_listing	listings	2024-07-03 11:30:00
4	14	reject_listing	listings	2024-07-04 12:45:00
5	5	refund_payment	payments	2024-07-05 14:00:00
6	5	resolve_dispute	disputes	2024-07-06 15:15:00
7	14	ban_user	users	2024-07-07 16:30:00
8	14	unban_user	users	2024-07-08 17:45:00

SELECT * FROM admin_logs;

Table: guest_profiles

```
CREATE TABLE guest_profiles (
  user_id INT PRIMARY KEY,
  preferences TEXT,
  guest_rating DECIMAL(2,1),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
INSERT INTO guest_profiles (user_id,
preferences, guest_rating) VALUES
(1,'early check-in, quiet',4.9),
(2,'no smoking',4.2),

...
```

Query #1 **Execution time: 0.33ms**

user_id	preferences	guest_rating
1	early check-in, quiet	4.9
2	no smoking	4.2
3	parking needed	4.6
4	pets allowed	4.3
5	family friendly	4.8
6	near transit	4.1
7	long stays	4.5

SELECT * FROM guest_profiles;

Table: property_types

```
CREATE TABLE property_types (
  id INT PRIMARY KEY,
  type_name VARCHAR(50)
);

INSERT INTO property_types (id, type_name)
VALUES
(1, 'Apartment’),
(2, 'House’),
(3, 'Villa’),
(4, 'Loft’),
(5, 'Studio');
```

Query #1 **Execution time: 0.43ms**

id	type_name
1	Apartment
2	House
3	Villa
4	Loft
5	Studio

SELECT * FROM property_types;

Table: reviews

```
CREATE TABLE reviews (
  id INT PRIMARY KEY,
  reviewer_id INT,
  reviewee_id INT,
  booking_id INT,
  rating_score INT,
  review_text TEXT,
  timestamp DATETIME,
  FOREIGN KEY (reviewer_id) REFERENCES
users(id),
  FOREIGN KEY (reviewee_id) REFERENCES
users(id),
  FOREIGN KEY (booking_id) REFERENCES
bookings(id)
);

INSERT INTO reviews (id, reviewer_id,
reviewee_id, booking_id, rating_score,
review_text, timestamp) VALUES

(1, 1, 2, 1, 5, 'Great stay! Very clean and
quiet.', '2024-08-06 10:15:00'),

(2, 3, 4, 2, 4, 'Nice place, just a bit noisy.',
'2024-08-15 09:45:00'),

...
```

Query #1 **Execution time: 0.96ms**

id	reviewer_id	reviewee_id	booking_id	rating_score	review_text	timestamp
1	1	2	1	5	Great stay! Very clean and quiet.	2024-08-06 10:15:00
2	3	4	2	4	Nice place, just a bit noisy.	2024-08-15 09:45:00
9	1	2	11	5	Host made sure we had everything.	2024-08-19 16:10:00
3	8	10	4	5	Host was super helpful and friendly.	2024-09-04 20:30:00
19	6	19	13	4	Comfy and warm stay.	2024-09-06 08:00:00
11	6	7	13	4	Smooth check-in and quiet area.	2024-09-07 14:30:00
4	9	12	5	5	Exactly as described. Highly recommend!	2024-09-11 08:10:00

SELECT * FROM reviews WHERE rating_score >=4 ORDER BY timestamp;

Table: disputes

```
CREATE TABLE disputes (
  id INT PRIMARY KEY,
  booking_id INT,
  user_id INT,
  description TEXT,
  status VARCHAR(20),
  created_at DATETIME,
  FOREIGN KEY (booking_id) REFERENCES
bookings(id),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
INSERT INTO disputes (id, booking_id, user_id,
description, status, created_at) VALUES

(1,3,6,'Payment not processed','open','2024-07-
20 10:00:00'),

(2,6,11,'Listing not as described','open','2024-
07-17 09:30:00'),

...
```

Query #1 **Execution time: 0.29ms**

id	booking_id	user_id	description	status	created_at
1	3	6	Payment not processed	open	2024-07-20 10:00:00
2	6	11	Listing not as described	open	2024-07-17 09:30:00
3	12	3	Noise complaint	resolved	2024-08-29 12:00:00
4	14	8	Late check-in	open	2024-09-06 15:00:00
5	17	13	Cleaning issue	resolved	2024-11-04 10:15:00
6	19	18	WiFi down	open	2024-11-13 11:20:00
7	20	20	Missing amenities	resolved	2024-11-19 14:30:00
8	2	3	Overcharge	resolved	2024-08-14 16:00:00

SELECT * FROM disputes;

Table: pricing

```
CREATE TABLE pricing (
  id INT PRIMARY KEY,
  listing_id INT,
  price DECIMAL(10,2),
  start_date DATE,
  end_date DATE,
  FOREIGN KEY (listing_id) REFERENCES
listings(id)
);
INSERT INTO pricing (id, listing_id, price,
start_date, end_date) VALUES
(1,1,95.00,'2024-07-01','2024-07-31’),

(2,2,120.50,'2024-07-01','2024-07-31’),

...
```

Query #1 **Execution time: 0.25ms**

id	listing_id	price	start_date	end_date
1	1	95.00	2024-07-01	2024-07-31
2	2	120.50	2024-07-01	2024-07-31
3	3	105.75	2024-07-01	2024-07-31
4	4	70.00	2024-07-01	2024-07-31
5	5	150.00	2024-07-01	2024-07-31
6	6	200.00	2024-07-01	2024-07-31
7	7	85.00	2024-07-01	2024-07-31
8	8	60.00	2024-07-01	2024-07-31
9	9	130.00	2024-07-01	2024-07-31

SELECT * FROM pricing;

Table: administrations

```
CREATE TABLE administrators (
  user_id INT PRIMARY KEY,
  role_description VARCHAR(100),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
INSERT INTO administrators (user_id,
role_description) VALUES
(1,'full system access’),
(2,'audit logs manager’),

...
```

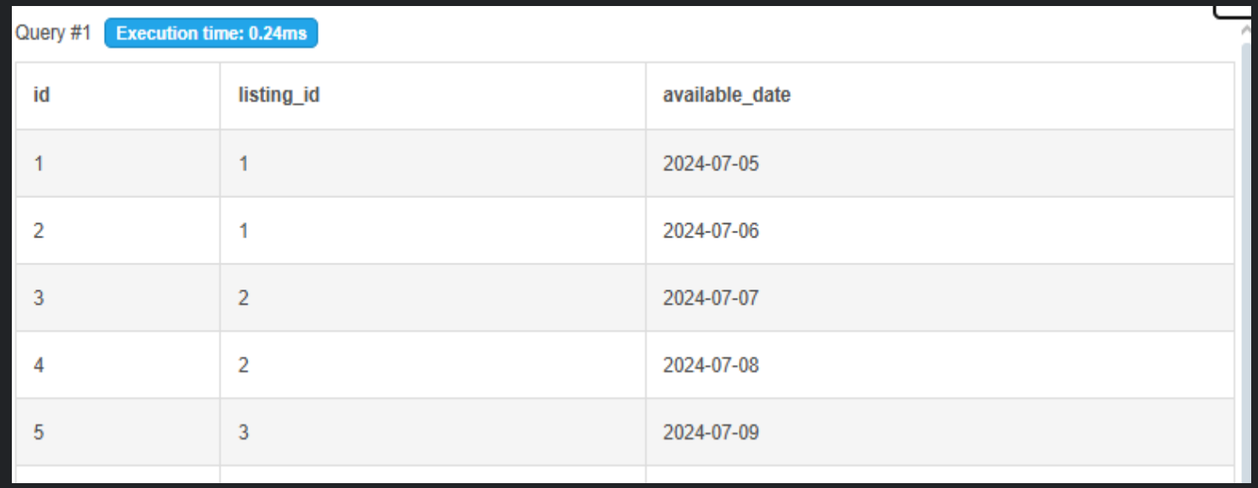
Query #1 **Execution time: 0.27ms**

user_id	role_description
1	full system access
2	audit logs manager
3	test admin
4	support agent

SELECT * FROM administrators;

Table: availability

```
CREATE TABLE availability (
  id INT PRIMARY KEY,
  listing_id INT,
  available_date DATE,
  FOREIGN KEY (listing_id) REFERENCES
listings(id)
);
INSERT INTO availability (id, listing_id,
available_date) VALUES
(1,1,'2024-07-05’),
(2,1,'2024-07-06’),
...
```



Query #1 **Execution time: 0.24ms**

id	listing_id	available_date
1	1	2024-07-05
2	1	2024-07-06
3	2	2024-07-07
4	2	2024-07-08
5	3	2024-07-09

SELECT * FROM availability;

Table: messages

```sql
CREATE TABLE messages (
  id INT PRIMARY KEY,
  sender_id INT,
  receiver_id INT,
  message_text TEXT,
  sent_at DATETIME,
  FOREIGN KEY (sender_id) REFERENCES users(id),
  FOREIGN KEY (receiver_id) REFERENCES users(id)
);

INSERT INTO messages (id, sender_id,
receiver_id, message_text, sent_at) VALUES

(1, 1, 2, 'Hi, is the apartment still available
from August 1st?', '2024-07-25 09:00:00'),

(2, 2, 1, 'Yes, it is available. Feel free to
book!', '2024-07-25 09:10:00'),

...
```

Query #1 **Execution time: 0.35ms**

id	sender_id	receiver_id	message_text	sent_at
1	1	2	Hi, is the apartment still available from August 1st?	2024-07-25 09:00:00
2	2	1	Yes, it is available. Feel free to book!	2024-07-25 09:10:00
3	3	4	Hello, can I check in early?	2024-08-01 08:30:00
4	4	3	Sure, early check-in is okay after 1 PM.	2024-08-01 08:45:00
5	6	7	Thanks for hosting me, it was a great stay!	2024-08-15 11:00:00
6	7	6	Glad you enjoyed it. Take care!	2024-08-15 11:30:00
7	8	10	Hi! Is your listing pet-friendly?	2024-08-20 14:00:00

SELECT * FROM messages ORDER BY sent_at;

Table: cancellations

```
CREATE TABLE cancellations (
  id INT PRIMARY KEY,
  booking_id INT,
  cancelled_by INT,
  cancellation_date DATE,
  reason TEXT,
  FOREIGN KEY (booking_id) REFERENCES
bookings(id),
  FOREIGN KEY (cancelled_by) REFERENCES
users(id)
);
INSERT INTO cancellations (id, booking_id,
cancelled_by, cancellation_date, reason) VALUES
(1,3,6,'2024-07-19','guest no-show'),

(2,6,11,'2024-07-14','double-booked'),

...
```

Query #1 **Execution time: 0.26ms**

id	booking_id	cancelled_by	cancellation_date	reason
1	3	6	2024-07-19	guest no-show
2	6	11	2024-07-14	double-booked
3	11	1	2024-08-18	changed plans
4	12	3	2024-08-28	health issue
5	13	6	2024-09-06	weather
6	14	8	2024-09-11	emergency

SELECT * FROM cancellations;

Table: host_reviews

```
CREATE TABLE host_reviews (
  id INT PRIMARY KEY,
  host_id INT,
  guest_id INT,
  rating INT,
  review_text TEXT,
  timestamp DATETIME,
  FOREIGN KEY (host_id)  REFERENCES users(id),
  FOREIGN KEY (guest_id) REFERENCES users(id));

INSERT INTO host_reviews (id, host_id, guest_id,
rating, review_text, timestamp) VALUES

(1,  2,  1, 5, 'Excellent guest, left the place
spotless','2024-07-06 10:00:00'),

(2,  2,  3, 4, 'Good communication and punctual
check-in','2024-07-07 11:15:00'),

...
```

Query #1 **Execution time: 0.35ms**

id	host_id	guest_id	rating	review_text	timestamp
1	2	1	5	Excellent guest, left the place spotless	2024-07-06 10:00:00
2	2	3	4	Good communication and punctual check-in	2024-07-07 11:15:00
3	4	6	5	Very respectful guest, would host again	2024-07-08 12:30:00
4	4	8	3	Quiet guest but late checkout	2024-07-09 13:45:00
5	7	9	4	Friendly and tidy, enjoyed hosting	2024-07-10 15:00:00
6	7	11	5	Outstanding guest, followed all house rules	2024-07-11 16:15:00
7	10	13	4	Good guest, minor issue with noise	2024-07-12 17:30:00

SELECT * FROM host_reviews;

Table: listings

```
CREATE TABLE listings (
  id INT PRIMARY KEY,
  host_id INT,
  title VARCHAR(100),
  description TEXT,
  price DECIMAL(10,2),
  property_type_id INT,
  location_id INT,
  FOREIGN KEY (host_id) REFERENCES users(id),
  FOREIGN KEY (property_type_id) REFERENCES
property_types(id),
  FOREIGN KEY (location_id) REFERENCES
locations(id)
);

INSERT INTO listings (id, host_id, title,
description, price, property_type_id, location_id)
VALUES

(1, 2, 'Modern Apartment in Berlin', 'Bright space
near city center', 95.00, 1, 1),

(2, 4, 'Cozy House in Munich', 'Private garden and
kitchen', 120.50, 2, 4),

...
```

id	host_id	title	description	price	property_type_id	location_id
1	2	Modern Apartment in Berlin	Bright space near city center	95.00	1	1
2	4	Cozy House in Munich	Private garden and kitchen	120.50	2	4
3	7	Stylish Loft in Hamburg	Great nightlife nearby	105.75	4	5
4	10	Quiet Studio in Leipzig	Peaceful and modern	70.00	5	9
5	12	Family Home in Frankfurt	Perfect for long stays	150.00	2	2
6	15	Charming Villa in Dresden	Luxury stay near palace	200.00	3	10
7	17	Apartment in Stuttgart	Close to transport and shops	85.00	1	6
8	19	Budget Studio in Cologne	Great for solo travelers	60.00	5	7

SELECT * FROM listings;

Table: bookings

```sql
CREATE TABLE bookings (
  id INT PRIMARY KEY,
  guest_id INT,
  listing_id INT,
  start_date DATE,
  end_date DATE,
  status VARCHAR(20),
  total_price DECIMAL(10,2),
  FOREIGN KEY (guest_id) REFERENCES users(id),
  FOREIGN KEY (listing_id) REFERENCES
listings(id)
);

INSERT INTO bookings (id, guest_id, listing_id,
start_date, end_date, status, total_price)
VALUES

(1, 1, 1, '2024-08-01', '2024-08-05',
'confirmed', 380.00),

(2, 3, 2, '2024-08-10', '2024-08-14',
'confirmed', 482.00),


...
```

Query #1 **Execution time: 0.61ms**

id	guest_id	listing_id	start_date	end_date	status	total_price
1	1	1	2024-08-01	2024-08-05	confirmed	380.00
2	3	2	2024-08-10	2024-08-14	confirmed	482.00
3	6	3	2024-07-20	2024-07-23	cancelled	0.00
4	8	4	2024-09-01	2024-09-03	confirmed	140.00
5	9	5	2024-09-05	2024-09-10	confirmed	750.00
6	11	6	2024-07-15	2024-07-17	cancelled	0.00
7	13	7	2024-09-12	2024-09-16	confirmed	340.00

SELECT * FROM bookings;

Table: amenities

```
CREATE TABLE amenities (
  id INT PRIMARY KEY,
  amenity_name VARCHAR(50)
);
INSERT INTO amenities (id, amenity_name) VALUES
(1,'WiFi’),
(2,'Air Conditioning’),
(3,'Kitchen’),

...
```

Query #1 **Execution time: 0.21ms**

id	amenity_name
1	WiFi
2	Air Conditioning
3	Kitchen
4	Heating
5	TV

SELECT * FROM amenities;

Table: user_referrals

```
CREATE TABLE listing_amenities (
  id INT PRIMARY KEY,
  listing_id INT,
  amenity_id INT,
  FOREIGN KEY (listing_id) REFERENCES
listings(id),
  FOREIGN KEY (amenity_id) REFERENCES
amenities(id)
);
INSERT INTO listing_amenities (id, listing_id,
amenity_id) VALUES
(1,1,1),

(2,1,3),

(3,2,2),

...
```

Query #1 **Execution time: 0.3ms**

id	referrer_id	referee_id	referral_date
1	2	6	2024-07-01
2	3	7	2024-07-02
3	4	8	2024-07-03
4	5	9	2024-07-04
5	6	10	2024-07-05
6	7	11	2024-07-06

SELECT * FROM user_referrals ORDER BY referral_date;

Join Query : listings & users & property_types & locations

```
SELECT
  l.title,
  u.name          AS host_name,
  p.type_name     AS property_type,
  loc.city,
  l.price
FROM listings l
JOIN users u ON l.host_id = u.id
JOIN property_types p ON l.property_type_id = p.id
JOIN locations loc ON l.location_id = loc.id;
```

Query #1 **Execution time: 12.96ms**

title	host_name	property_type	city	price
Modern Apartment in Berlin	Bob Johnson	Apartment	Berlin	95.00
Apartment in Stuttgart	Quinn Wood	Apartment	Stuttgart	85.00
Attic Apartment in Bremen	Grace Hall	Apartment	Bremen	80.00
Budget Apartment in Berlin	Jake Moore	Apartment	Berlin	65.00
Student Flat in Regensburg	David Kim	Apartment	Regensburg	70.00
Cozy House in Munich	David Kim	House	Munich	120.50
Family Home in Frankfurt	Leo Black	House	Frankfurt	150.00
Modern House in Bonn	Leo Black	House	Bonn	145.00
Tiny House in Nuremberg	Bob Johnson	House	Nuremberg	75.00
Elegant House in Erfurt	Quinn Wood	House	Erfurt	130.00
Charming Villa in Dresden	Oscar Fox	Villa	Dresden	200.00

This join query connects listing data with host information, property type details, and location data to provide comprehensive listing information.

Join Query: Confirmed Bookings with Guest & Host

```
SELECT
  b.id            AS booking_id,
  guest.name      AS guest_name,
  host.name       AS host_name,
  l.title         AS listing_title,
  b.start_date,
  b.end_date,
  b.total_price
FROM bookings b
JOIN users guest    ON b.guest_id   = guest.id
JOIN listings l     ON b.listing_id = l.id
JOIN users host     ON l.host_id    = host.id
WHERE b.status = 'confirmed';
```

Query #1 **Execution time: 1.57ms**

booking_id	guest_name	host_name	listing_title	start_date	end_date	total_price
1	Alice Smith	Bob Johnson	Modern Apartment in Berlin	2024-08-01	2024-08-05	380.00
2	Clara Lee	David Kim	Cozy House in Munich	2024-08-10	2024-08-14	482.00
4	Henry Green	Jake Moore	Quiet Studio in Leipzig	2024-09-01	2024-09-03	140.00
5	Isla Patel	Leo Black	Family Home in Frankfurt	2024-09-05	2024-09-10	750.00
7	Mona Adams	Quinn Wood	Apartment in Stuttgart	2024-09-12	2024-09-16	340.00
8	Paula Ray	Sara Khan	Budget Studio in Cologne	2024-09-20	2024-09-23	180.00
9	Ravi Singh	Jake Moore	Open Loft in Düsseldorf	2024-10-01	2024-10-06	650.00
10	Tom Dean	Leo Black	Modern House in Bonn	2024-10-10	2024-10-12	290.00

This multi-table query demonstrates the relationship between bookings, users (as both guests and hosts), and listings to provide a comprehensive view of confirmed reservations.

Implementation & Testing Procedure

1

Database Setup

Implemented the schema in MySQL 8.0 using InnoDB for ACID compliance and utf8mb4 for full Unicode support.

2

Table Creation

Wrote CREATE TABLE statements for each entity, defining primary and foreign keys with ON UPDATE CASCADE and ON DELETE RESTRICT.

3

Data Seeding

Inserted 20 realistic dummy records - German addresses, sample reviews and booking scenarios.

4

Data Types

Used TINYINT(1) for boolean flags and DECIMAL(10,2) for monetary fields.

5

Validation Queries

Executed targeted test queries to validate table relationships, data integrity, and query functionality.

6

Documentation

Captured query results as screenshots and documented all SQL scripts.

Database Normalization

All tables in 3NF

The entire schema adheres to Third Normal Form, ensuring that every non-key attribute depends exclusively on its table's primary key and never on other non-key fields. This design eliminates transitive dependencies, prevents update anomalies, and keeps the data model lean and consistent.

No redundant or derived data

By avoiding redundant or derived columns, the database stores each piece of information exactly once. For example, rather than persisting a "booking duration," queries calculate it dynamically from the stored start_date and end_date values, ensuring accuracy and reducing storage overhead.

FKs enforce referential integrity

Referential integrity is strictly enforced through foreign key constraints, which guarantee that every child record points to an existing parent row. This prevents orphaned records and maintains data consistency across all related tables.

Technical Details

1 MySQL v8 used

Implementation leverages MySQL 8.0's enhanced features including atomic DDL statements, improved JSON support, and window functions for advanced analytics capabilities.

2 Boolean stored as TINYINT(l)

Boolean values are implemented using TINYINT(1) for maximum compatibility, where 0 represents FALSE and 1 represents TRUE across all relevant tables.

3 Join tables for all M:N relationships

Many-to-many relationships are properly normalized using junction tables with composite keys to maintain referential integrity and optimize query performance.

Key Challenges

Designing recursive referrals

Implementing the user_referrals table presented challenges with self-referencing relationships. The solution required careful handling of foreign key constraints to ensure both referrer_id and referred_id could properly reference the users table without creating circular dependencies.

Ensuring test data covers cascade rules

Creating comprehensive test data that properly exercised all cascading update and delete rules required careful planning. Each relationship needed to be thoroughly tested to verify that referential integrity constraints functioned correctly under various scenarios.

Balancing realism vs. simplicity

Finding the right balance between creating a realistic data model and maintaining simplicity for educational purposes was challenging. The final schema achieves a good compromise by including essential relationships while avoiding unnecessary complexity.

Future Enhancements

Add calendar availability table

Implement a dedicated calendar_availability table to track listing availability with granular date-level control, enabling complex booking rules and blackout dates.

Dispute resolution module

Create tables for tracking customer service interactions, dispute cases, and resolution outcomes to improve the booking experience.

Reporting & analytics views

Develop a set of materialized views and summary tables to optimize common reporting queries and enable real-time business intelligence capabilities.

Reflection

In Phase 2 I implemented and tested a fully normalized and scalable SQL schema for Airbnb. I populated each table with 20+ rows of realistic entries data, documented every SQL statement, and verified all relationships with test queries. This groundwork sets the stage for Phase 3: finalization and optimization.

The implementation process reinforced the importance of careful planning before execution. By designing a comprehensive entity-relationship model first, I was able to identify potential issues early and create a more robust schema. The test data creation process was particularly valuable for understanding how real world scenarios would map to database operations.

One key insight was the value of junction tables for handling complex relationships. These tables not only simplified the schema but also improved query performance for common operations. I also gained appreciation for the power of foreign key constraints in maintaining data integrity across the entire system.