Phase 1 – Concept Documents

Airbnb Database model requirements analysis – Tom Schmaeling

In this short paper we will analyze the web application Airbnb to try and imitate the database model it uses for its many functionalities.

When imitating the Airbnb database model, it is crucial to provide robust support scalability, ensuring the system can handle vast amounts of data, user profiles, and transactions. Additionally, flexibility in schema design is vital to accommodate diverse property types, amenities, allowing seamless modifications to adapt to changing business requirements and user needs.

To achieve this, we will start by studying and researching the Airbnb website as well as the assignment text, as it also contains some relevant information about the application. While doing so we will note down relevant user groups and dynamic information that is used by the applications' functionalities.

By first walking through the most common use cases of the website we can identify multiple entities right away, the following list is the result after a simple "walk-through" of the website and noting all relevant entities in a mostly chronological order:

- Users
- Hosts
- Guests
- Addresses
- Emergency Contacts
- Property listings
- Property type
- Property categories
- Property amenities
- Reviews
- Ratings
- Images
- Wishlist
- Trip History
- Reservations / Bookings
- Chats
- Messages

What role some of these entities play in the application and what attributes might be relevant is listed in more detail below.

Once these entities have been noted down, it is essential to figure out how these entities might relate to each other so we can document their relationships as well as start thinking about the cardinalities of said relationships.

As the process is an iterative one, changes to entities or relationships are quite common during this step. The result should be cohesive, easily understandable and provide the template for implementation is the second phase.

The main problem this database is facing is how we structure the property listings and its different elements as well as the users and bookings and transactions. These are the most important entities in the database and will form the basis for the other tables.

Roles / User Groups

As noted in the assignment there are two main categories of users, hosts and guests. These two groups differ somewhat, in that they share most of their attributes but the requirements (NOT NULL) of such some attributes change depending on which type of user it is.

Generally, a Host is required to upload a profile image, personal information, a description and bank information to receive payments.

Guests, on the other hand, need to upload a government ID and credit card information. Important to note here is that a booing can only have one guest. Other guests are still handled through the booking guest user and are handled by the “number of guests” attribute.

We will use the joined subclass table strategy for this purpose. The reasoning for the “duplicate” attributes in guest and host is the difference in requirements.

Property-Listings

The term property-listings refers to the many different properties offered from hosts to guests on the website. Looking at the filters, we can see that a property has a type, such as house, apartment, guesthouse, or hotel and may also have multiple categories, such as cabin, tiny homes, mansions, etc.

When looking at a listing we see that there are images, a name, an address, prices per night (including cleaning fee, Airbnb service fee and taxes), reviews, descriptions, amenities and a calendar that displays the available times for booking. The host is also displayed on the property listing page.

Bookings

A booking, also called reservation lists the nightly and total price, the number of guests, a confirmation code, the timeframe (start and end date), etc. More specific payment information as well as other relevant information is stored in the transaction entity.

Addresses

The addresses in Airbnb are made up of country, state or region, and city. The street / house number are also part of the address but are not relevant when first filtering through the property listings.

Message

PK	**id**
	text
	created
	image_id
FK	*owned_chat_id*

BankInformation

PK	**id**
FK	*owning_host_id*
	account_nr
	code
FK	*address_id*

Chat

PK	**id**
FK	*chat_partner_id*
	created
	last_modified

CreditCardInformation

PK	**id**
FK	*owning_guest_id*
	bank
	card_number
	cvc_code
	expiration_date

GiftCard

PK	**code**
	amount_usd
FK	*currency_id*
	valid_until_date
	created

Transaction

PK	**id**
FK	*booking_id*
FK	*bankinformation_id*
FK	*creditcardinformation_id*
FK	*created*
	last_modified

Currency

PK	**id**
	name
	abbreviation

UserReview

PK	**id**
FK	*user_id*
	comment
	created
	last_modified

Language

PK	**id**
	name
	country

EmergencyContact

PK	**id**
	name
FK	*profile_image_id*
	relationship
FK	*prefered_language_id*
	email
	country_code
	phone_number
FK	*user_id*

User

PK	**id**
FK	address_id
	email
	legal_name
	government_id

Guest

PK	**id**
FK	*user_id*
FK	*profile_image_id*
	description
	phone_number
FK	*triphistory_id*

Host

PK	**id**
FK	*user_id*
FK	*profile_image_id*
	description
	phone_number

Address

PK	**id**
	country
	state
	city
	street
	house_nr
	zip_code

Booking

PK	**id**
	total_price
FK	*currency_id*
	confirmation_code
	cancelled
	cancellation_date
	refund_amount
FK	*transaction_id*
FK	*triphistory_id*
FK	*propertylisting_id*
FK	*host_id*
FK	*guest_id*
	created

Wishlist

PK	**id**
	name
	private
FK	*owning_user_id*

Wishlist_PropertyListing

PK	**id**
FK	*propertylisting_id*
FK	*wishlist_id*

Images

PK	**id**
FK	*uploaded_by_user_id*
	image_url
	created

TripHistory

PK	**id**

PropertyListing

PK	**id**
	name
FK	*address_id*
FK	*property_type_id*
	bedroom_count
	bathroom_count
	available_booking_slots
	price_per_night
	cleaning_fee
FK	*currency_id*
	check_in_time
	check_out_time
	average_rating
	total_nr_of_ratings

PropertyReview

PK	**id**
FK	*property_id*
FK	*author_guest_id*
	overall
	location
	cleanliness
	communication
	check_in
	accuracy
	images_id
	comment
	created

Property_Amenity

PK	**id**
FK	*property_id*
FK	*amenity_id*

Amenity

PK	**id**
	name
	icon_url
	created
	last_modified

Property_HouseRule

PK	**id**
FK	*property_id*
FK	*houserule_id*

HouseRule

PK	**id**
	name
	icon_url
	created
	last_modified

Category

PK	**id**
	name
	icon_url
	created
	last_modified

Property_Category

PK	**id**
FK	*property_id*
FK	*category_id*

PropertyType

PK	**id**
	name
	icon_url
	created
	last_modified

Location	Attribute Name	Data Type	Constraint
Address	address_id	INT	AUTO_INCREMENT PRIMARY KEY
Address	country	VARCHAR(128)	NOT NULL
Address	state	VARCHAR(128)	NOT NULL
Address	city	VARCHAR(128)	NOT NULL
Address	street	VARCHAR(128)	NOT NULL
Address	house_nr	INT	NOT NULL
Address	zip_code	INT	NOT NULL
Address	address_type	VARCHAR(32)	
User	user_id	INT	AUTO_INCREMENT PRIMARY KEY
User	address_id	INT	FOREIGN KEY
User	email	VARCHAR(128)	NOT NULL
User	legal_name	VARCHAR(64)	NOT NULL
User	governmentid_image_id	INT	
Guest	guest_id	INT	AUTO_INCREMENT PRIMARY KEY
Guest	user_id	INT	FOREIGN KEY UNIQUE NOT NULL
Guest	profile_image_id	INT	FOREIGN KEY UNIQUE NOT NULL
Guest	guest_description	TEXT	
Guest	phone_number	VARCHAR(16)	NOT NULL
Guest	triphistory_id	INT	FOREIGN KEY
Host	host_id	INT	AUTO_INCREMENT PRIMARY KEY
Host	user_id	INT	FOREIGN KEY UNIQUE NOT NULL
Host	profile_image_id	INT	FOREIGN KEY UNIQUE NOT NULL
Host	host_description	TEXT	NOT NULL
Host	phone_number	VARCHAR(16)	
Image	image_id	INT	AUTO_INCREMENT PRIMARY KEY
Image	uploaded_by_user_id	INT	FOREIGN KEY NOT NULL
Image	image_url	VARCHAR(1024)	NOT NULL
Image	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
UserReview	userreview_id	INT	AUTO_INCREMENT PRIMARY KEY
UserReview	user_id	INT	FOREIGN KEY NOT NULL
UserReview	author_user_id	INT	FOREIGN KEY NOT NULL
UserReview	comment	VARCHAR(2000)	
UserReview	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP

UserReview	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
PropertyListing	propertylisting_id	INT	AUTO_INCREMENT PRIMARY KEY
PropertyListing	owning_host_id	INT	FOREIGN KEY NOT NULL
PropertyListing	name	VARCHAR(255)	FOREIGN KEY NOT NULL
PropertyListing	address_id	INT	FOREIGN KEY UNIQUE NOT NULL
PropertyListing	property_type_id	INT	FOREIGN KEY
PropertyListing	bedroom_count	INT	
PropertyListing	bathroom_count	INT	
PropertyListing	available_booking_slots	INT	
PropertyListing	price_per_night	DECIMAL(10,2)	
PropertyListing	currency_id	INT	FOREIGN KEY NOT NULL DEFAULT 1
PropertyListing	check_in_time	TIME	DEFAULT '14:00'
PropertyListing	check_out_time	TIME	DEFAULT '10:00'
PropertyListing	average_rating	DECIMAL(2,1)	DEFAULT 0
PropertyListing	total_nr_of_ratings	INT	DEFAULT 0
PropertyReview	propertyreview_id	INT	AUTO_INCREMENT PRIMARY KEY
PropertyReview	property_id	INT	FOREIGN KEY NOT NULL
PropertyReview	author_guest_id	INT	FOREIGN KEY NOT NULL
PropertyReview	cleanliness_score	INT	
PropertyReview	accuracy_score	INT	
PropertyReview	check_in_score	INT	
PropertyReview	communication_score	INT	
PropertyReview	location_score	INT	
PropertyReview	value_score	INT	
PropertyReview	comment	VARCHAR(2000)	
PropertyReview	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Property_Category	property_category_id	INT	AUTO_INCREMENT PRIMARY KEY
Property_Category	name	VARCHAR(255)	FOREIGN KEY
Property_Category	icon_url	VARHCAR(1024)	FOREIGN KEY
Category	category_id	INT	AUTO_INCREMENT PRIMARY KEY
Category	name	VARHCAR(255)	
Category	icon_url	VARCHAR(1024)	
Category	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Category	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

Property_Amenity	property_amenity_id	INT	AUTO_INCREMENT PRIMARY KEY
Property_Amenity	property_id	INT	FOREIGN KEY
Property_Amenity	amenity_id	INT	FOREIGN KEY
Amenity	amenity_id	INT	AUTO_INCREMENT PRIMARY KEY
Amenity	name	VARCHAR(255)	
Amenity	icon_url	VARCHAR(1024)	
Amenity	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Amenity	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
Property_HouseRule	property_houserule_id	INT	AUTO_INCREMENT PRIMARY KEY
Property_HouseRule	property_id	INT	FOREIGN KEY
Property_HouseRule	houserule_id	INT	FOREIGN KEY
HouseRule	houserule_id	INT	AUTO_INCREMENT PRIMARY KEY
HouseRule	name	VARCHAR(255)	
HouseRule	icon_url	VARCHAR(1024)	
HouseRule	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
HouseRule	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
PropertyType	propertytype_id	INT	AUTO_INCREMENT PRIMARY KEY
PropertyType	name	VARCHAR(255)	
PropertyType	icon_url	VARCHAR(1024)	
PropertyType	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
PropertyType	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
Booking	booking_id	INT	AUTO_INCREMENT PRIMARY KEY
Booking	total_price	DECIMAL(10,2)	NOT NULL
Booking	currency_id	INT	FOREIGN KEY DEFAULT 1
Booking	booking_start_date	TIMESTAMP	NOT NULL
Booking	booking_end_date	TIMESTAMP	NOT NULL CHECK (booking_end_date > booking_start_date) *added via trigger.sql
Booking	confirmation_code	VARCHAR(16)	NOT NULL
Booking	cancelled	BOOLEAN	NOT NULL
Booking	cancellation_date	DATE	
Booking	refund_amount	DECIMAL(10,2)	
Booking	propertylisting_id	INT	FOREIGN KEY NOT NULL
Booking	host_id	INT	FOREIGN KEY NOT NULL
Booking	guest_id	INT	FOREIGN KEY NOT NULL
Booking	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP

Wishlist	wishlist_id	INT	AUTO_INCREMENT PRIMARY KEY
Wishlist	name	VARCHAR(255)	NOT NULL
Wishlist	private	BOOLEAN	NOT NULL DEFAULT TRUE
Wishlist	owning_user_id	INT	FOREIGN KEY NOT NULL
Wishlist_PropertyListing	wishlist_propertylisting_id	INT	AUTO_INCREMENT PRIMARY KEY
Wishlist_PropertyListing	propertylisting_id	INT	FOREIGN KEY
Wishlist_PropertyListing	wishlist_id	INT	FOREIGN KEY
EmergencyContact	emergencycontact_id	INT	AUTO_INCREMENT PRIMARY KEY
EmergencyContact	name	VARCHAR(255)	NOT NULL
EmergencyContact	relationship	VARCHAR(255)	
EmergencyContact	prefered_language_id	INT	FOREIGN KEY DEFAULT 1
EmergencyContact	email	VARCHAR(255)	
EmergencyContact	country_code	VARCHAR(32)	
EmergencyContact	phone_number	VARCHAR(15)	NOT NULL
EmergencyContact	owning_user_id	INT	FOREIGN KEY NOT NULL
Language	language_id	INT	AUTO_INCREMENT PRIMARY KEY
Language	name	VARCHAR(32)	NOT NULL
Language	country	VARCHAR(32)	
Currency	currency_id	INT	AUTO_INCREMENT PRIMARY KEY
Currency	name	VARCHAR(64)	NOT NULL
Currency	abbreviation	VARCHAR(3)	
GiftCard	giftcard_code	VARCHAR(16)	PRIMARY KEY
GiftCard	amount	DECIMAL(10,2)	NOT NULL
GiftCard	currency_id	INT	FOREIGN KEY NOT NULL DEFAULT 1
GiftCard	valid_until_date	TIMESTAMP	NOT NULL
GiftCard	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Transaction	transaction_id	INT	AUTO_INCREMENT PRIMARY KEY
Transaction	booking_id	INT	FOREIGN KEY NOT NULL
Transaction	bankinformation_id	INT	FOREIGN KEY
Transaction	creditcardinformation_id	INT	FOREIGN KEY
Transaction	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Transaction	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
CreditCardInformation	creditcardinformation_id	INT	AUTO_INCREMENT PRIMARY KEY
CreditCardInformation	owning_guest_id	INT	FOREIGN KEY NOT NULL

CreditCardInformation	bank	VARCHAR(128)	NOT NULL
CreditCardInformation	card_number	VARCHAR(64)	NOT NULL
CreditCardInformation	cvc_code	VARCHAR(3)	NOT NULL
CreditCardInformation	expiration_date	DATE	NOT NULL
BankInformation	bankinformation_id	INT	AUTO_INCREMENT PRIMARY KEY
BankInformation	owning_host_id	INT	FOREIGN KEY
BankInformation	name	VARCHAR(128)	NOT NULL
BankInformation	account_nr	VARCHAR(32)	NOT NULL
BankInformation	code	VARCHAR(64)	NOT NULL
BankInformation	address_id	INT	FOREIGN KEY NOT NULL
Chat	chat_id	INT	AUTO_INCREMENT PRIMARY KEY
Chat	chat_partner_id	INT	FOREIGN KEY
Chat	owner_user_id	INT	FOREIGN KEY
Chat	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP
Chat	last_modified	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
Message	message_id	INT	AUTO_INCREMENT PRIMARY KEY
Message	text	TEXT	
Message	image_id	INT	FOREIGN KEY
Message	author_user_id	INT	FOREIGN KEY
Message	owning_chat_id	INT	FOREIGN KEY
Message	created	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP

Phase 2 – Implementation Documents

AIRBNB DATABASE PRESENTATION

PHASE 2 - IMPLEMENTATION

By Tom Schmäling

TABLE OF CONTENTS

INTRODUCTION

This presentation is a part of phase 2 and intends to provide extensive documentation of the database structure, it's tables, relationships and constraints, as well as explain the provided test cases, and their results.

We will first explain the changes to the concept, the overall structure and elements of the database before examining each of the 27* tables that make up the database.

*One table has been removed; all changes can be found on the following slide.

I encourage the reader to run the test commands in their own environment for better readability, source code will be provided in phase 3.. any of the statements that are referred to but are not part of this presentation are very simple and are omitted without losing context. (usually simple select all test statements)

CHANGES

- I have decided to remove the triphistory table. The idea behind the table was to improve the access to the bookings of a guest for easier access. There is however not a significant enough improvement to justify the redundancy.
- There have been some adjustments to the attribute distribution of the user, guest and host tables after reflecting the requirements/constraints of the individual attributes.
- Small additional changes include:
  - The primary key's now have more descriptive names.
  - Addresses now have an 'address_type' attribute
  - User attribute "government_id' was changed to 'governmentid_image_id'
  - Messages now have an 'author_user_id' attribute
  - Bookings no longer have the 'transaction_id' attribute
  - PropertyListing now has the 'owning_host_id' attribute
  - PropertyReview attributes have changed to better align with the AirBnB app.
  - Currency attribute 'amount_usd' was changed to 'amount'
  - BankInformation now has a 'name' attribute for the bank's name

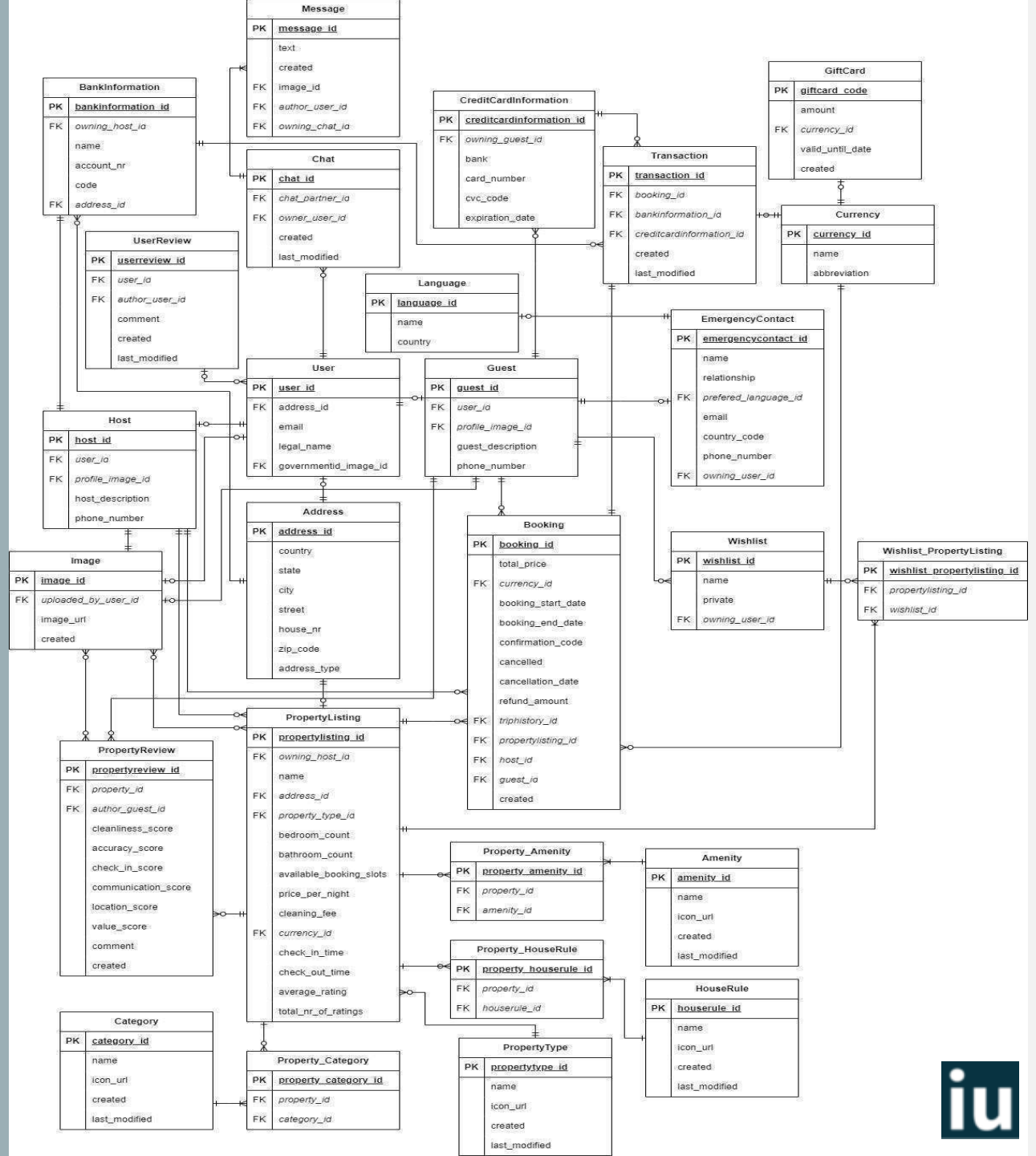
STRUCTURE

There are two main “blocks” of information the database needs to support: Users and Properties.

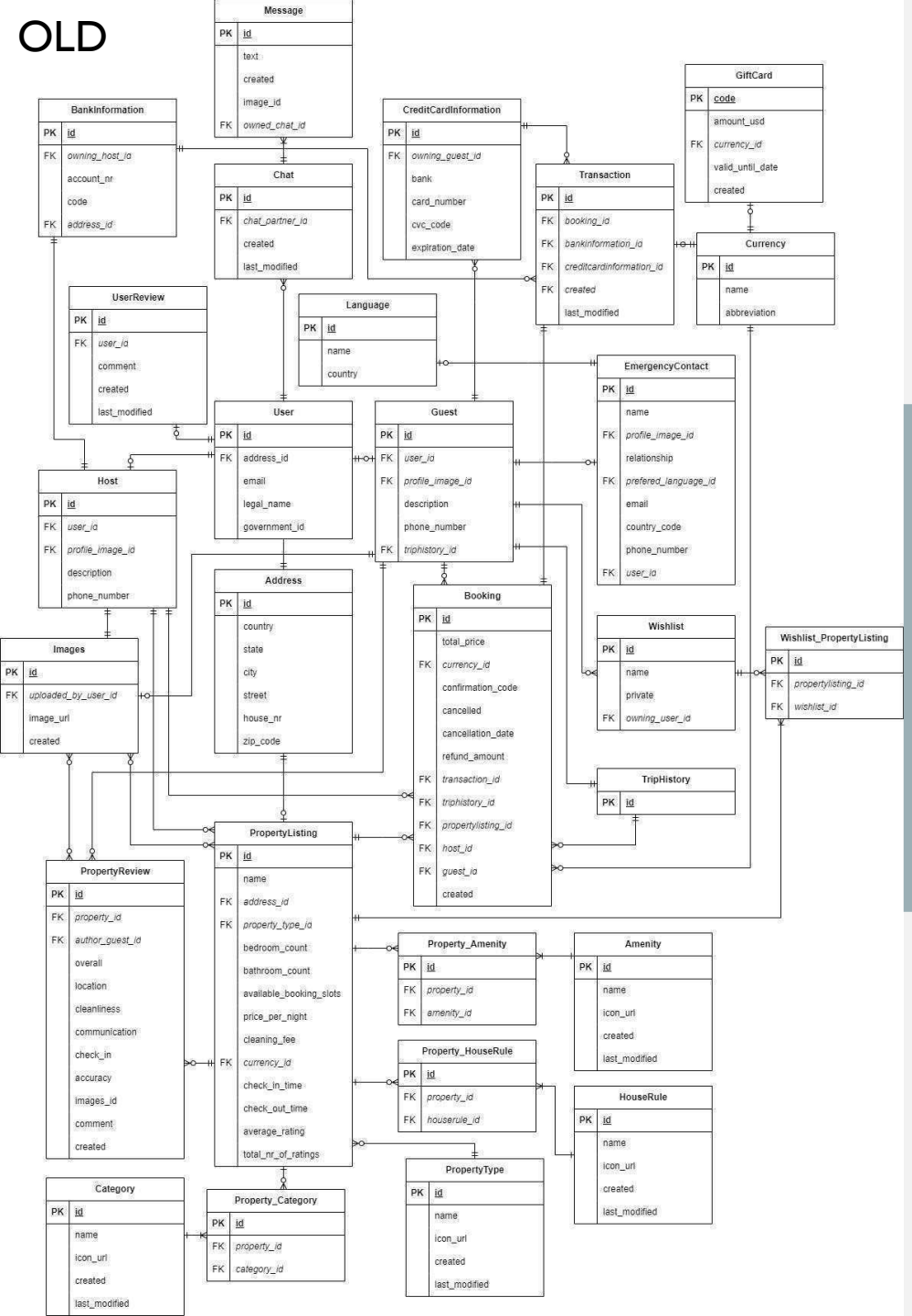
There are multiple tables facilitating each of these data sets, and their attributes.

The two categories of users, ‘guest’ and ‘host’ share a base ‘user’ class, creating a joined subclass table strategy.

The ‘PropertyListing’ table includes relevant attributes for the properties offered on the page, multiple of which use N:M relations and therefore need to be normalized via additional tables.

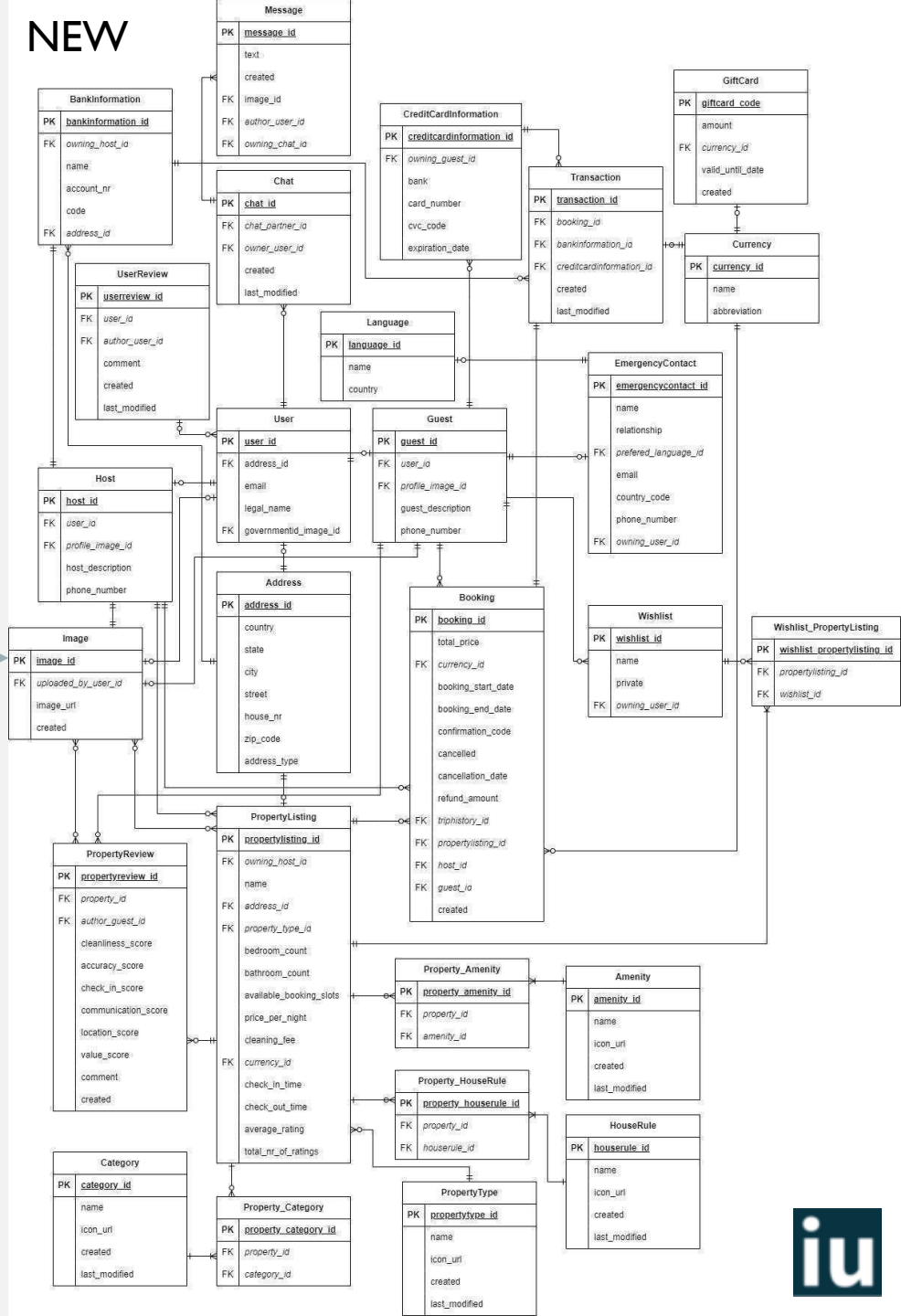


OLD



STRUCTURE CHANGES OVERVIEW

NEW



```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|       40 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE - USER

```
27  CREATE TABLE User (
28      user_id INT AUTO_INCREMENT PRIMARY KEY,
29      address_id INT, /*FK*/
30      email VARCHAR(128) NOT NULL,
31      legal_name VARCHAR(64) NOT NULL,
32      governmentid_image_id INT
33  );
```

```
263 /* Users constraints */
264 ALTER TABLE User
265 ADD CONSTRAINT fk_user_address
266 FOREIGN KEY (user_id)
267 REFERENCES Address(address_id)
268 ON DELETE CASCADE ON UPDATE RESTRICT;
```

User	
PK	user_id
FK	address_id
	email
	legal_name
FK	governmentid_image_id

*code line indicator in screenshots may differ slightly compared to the source file.

Tested via ‘Guest’ and ‘Host’ test cases
Insert statements are part of ‘Guest’/’Host’ transaction.

- The User is the base/super class for all users and holds attributes any user will have.
- The ‘address_id’ is a foreign key that references the address table.
- The ‘governmentid_image_id’ is a foreign key that references the image table.
- This table is tested via the host and guest tables which can be seen in the following two slides.

TABLE - GUEST

COUNT

```
mysql> SELECT COUNT(*) FROM Guest;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

- The ‘user_id’ attribute refers to the base class.
- ‘profile_image_id’ references the id of an image that can be loaded to display the users profile image. It refers to the image table.
- The test case is meant to test the relationship between the ‘Guest’ table/class and its super class ‘User’. As well as testing other relevant relationships via all the foreign keys.

INSERT

```sql

User, Guest, EmergencyContact, CreditCardInformation, Address (SQL Transaction)

Using a transaction for these insert statement means that we can use ROLLBACK to revert all changes
in case one or more querries fail.
We use a transaction for this because the data is cloesly related to each other.

* i will only comment the first transaction as the rest follow the same schematic
*/
-- Guest 1
-- start the transaction
START TRANSACTION;

-- Insert Guest's Address into Address table
INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
VALUES ("Germany", "Hessen", "Frankfurt", "Musterstraße", 1, "60306", "guest");
-- Retrieve the auto-generated address_id
SET @address_id = LAST_INSERT_ID();

-- Insert user data into User table, using the retrieved address_id
INSERT INTO User (address_id, email, legal_name)
VALUES (@address_id, "max.musterman@example.com", "Max Musterman");
-- Retrieve the auto-generated user_id
SET @user_id = LAST_INSERT_ID();

-- Insert Image of government id, using example link
INSERT INTO Image (uploaded_by_user_id, image_url)
VALUES (@user_id, "https://airbnb.com/images/governmentid_image1.jpg");
-- Retrieve image_id of goverment id picture
SET @image_governid_id = LAST_INSERT_ID();
-- Insert image_id of government id picture into the created user entry, linking them (this replaces NULL)
UPDATE User SET governmentid_image_id = @image_governid_id WHERE user_id = @user_id;

-- Insert profile image data into the Image table using example URL and retrieved user_id
INSERT INTO Image (uploaded_by_user_id, image_url)
VALUES (@user_id, "https://airbnb.com/images/profile_image1.jpg");
-- Retrieve the auto-generated image_id
SET @image_profile_id = LAST_INSERT_ID();

-- Insert into EmergencyContact table
INSERT INTO EmergencyContact (name, relationship, prefered_language_id, email, country_code, phone_number, owning_user_id)
VALUES ("Anna Mustermann", "Family", 4, "anna@example.com", "+49", "123456789", @user_id);
-- Retrieve the auto-generated emergencycontact_id
SET @emergencycontact_id = LAST_INSERT_ID();

-- Insert into Guest table using retreived user and image ids.
INSERT INTO Guest (user_id, profile_image_id, guest_description, phone_number)
VALUES (@user_id, @image_profile_id, "Hello i am Max, a 41 years old digital nomand. I love traveling...", "+49987654321");

-- Retrieve the auto-generated guest_id
SET @guest_id = LAST_INSERT_ID();

-- Insert into CreditCardInformation table
INSERT INTO CreditCardInformation (owning_guest_id, bank, card_number, cvc_code, expiration_date)
VALUES (@guest_id, "Sparda Bank", "3353422819762527", "123", "2026-10-31");

-- commit the transaction
COMMIT;
```

CREATE

```sql
CREATE TABLE Guest (
    guest_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNIQUE NOT NULL, /*FK*/
    profile_image_id INT UNIQUE, /*FK*/
    guest_description TEXT,
    phone_number VARCHAR(16) NOT NULL
);
```

```sql
/* Guest constraints */
ALTER TABLE Guest
ADD CONSTRAINT fk_guest_user
FOREIGN KEY (user_id)
REFERENCES User(user_id)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE Guest
ADD CONSTRAINT fk_guest_image
FOREIGN KEY (profile_image_id)
REFERENCES Image(image_id)
ON DELETE CASCADE ON UPDATE RESTRICT;
```

Test Case

```sql
/* relevant guest data
- it could be argued that this view should be based on the guest_id instead of the user_id but this
is an easy change and depends on how the view is to be used in practice
- as it is this view serves as a demonstration for how the connected data can be querried by comprosin
a complex select statement into a view for easier use.
*/
CREATE VIEW iu_userguest_view AS
SELECT
    U.user_id,
    U.legal_name,
    U.email,
    A.country,
    A.state,
    A.city,
    A.street,
    A.house_nr,
    A.zip_code,
    G.guest_description,
    G.phone_number,
    IProfile.image_url AS profile_image_url,
    IGovernID.image_url AS governmentid_image_url,
    EC.name AS emergency_contact_name,
    EC.relationship AS emergency_contact_relationship,
    EC.email AS emergency_contact_email,
    EC.country_code AS emergency_contact_country_code,
    EC.phone_number AS emergency_contact_phone_number,
    CCI.bank AS credit_card_bank,
    CCI.card_number AS credit_card_number,
    CCI.cvc_code AS credit_card_cvc,
    CCI.expiration_date AS credit_card_expiration_date
FROM
    User U
JOIN Address A ON U.address_id = A.address_id
JOIN Guest G ON U.user_id = G.user_id
JOIN Image IProfile ON G.profile_image_id = IProfile.image_id
JOIN Image IGovernID ON U.governmentid_image_id = IGovernID.image_id
LEFT JOIN EmergencyContact EC ON U.user_id = EC.owning_user_id
LEFT JOIN CreditCardInformation CCI ON G.guest_id = CCI.owning_guest_id;

-- usage of view
SELECT * FROM iu_userguest_view WHERE user_id = 1;
```

Guest	
PK	guest_id
FK	*user_id*
FK	*profile_image_id*
	guest_description
	phone_number

```
mysql> SELECT * FROM iu_userguest_view WHERE user_id = 1;
| user_id | legal_name    | email                     | country | state  | city      | street       | house_nr | zip_code | guest_description                                                  | phone_number  | profile_image_url                            | governmentid_imag
e_url                          | emergency_contact_name | emergency_contact_relationship | emergency_contact_email | emergency_contact_country_code | emergency_contact_phone_number | credit_card_bank | credit_card_number | credit_card_cvc | credit_card_expiration_date |
|       1 | Max Musterman | max.musterman@example.com | Germany | Hessen | Frankfurt | Musterstraße |        1 | 60306    | Hello i am Max, a 41 years old digital nomand. I love traveling... | +49987654321 | https://airbnb.com/images/profile_image1.jpg | https://airbnb.co
m/images/governmentid_image1.jpg | Anna Mustermann        | Family                         | anna@example.com        | +49                            | 123456789                      | Sparda Bank      | 3353422819762527   | 123             | 2026-10-31                  |
1 row in set (0.00 sec)
```

TABLE - HOST

```
mysql> SELECT COUNT(*) FROM Host;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

- The ‘user_id’ attribute refers back to the base class.
- ‘profile_image_id’ references the id of an image that can be loaded to display the users profile image.
  - The reasoning for having this attribute in both subclasses is that it is only required, or ‘NOT NULL’ for Hosts.
- The test case is very similar to that of the previous guest class, adjusting where relevant relationships change compared to the previous slide. This should ensure that all the data for single user that is stored in multiple tables is properly connected.

INSERT

```
/*
User, Host, Image, BankInformation, Address (SQL Transaction)

the structure of the host insert statements is very similar to the guest as they share a lot of similaritities

* i will only comment the first transaction as the rest follow the same schematic
*/
-- Host 1
START TRANSACTION;

/* Address for the Host, saved for later use */
INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
VALUES ("United Kingdom", "England", "London", "123 Main St", 123, "SW1A 1AA", "host");
SET @address_id = LAST_INSERT_ID();

/* User (super) class of the Host, saved for later use  */
INSERT INTO User (address_id, email, legal_name)
VALUES (@address_id, "william.turner@hotmail.com", "William Turner");
SET @user_id = LAST_INSERT_ID();

/* Profile Image of the Host, saved for later use */
INSERT INTO Image (uploaded_by_user_id, image_url)
VALUES (@user_id, "https://airbnb.com/images/governmentid_image21.jpg");
SET @image_governmentid_id = LAST_INSERT_ID();
UPDATE User SET governmentid_image_id = @image_governmentid_id WHERE user_id = @user_id;

/* government id image of the Host (User) */
INSERT INTO Image (uploaded_by_user_id, image_url)
VALUES (@user_id, "https://airbnb.com/images/profile_image21.jpg");
SET @image_profile_id = LAST_INSERT_ID();

/* Host data, using the previously inserted data */
INSERT INTO Host (user_id, profile_image_id, host_description, phone_number)
VALUES (@user_id, @image_profile_id, "Host description for host 1", "9876123469");
SET @host_id = LAST_INSERT_ID();

/* Bank Address - this information should likely be inserted differently in an actual production environment, but this works for now */
INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
VALUES ("United Kingdom", "England", "London", "456 Oak St", 456, "SW1A 1AA", "bank");
SET @bankaddress_id = LAST_INSERT_ID();

/* BankInformaion data, using the previously inserted address data */
INSERT INTO BankInformation (owning_host_id, name, account_nr, code, address_id)
VALUES (@host_id, "Barclays Bank plc", "9876543229", "123", @bankaddress_id);

COMMIT;
```

CREATE

```
CREATE TABLE Host (
    host_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNIQUE NOT NULL, /*FK*/
    profile_image_id INT UNIQUE NOT NULL, /*FK*/
    host_description TEXT NOT NULL,
    phone_number VARCHAR(16)
);

/* Host constraints */
ALTER TABLE Host
ADD CONSTRAINT fk_host_user
FOREIGN KEY (user_id)
REFERENCES User(user_id)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE Host
ADD CONSTRAINT fk_host_image
FOREIGN KEY (profile_image_id)
REFERENCES Image(image_id)
ON DELETE CASCADE ON UPDATE RESTRICT;
```

Test Case

```
/* relevant host data
- it could be argued that this view should be based on the host_id instead of the user_id but this is an easy change and depends on how the view is to be used in practice
- as it is this view serves as a demonstration for how the connected data can be querried by comprosing a complex select statement into a view for easier use.
*/
CREATE VIEW iu_userhost_view AS
SELECT
    U.user_id,
    U.legal_name,
    U.email,
    A.country,
    A.state,
    A.city,
    A.street,
    A.house_nr,
    A.zip_code,
    H.host_description,
    H.phone_number,
    IProfile.image_url AS profile_image_url,
    IGovernID.image_url AS governmentid_image_url,
    BI.account_nr AS bank_account_nr,
    BI.code AS bank_code,
    BI.address_id AS bank_address_id,
    BIA.country AS bank_address_country
FROM
    User U
JOIN Address A ON U.address_id = A.address_id
JOIN Host H ON U.user_id = H.user_id
JOIN Image IProfile ON H.profile_image_id = IProfile.image_id
JOIN Image IGovernID ON U.governmentid_image_id = IGovernID.image_id
LEFT JOIN BankInformation BI ON H.host_id = BI.owning_host_id
JOIN Address BIA ON BI.address_id = BIA.address_id;

-- usage example of view
SELECT * FROM iu_userhost_view WHERE user_id = 21;
```

Host	
PK	host_id
FK	user_id
FK	profile_image_id
	host_description
	phone_number

```
mysql> SELECT * FROM iu_userhost_view WHERE user_id = 21;
+---------+-------------+-------------------+---------+---------+---------+-------------+----------+----------+
| user_id | legal_name  | email             | country | state   | city    | street      | house_nr | zip_code |
| host_description | phone_number | profile_image_url | governmentid_image_url |
| bank_account_nr | bank_code | bank_address_id | bank_address_country |
+---------+-------------+-------------------+---------+---------+---------+-------------+----------+----------+
|      21 | William Turner | william.turner@hotmail.com | United Kingdom | England | London | 123 Main St |      123 | SW1A 1AA |
| Host description for host 1 | 9876123469 | https://airbnb.com/images/profile_image21.jpg | https://airbnb.com/images/governmen
tid_image21.jpg | 9876543229 | 123 | 22 | United Kingdom |
+---------+-------------+-------------------+---------+---------+---------+-------------+----------+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM UserReview;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE - USERREVIEW

INSERT

```sql
/* UserReview
- The UserReview is written by a host to the profile of a guest who stayed at their property
- The author therefore has to be a host, while the user the review is written to has to be a guest
- There are no other restrictions except that both id's have to be valid
*/
-- UserReview 1
INSERT INTO UserReview (user_id, author_user_id, comment)
VALUES (1, 21, "They were a lovely guest, we hope to meet you again some time!");
```

CREATE

```sql
CREATE TABLE UserReview (
    userreview_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL, /*FK*/
    author_user_id INT NOT NULL, /*FK*/
    comment VARCHAR(2000),
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (author_user_id) REFERENCES User(user_id)
);
```

- The ‘UserReview’ table holds all the reviews given on users.
- These are comments left by hosts on the guest pages and are different from reviews left by guests on the property listing.
- The comment attribute holds the user created written text.
- User and author_user ids and timestamps are also saved.
- The test case tests the content and relationship of user reviews and users by returning all data entries of user reviews for a given user.

Test Case

```sql
/* UserReview data from related to explicit user
- quite simple as the userreview shares the attribute 'user_id' which can be used for a natural join,
otherwise a left join on user_id could be used when only specific data is desired
*/
CREATE VIEW iu_userreviews_from_user_view AS
SELECT *
FROM User
NATURAL JOIN UserReview;
/* another test example for only the UserReview data without User data:
SELECT UR.*
FROM User
NATURAL JOIN UserReview UR;
*/

-- usage example of view
SELECT * FROM iu_userreviews_from_user_view WHERE user_id = 1;
```

UserReview	
PK	userreview_id
FK	user_id
FK	author_user_id
	comment
	created
	last_modified

```
mysql> SELECT * FROM iu_userreviews_from_user_view WHERE user_id = 1;
+---------+------------+-------------------------+-------------+-----------------------+----------------+----------------+-----------------------------------------------------------+---------------------+---------------------+
| user_id | address_id | email                   | legal_name  | governmentid_image_id | userreview_id | author_user_id | comment                                                   | created             | last_modified       |
+---------+------------+-------------------------+-------------+-----------------------+----------------+----------------+-----------------------------------------------------------+---------------------+---------------------+
|       1 |          1 | max.musterman@example.com | Max Musterman |                     1 |              1 |             21 | They were a lovely guest, we hope to meet you again some time! | 2024-01-08 10:03:57 | 2024-01-08 10:03:57 |
|       1 |          1 | max.musterman@example.com | Max Musterman |                     1 |              2 |             23 | There were no problems, and they left the property clean and in order. | 2024-01-08 10:03:57 | 2024-01-08 10:03:57 |
|       1 |          1 | max.musterman@example.com | Max Musterman |                     1 |              3 |             24 | Thanks for your stay!                                     | 2024-01-08 10:03:57 | 2024-01-08 10:03:57 |
+---------+------------+-------------------------+-------------+-----------------------+----------------+----------------+-----------------------------------------------------------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Address;
+----------+
| COUNT(*) |
+----------+
|       80 |
+----------+
1 row in set (0.02 sec)
```

COUNT

TABLE - ADDRESS

CREATE

```sql
CREATE TABLE Address (
    address_id INT AUTO_INCREMENT PRIMARY KEY,
    country VARCHAR(128) NOT NULL,
    state VARCHAR(128) NOT NULL,
    city VARCHAR(128) NOT NULL,
    street VARCHAR(128) NOT NULL,
    house_nr INT NOT NULL,
    zip_code VARCHAR(10) NOT NULL,
    address_type VARCHAR(32)
);
```

Address	
PK	address_id
	country
	state
	city
	street
	house_nr
	zip_code
	address_type

Tested and inserted via multiple other test cases.
Please refer to one of the user classes for an example

- The address attributes themselves are self-explanatory in meaning.
- The reasoning for the selection of attributes is the categorization or search user flow on the AirBnB website.
- An address can be used by either users, properties (listings) or banks.
- This table is tested is a part of many other tables and is therefore already tested more than enough. There is still a simple select all test to check for completeness of content in the test.sql file.

```
mysql> SELECT COUNT(*) FROM Image;
+----------+
| COUNT(*) |
+----------+
|       85 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE - IMAGE

CREATE

```sql
CREATE TABLE Image (
    image_id INT AUTO_INCREMENT PRIMARY KEY,
    uploaded_by_user_id INT NOT NULL, /*FK*/
    image_url varchar(1024) NOT NULL,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (uploaded_by_user_id) REFERENCES User(user_id)
);
```

Image	
PK	image_id
FK	uploaded_by_user_id
	image_url
	created

Tested and inserted via multiple other test cases.
Please refer to one of the user classes for an example

- This table assumes that the images themselves are stored by a cloud storage provider for example.
- This means that the attribute itself can be of type VARCHAR instead of having to save the image as a BLOB, which is inefficient.
- The Table also stores the id of the user that uploaded the image.
- The ‘created’ attribute is defaulted to the current timestamp, meaning it shows the time the image was uploaded.
- Like the ‘Address’ table, the ‘Image’ table is also part of many other tables, meaning that the relationships of this table are already thoroughly tested. There is, again, still a select all test to check for completeness of content.

```
mysql> SELECT COUNT(*) FROM Currency;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.02 sec)
```

COUNT

TABLE - CURRENCY

CREATE

```sql
CREATE TABLE Currency (
    currency_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(64) NOT NULL,
    abbreviation VARCHAR(3)
);
```

Currency	
PK	currency_id
	name
	abbreviation

INSERT

```sql
INSERT INTO Currency (name, abbreviation) VALUES ("United States dollar", "USD");
```

'Currency' is a simple table, see test case in test.sql

- The currency table represents all currencies available in the application.
- Name represents the name of the currency while country represents the country it is used in.
- Abbreviation is used for display purposes.
- Currency is not a table that will see frequent changes and is rather used as a reference or lookup table.
- This table is very simple and does not require complicated testing, a simple select all test can be found in the test.sql file.

```
mysql> SELECT COUNT(*) FROM Language;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE - LANGUAGE

CREATE

```
195  CREATE TABLE Language (
196      language_id INT AUTO_INCREMENT PRIMARY KEY,
197      name VARCHAR(32) NOT NULL,
198      country VARCHAR(32)
199  );
```

Language	
PK	language_id
	name
	country

INSERT

```
19   INSERT INTO Language (name, country) VALUES ("English", "United States");
```

'Language' is a simple table, see test case in test.sql

- The language table represents all languages available in the application.
- Name represents the name of the language while country represents the country it is used in.
- The reason for this is that Airbnb differentiates between, for example, American and British English.
- Language is not a table that will see frequent changes and is rather used as a reference or look up table.
- Like the 'Currency', this table is also very simple, a test for content should suffice for this table, which can be found in the test.sql file.

TABLE - CHAT

COUNT

```
mysql> SELECT COUNT(*) FROM Chat;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

- A chat is a collection of messages, in the next slide we will inspect the 'message' table.
- These messages are linked to a chat by sharing the same chat id.
- Other than the 'created' and 'last_modified' timestamps, the chats table also saves both chat participants.
- The test case for this table works in combination with the message table (next slide). The first test case checks the general content of the chat table.

INSERT

```sql
/* Chat
- the chat table is hard to make sense of without the frontend, the chat table holds the ids of the two users chatting,
the owner being the initiator of the chat
- the id's are referenced by the messages to string together the whole chat
- important is that the user id's have to be valid and they cannot be identical, as a user can't chat with themselves
- the partner/owner naming is just for context and does not have to be regarded when requesting a chat
(otherwise duplicates would be needed, which makes no sense)
*/
INSERT INTO Chat (chat_partner_id, owner_user_id) VALUES (1, 2);
```

CREATE

```sql
CREATE TABLE Chat (
    chat_id INT AUTO_INCREMENT PRIMARY KEY,
    chat_partner_id INT, /*FK*/
    owner_user_id INT, /*FK*/
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```sql
/* Chat constraints */
ALTER TABLE Chat
ADD CONSTRAINT fk_chat_partner
FOREIGN KEY (chat_partner_id)
REFERENCES User(user_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Chat
ADD CONSTRAINT fk_chat_owner
FOREIGN KEY (owner_user_id)
REFERENCES User(user_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Test Case

```sql
/* Chat & Message data
- two examples, the first one will display general details of the chat, the second text will
return all messages of a given chat
*/
CREATE VIEW iu_chat_details_view AS
SELECT
    C.*,
    UO.legal_name AS owning_user_name,
    UP.legal_name AS partner_guest_name,
    COUNT(M.message_id) AS message_count
FROM Chat C
JOIN User UO ON C.owner_user_id = UO.user_id
JOIN User UP ON C.chat_partner_id = UP.user_id
LEFT JOIN Message M ON M.owning_chat_id = C.chat_id
GROUP BY C.chat_id, UO.legal_name, UP.legal_name;

-- usage example of view
SELECT * FROM iu_chat_details_view WHERE chat_id = 1;

CREATE VIEW iu_chat_messages_view AS
SELECT
    M.message_id,
    M.text,
    M.image_id,
    C.chat_id AS owning_chat_id_ref
FROM Message M
JOIN Chat C ON C.chat_id = M.owning_chat_id;

-- usage example of view
SELECT * FROM iu_chat_messages_view WHERE owning_chat_id_ref = 1;
```

Chat	
PK	chat_id
FK	chat_partner_id
FK	owner_user_id
	created
	last_modified

```
mysql> SELECT * FROM iu_chat_details_view WHERE chat_id = 1;
+---------+-----------------+---------------+---------------------+---------------------+-----------------+--------------------+---------------+
| chat_id | chat_partner_id | owner_user_id | created             | last_modified       | owning_user_name | partner_guest_name | message_count |
+---------+-----------------+---------------+---------------------+---------------------+-----------------+--------------------+---------------+
|       1 |               1 |             2 | 2024-01-08 10:03:57 | 2024-01-08 10:03:57 | John Doe        | Max Musterman      |             1 |
+---------+-----------------+---------------+---------------------+---------------------+-----------------+--------------------+---------------+
1 row in set (0.00 sec)
```

TABLE - MESSAGE

```
mysql> SELECT COUNT(*) FROM Message;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

- The message is linked to the chat via the chat id, represented here as 'owning_chat_id'. The 'author_user_id' attribute holds the author; Both id's can be used to reconstruct a chat.
- As Airbnb enables users to share images in chats, a message may contain an image instead of text. This necessitates the optional 'image_id' attribute.
- The created timestamp is essentially the 'sent' timestamp of the message.
- The second test case in the screenshot tests the relationship of 'Chat' and 'Message' tables by returning all data entries of messages for a given Chat.

INSERT

```
/* Message
- the messages hold the individual text of a message and their owning user, they reference the owning chat,
- their timestamp is used to recreate the chat when requried
*/
INSERT INTO Message (text, author_user_id, owning_chat_id) VALUES ("Content for message 1", 1, 1);
```

```
/* examples with images - the same user_id must be used for the image and the chat message */
INSERT INTO Image (uploaded_by_user_id, image_url)
VALUES (1, "https://airbnb.com/images/chat_message_image1.jpg");
SET @message_image_id = LAST_INSERT_ID();
INSERT INTO Message (image_id, author_user_id, owning_chat_id) VALUES (@message_image_id, 9, 12);
```

CREATE

```
CREATE TABLE Message (
    message_id INT AUTO_INCREMENT PRIMARY KEY,
    text TEXT,
    image_id INT, /*FK*/
    author_user_id INT, /*FK*/
    owning_chat_id INT, /*FK*/
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```
/* Message constraints */
ALTER TABLE Message
ADD CONSTRAINT fk_message_owningchat
FOREIGN KEY (owning_chat_id)
REFERENCES Chat(chat_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Test Case

```
/* Chat & Message data
- two examples, the first one will display general details of the chat, the second text will
return all messages of a given chat
*/
CREATE VIEW iu_chat_details_view AS
SELECT
    C.*,
    UO.legal_name AS owning_user_name,
    UP.legal_name AS partner_guest_name,
    COUNT(M.message_id) AS message_count
FROM Chat C
JOIN User UO ON C.owner_user_id = UO.user_id
JOIN User UP ON C.chat_partner_id = UP.user_id
LEFT JOIN Message M ON M.owning_chat_id = C.chat_id
GROUP BY C.chat_id, UO.legal_name, UP.legal_name;

-- usage example of view
SELECT * FROM iu_chat_details_view WHERE chat_id = 1;

CREATE VIEW iu_chat_messages_view AS
SELECT
    M.message_id,
    M.text,
    M.image_id,
    C.chat_id AS owning_chat_id_ref
FROM Message M
JOIN Chat C ON C.chat_id = M.owning_chat_id;

-- usage example of view
SELECT * FROM iu_chat_messages_view WHERE owning_chat_id_ref = 1;
```

Message	
PK	message_id
	text
	created
FK	image_id
FK	author_user_id
FK	owning_chat_id

```
mysql> SELECT * FROM iu_chat_messages_view WHERE owning_chat_id_ref = 1;
+------------+----------------------+----------+--------------------+
| message_id | text                 | image_id | owning_chat_id_ref |
+------------+----------------------+----------+--------------------+
|          1 | Content for message 1 |     NULL |                  1 |
+------------+----------------------+----------+--------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM EmergencyContact;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE - EMERGENCYCONTACT

- The emergency contact is different to a user as it is only a collection of information relevant to the contact without any of the functionality of an actual account in the application.
- All the information regarding the contact should be self-explanatory.
- The 'owning_user_id' refers to the user that "owns" the entry.
- This table is covered in the Guest user test cases, another simple select all statement is provided in the test.sql file to check for completeness of content.

CREATE

```sql
CREATE TABLE EmergencyContact (
    emergencycontact_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    relationship VARCHAR(255),
    prefered_language_id INT DEFAULT 1, /*FK*/
    email VARCHAR(255),
    country_code VARCHAR (32),
    phone_number VARCHAR(15) NOT NULL,
    owning_user_id INT NOT NULL /*FK*/

/* EmergencyContact constraints */
ALTER TABLE EmergencyContact
ADD CONSTRAINT fk_econtact_language
FOREIGN KEY (prefered_language_id)
REFERENCES Language(language_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE EmergencyContact
ADD CONSTRAINT fk_econtact_user
FOREIGN KEY (owning_user_id)
REFERENCES User(user_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

EmergencyContact	
PK	emergencycontact_id
	name
	relationship
FK	prefered_language_id
	email
	country_code
	phone_number
FK	owning_user_id

INSERT

```sql
-- Insert into EmergencyContact table
INSERT INTO EmergencyContact (name, relationship, prefered_language_id, email, country_code, phone_number, owning_user_id)
VALUES ("Anna Mustermann", "Family", 4, "anna@example.com", "+49", "123456789", @user_id);
-- Retrieve the auto-generated emergencycontact_id
SET @emergencycontact_id = LAST_INSERT_ID();
```

'Language' is a simple table that gets added to during the guest transaction.

TABLE - WISHLIST

COUNT

```
mysql> SELECT COUNT(*) FROM Wishlist;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

- Wishlists in the Airbnb application are a collection of property listings.
- As this is a M:N relation it needs to be normalized. We achieve this by using this 'wishlist' and a 'wishlist_propertylisting' table (see next slide).
- A user can have multiple Wishlists, hence the need for a name.
- The 'wishlist_id' is used in the next table to normalize the M:N relation.
- The table works in close relation to the 'wishlist_propertylisting' table (next slide). This first test case checks for general data of the wishlist table and other related tables in conjunction. The second test case checks the relation of a wishlist and its propertylistings.

CREATE

```sql
CREATE TABLE Wishlist (
    wishlist_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    private BOOLEAN NOT NULL DEFAULT TRUE,
    owning_user_id INT NOT NULL /*FK*/
);
```

```sql
/* Wishlist constraints */
ALTER TABLE Wishlist
ADD CONSTRAINT fk_wishlist_guest
FOREIGN KEY (owning_user_id)
REFERENCES Guest(guest_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

```sql
/* Wishlist
- the wishlist is the relation that stores the data relevant to the wishlists
- the wishlist is referenced by the wishlist_propertylisting relation to link the properties in the list with the list itself
- similar to the chat/messages, it is hard to visualize the wishlist tables without the frontend
*/
INSERT INTO Wishlist (name, private, owning_user_id) VALUES ("wishlist 1", TRUE, 1);
```

Test Case

```sql
/* Wishlist data
- test case that shows the wishlist, user and property listing data to proves the proper
implementation of links between them. Goal is to see the relationship of a user owning
a wishlist, which in turn 'owns' (multiple) propertylistings.
*/
CREATE VIEW iu_wishlist_details_view AS
SELECT
    W.*,
    U.legal_name AS owning_user_name,
    PL.name AS property_listing_namne
FROM
    Wishlist W
JOIN User U ON W.owning_user_id = U.user_id
JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;

-- usage example of view
SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;

-- this view gets all data regarding the property listings that are in a given wishlist
CREATE VIEW iu_wishlist_propertylistings_view AS
SELECT
    W.wishlist_id,
    PL.*
FROM Wishlist W
JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;

-- usage example of view
SELECT * FROM iu_wishlist_propertylistings_view WHERE wishlist_id = 1;
```

Wishlist	
PK	wishlist_id
	name
	private
FK	owning_user_id

```
mysql> SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;
+-------------+------------+---------+---------------+-----------------+--------------------------+
| wishlist_id | name       | private | owning_user_id | owning_user_name | property_listing_namne   |
+-------------+------------+---------+---------------+-----------------+--------------------------+
|           1 | wishlist 1 |       1 |             1 | Max Musterman   | Cozy Studio Apartment    |
|           1 | wishlist 1 |       1 |             1 | Max Musterman   | Cozy Studio Apartment    |
|           1 | wishlist 1 |       1 |             1 | Max Musterman   | Spacious Loft in the City |
|           1 | wishlist 1 |       1 |             1 | Max Musterman   | Luxurious Beachfront Villa |
|           1 | wishlist 1 |       1 |             1 | Max Musterman   | Mountain Retreat Cabin   |
+-------------+------------+---------+---------------+-----------------+--------------------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Wishlist_Propertylisting;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE – WISHLIST_PROPERTYLISTING

- This table, as mentioned, is used to normalize the wishlist – propertylisting relation.
- The table matches Propertylistings to Wishlists using the two foreign key ids.
- The propertylisting table will be introduced in the following slides.
- As explained in the previous slide, this table stands in close relation to the wishlist table. The second test case demonstrates the relationship of a wishlist with the propertylisting table via the link of this table very well.

CREATE

```sql
CREATE TABLE Wishlist_PropertyListing (
    wishlist_propertylisting_id INT AUTO_INCREMENT PRIMARY KEY,
    propertylisting_id INT, /*FK*/
    wishlist_id INT /*FK*/
);
```

```sql
/* Wishlist_PropertyListing constraints */
ALTER TABLE Wishlist_PropertyListing
ADD CONSTRAINT fk_wishlist_property
FOREIGN KEY (propertylisting_id)
REFERENCES PropertyListing(propertylisting_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Wishlist_PropertyListing
ADD CONSTRAINT fk_wishlist_wishlist
FOREIGN KEY (wishlist_id)
REFERENCES Wishlist(wishlist_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

```sql
/* Wishlist_propertylisting
- this is a relation table that links the wishlists and the properties together
- the wishlist_id references the previous Wishlist relation
*/
INSERT INTO Wishlist_PropertyListing (propertylisting_id, wishlist_id) VALUES (1, 1);
```

Test Case

```sql
/* Wishlist data
- test case that shows the wishlist, user and property listing data to proves the proper
implementation of links between them. Goal is to see the relationship of a user owning
a wishlist, which in turn 'owns' (multiple) propertylistings.
*/
CREATE VIEW iu_wishlist_details_view AS
SELECT
    W.*,
    U.legal_name AS owning_user_name,
    PL.name AS property_listing_namne
FROM
    Wishlist W
JOIN User U ON W.owning_user_id = U.user_id
JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;

-- usage example of view
SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;

-- this view gets all data regarding the property listings that are in a given wishlist
CREATE VIEW iu_wishlist_propertylistings_view AS
SELECT
    W.wishlist_id,
    PL.*
FROM Wishlist W
JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;

-- usage example of view
SELECT * FROM iu_wishlist_propertylistings_view WHERE wishlist_id = 1;
```

Wishlist_PropertyListing	
PK	wishlist_propertylisting_id
FK	*propertylisting_id*
FK	*wishlist_id*

```
mysql> SELECT * FROM iu_wishlist_propertylistings_view WHERE wishlist_id = 1;
+-------------+---------------------+---------------+---------------------------+------------+------------------+---------------+----------------+-------------------------+-----------------+-------------+---------------------+---------------------+----------------+-------------------------+
| wishlist_id | propertylisting_id  | owning_host_id | name                      | address_id | property_type_id | bedroom_count | bathroom_count | available_booking_slots | price_per_night | currency_id | check_in_time       | check_out_time      | average_rating | total_nr_of_ratings     |
+-------------+---------------------+---------------+---------------------------+------------+------------------+---------------+----------------+-------------------------+-----------------+-------------+---------------------+---------------------+----------------+-------------------------+
|           1 |                   1 |          NULL | Cozy Studio Apartment     |         61 |                1 |             1 |              1 |                       2 |           50.00 |           1 | 15:00:00            | 11:00:00            |            0.0 |                       0 |
|           1 |                   2 |          NULL | Cozy Studio Apartment     |         62 |                1 |             1 |              1 |                       2 |           50.00 |           1 | 15:00:00            | 11:00:00            |            0.0 |                       0 |
|           1 |                   3 |          NULL | Spacious Loft in the City |         63 |                2 |             2 |              1 |                       4 |          120.00 |           1 | 16:00:00            | 10:00:00            |            0.0 |                       0 |
|           1 |                   4 |          NULL | Luxurious Beachfront Villa |         64 |                3 |             4 |              3 |                       6 |          300.00 |           1 | 14:00:00            | 12:00:00            |            0.0 |                       0 |
|           1 |                   5 |          NULL | Mountain Retreat Cabin    |         65 |                4 |             2 |              1 |                       3 |           80.00 |           1 | 12:00:00            | 10:00:00            |            0.0 |                       0 |
+-------------+---------------------+---------------+---------------------------+------------+------------------+---------------+----------------+-------------------------+-----------------+-------------+---------------------+---------------------+----------------+-------------------------+
5 rows in set (0.00 sec)
```

TABLE - PROPERTYLISTING

```
mysql> SELECT COUNT(*) FROM Propertylisting;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)
```

COUNT

- The ‘PropertyListing’ table marks the second important ‘block’ of data mentioned in the introduction.
- This table holds all relevant information for the listings and has multiple M:N relations that are not shown in the table itself.
- These relations rely on the ‘propertylisting_id’ and are introduced in the following slides.
- Although many of the relations of this table have already been tested, this table is of great importance to the overall system. It is therefore reasonable to create a test case that generally tests all relationships of this table. Said test case can be seen here, it focuses on returning relevant data from all tables that have a relationship with the propertylisting table. (Either via property or host id)

CREATE

```sql
CREATE TABLE PropertyListing (
    propertylisting_id INT AUTO_INCREMENT PRIMARY KEY,
    owning_host_id INT NOT NULL, /*FK*/
    name VARCHAR(255) NOT NULL,
    address_id INT UNIQUE NOT NULL, /*FK*/
    property_type_id INT, /*FK*/
    bedroom_count INT,
    bathroom_count INT,
    available_booking_slots INT,
    price_per_night DECIMAL(10,2),
    currency_id INT NOT NULL DEFAULT 1, /*FK*/
    check_in_time TIME DEFAULT '14:00',
    check_out_time TIME DEFAULT '10:00',
    average_rating DECIMAL(2,1) DEFAULT 0, /* should get updated by the applciation when a new review is published, is initially 0 */
    total_nr_of_ratings INT DEFAULT 0, /* should get updated/incremented by the applciation when a new review is published, is initially 0 */

    FOREIGN KEY (owning_host_id) REFERENCES Host(host_id)
);
```

```sql
/* PropertyListing constraints */
ALTER TABLE PropertyListing
ADD CONSTRAINT fk_listing_address
FOREIGN KEY (address_id)
REFERENCES Address(address_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE PropertyListing
ADD CONSTRAINT fk_listing_currency
FOREIGN KEY (currency_id)
REFERENCES Currency(currency_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

```sql
/* PropertyListing
- the propertylisting is referenced by the previously created tables for the relevant categoy, amenities, and houserules
- the structure of the inserts is fairly simple, we first insert the address data and then use the insert id for the propertylisting
- for the other attributes we simlpy enter semi-random dummy data for testing/demonstration purposes
*/
-- PropertyListing 1
INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
VALUES ("United States", "New York", "Brooklyn", "Elm Street", "1234", "11201", "property");
SET @property_address_id = LAST_INSERT_ID();
INSERT INTO PropertyListing (owning_host_id, name, address_id, property_type_id, bedroom_count, bathroom_count, available_booking_slots, price_per_night, currency_id, check_in_time, check_out_time)
VALUES (1, "Cozy Studio Apartment", @property_address_id, 1, 1, 1, 2, 50.00, 1, "15:00:00", "11:00:00");
```

Test Case

```sql
/* relevant property listing data
- this first view shows the data attributes that somewhat directly relate to the propertylisting while
the the following view will serve as an example for one of the tables that is linked via another table.
In that case we will use the amenitites table to demonstrate the proper relationship between a
propertylisting and amenities, categories and houserules.
- includes property_type relation test
*/
CREATE VIEW iu_propertylisting_view AS
SELECT
    PL.propertylisting_id,
    PL.name AS propertylisting_name,
    PT.name AS property_type,
    PL.price_per_night,
    C.name AS currency,
    A.country,
    A.state,
    A.city,
    A.street,
    A.house_nr,
    A.zip_code,
    H.host_id AS host_id,
    HU.legal_name AS host_name
FROM
    PropertyListing PL
JOIN Address A ON PL.address_id = A.address_id
JOIN Host H ON PL.owning_host_id = H.host_id
JOIN User HU ON H.user_id = HU.user_id
JOIN Currency C ON PL.currency_id = C.currency_id
JOIN PropertyType PT ON PL.property_type_id = PT.propertytype_id;

-- usage example of view
SELECT * FROM iu_propertylisting_view WHERE propertylisting_id = 1;
SELECT * FROM iu_propertylisting_view WHERE host_id = 1;
```

PropertyListing	
PK	propertylisting_id
FK	owning_host_id
	name
FK	address_id
FK	property_type_id
	bedroom_count
	bathroom_count
	available_booking_slots
	price_per_night
	cleaning_fee
FK	currency_id
	check_in_time
	check_out_time
	average_rating
	total_nr_of_ratings

```
mysql> SELECT * FROM iu_propertylisting_view WHERE host_id = 1;
+---------------------+---------------------+---------------+--------------+-----------------------+---------------+------------+-------------+---------------+----------+----------+---------+-----------------+
| propertylisting_id  | propertylisting_name| property_type | price_per_night | currency           | country       | state      | city        | street        | house_nr | zip_code | host_id | host_name       |
+---------------------+---------------------+---------------+--------------+-----------------------+---------------+------------+-------------+---------------+----------+----------+---------+-----------------+
|                   1 | Cozy Studio Apartment | Entire place  |        50.00 | United States dollar  | United States | New York   | Brooklyn    | Elm Street    |     1234 |    11201 |       1 | William Turner  |
|                   2 | Tranquil Retreat Cottage | Entire place |        50.00 | United States dollar  | United States | California | San Francisco | Oak Avenue  |     5678 |    94110 |       1 | William Turner  |
|                   3 | Spacious Loft in the City | Private room |       120.00 | United States dollar  | United States | California | Los Angeles | Maple Drive   |      910 |    90046 |       1 | William Turner  |
|                   4 | Luxurious Beachfront Villa | Hotel room  |       300.00 | United States dollar  | United States | Illinois   | Chicago     | Pine Lane     |      123 |    60611 |       1 | William Turner  |
+---------------------+---------------------+---------------+--------------+-----------------------+---------------+------------+-------------+---------------+----------+----------+---------+-----------------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Propertyreview;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE - PROPERTYREVIEW

- Similar to a User Review this table holds the reviews given by guests to properties.
- It holds multiple ratings/scores which are displayed on the listing page. These are represented as integers with values from 0 to 5. (Star rating). The application should make sure that the input is an integer within the given range before sending it to the DBMS.
- Users can also add comments to the review.
- The test case simple checks the relationship between propertylistings and reviews, as well as the review content.

CREATE

```sql
CREATE TABLE PropertyReview (
    propertyreview_id INT AUTO_INCREMENT PRIMARY KEY,
    property_id INT NOT NULL, /*FK*/
    author_guest_id INT NOT NULL, /*FK*/
    cleanliness_score INT,
    accuracy_score INT,
    check_in_score INT,
    communication_score INT,
    location_score INT,
    value_score INT,
    comment VARCHAR(2000),
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```sql
/* PropertyReview constraints */
ALTER TABLE PropertyReview
ADD CONSTRAINT fk_review_property
FOREIGN KEY (property_id)
REFERENCES PropertyListing(propertylisting_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE PropertyReview
ADD CONSTRAINT fk_review_author
FOREIGN KEY (author_guest_id)
REFERENCES Guest(guest_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Test Case

```sql
/* PropertyReviews are very similar to the UserReviews above, i will therefore not go into much detail
about the structure of the query / test case
- the reason i constrained the query is that it would otherwise be very bloated with less readability
*/
CREATE VIEW iu_propertyreviews_view AS
SELECT
    PL.propertylisting_id, -- left in to show that PL.id and PR. id are equal -> functioning relation
    PL.name,
    PR.*
FROM PropertyListing PL
JOIN PropertyReview PR ON PL.propertylisting_id = PR.property_id;

-- usage example of view
SELECT * FROM iu_propertyreviews_view WHERE propertylisting_id = 1;
```

INSERT

```sql
/* Property Review
- the propertyreview table is very simple, in that it only contains two foreign keys, which are easy to explain
- the property_id is the property the review is about, while the author_guest_id is the guest who published the review
- the other attributes are simple and obvious
*/
-- PropertyReview 1
INSERT INTO PropertyReview (property_id, author_guest_id, cleanliness_score, accuracy_score, check_in_score, communication_score, location_score, value_score, comment)
VALUES (1, 1, 4, 4, 5, 4, 5, 4, "We had a wonderful stay at this place!");
```

PropertyReview	
PK	propertyreview_id
FK	property_id
FK	author_guest_id
	cleanliness_score
	accuracy_score
	check_in_score
	communication_score
	location_score
	value_score
	comment
	created

```
mysql> SELECT * FROM iu_propertyreviews_view WHERE propertylisting_id = 1;
+--------------------+------------------+--------------------+-------------+----------------+-------------------+----------------+----------------+---------------------+----------------+-------------+-----------------------------+---------------------+
| propertylisting_id | name             | propertyreview_id  | property_id | author_guest_id | cleanliness_score | accuracy_score | check_in_score | communication_score | location_score | value_score | comment                     | created             |
+--------------------+------------------+--------------------+-------------+----------------+-------------------+----------------+----------------+---------------------+----------------+-------------+-----------------------------+---------------------+
|                  1 | Cozy Studio Apartment |                  1 |           1 |              1 |                 4 |              4 |              5 |                   4 |              5 |           4 | We had a wonderful stay at this place! | 2024-01-18 20:59:43 |
+--------------------+------------------+--------------------+-------------+----------------+-------------------+----------------+----------------+---------------------+----------------+-------------+-----------------------------+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Property_Amenity;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE – PROPERTY_X

- Each table is used to normalize a relation between the PropertyListing and their respective table. I see these tables as one equivalence class.
- The naming convention of these tables is meant to represent the link between 'PropertyListing' and 'Amenity', 'Category' or 'HouseRule' table.
- The individual tables they are related to will be introduced in the following slide.
- These table are used to link other tables and hence do not necessarily need an id attribute.
  (I am considering removing these ids for the finalization phase and would appreciate feedback regarding this)
- The test case checks the link of the propertylisting to the respective table via the link of these tables by returning data of each data entry.

*X = Category/Amenity/HouseRule

Property_Category	
PK	property_category_id
FK	property_id
FK	category_id

Property_Amenity	
PK	property_amenity_id
FK	property_id
FK	amenity_id

Property_HouseRule	
PK	property_houserule_id
FK	property_id
FK	houserule_id

```sql
CREATE TABLE Property_Amenity (
    property_amenity_id INT AUTO_INCREMENT PRIMARY KEY,
    property_id INT, /*FK*/
    amenity_id INT /*FK*/
);
```

CREATE

```sql
/* Property_Amenity constraints */
ALTER TABLE Property_Amenity
ADD CONSTRAINT fk_amenity_property
FOREIGN KEY (property_id)
REFERENCES PropertyListing(propertylisting_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Property_Amenity
ADD CONSTRAINT fk_amenity_amenity
FOREIGN KEY (amenity_id)
REFERENCES Amenity(amenity_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

```sql
/* Property_Amenity
- this relation links the properties with their respective amentities
- each propertylisting will essentially have multiple entries with its id in this table,
each pointing at a the different amenities selected in the propertylisting
*/
INSERT INTO Property_Amenity (property_id, amenity_id) VALUES (1,1);
```

```sql
/* this test case shows the selected amenities for a certain propertylisting
- this same structure will work for categories and houserules as well, which will not be included as
they function identically to this view (with different names, etc.)
- i have essentially decided to view amenities, categories and houserules as an equivalence class
and chose Amenity as the representative. this can reduce testing workload/effort
(this can/could be reasoned as trying to reduce testing costs for example)
*/
CREATE VIEW iu_propertylisting_amentities_view AS
SELECT
    PL.propertylisting_id,
    PL.name AS property_name,
    A.name AS amenity_name
FROM PropertyListing PL
JOIN Property_Amenity PA ON PL.propertylisting_id = PA.property_id
JOIN Amenity A ON PA.amenity_id = A.amenity_id;

-- usage example of view
SELECT * FROM iu_propertylisting_amentities_view WHERE propertylisting_id = 1;
```

Test Case

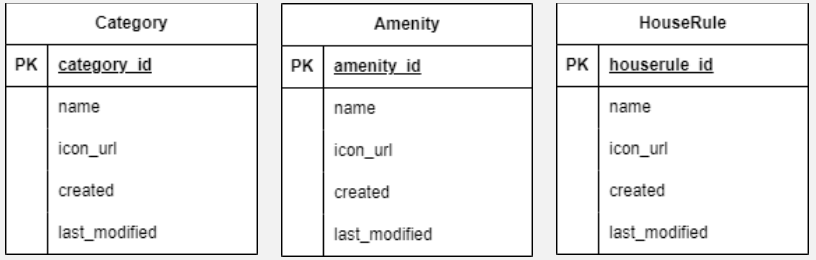
```
mysql> SELECT * FROM iu_propertylisting_amentities_view WHERE propertylisting_id = 1;
+---------------------+---------------------+--------------+
| propertylisting_id | property_name        | amenity_name |
+---------------------+---------------------+--------------+
|                  1 | Cozy Studio Apartment | Kitchen      |
|                  1 | Cozy Studio Apartment | Smoke alarm  |
|                  1 | Cozy Studio Apartment | Refrigerator |
|                  1 | Cozy Studio Apartment | Dishwasher   |
|                  1 | Cozy Studio Apartment | Lockbox      |
+---------------------+---------------------+--------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Amenity;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.01 sec)
```

COUNT

TABLE – AMENITY/CATEGORY/HOUSERULE

- These are the tables that are in a M:N relationship with the 'PropertyListing' table.
- Like the tables in the last slide, I also consider these tables one equivalence class.
- These table hold unexpectedly little information because they are only represented by their name and an icon in the application.
- Relevant for test cases are the relationships of these tables; these are already tested via other test cases, resulting in these simple queries, which check for content.



CREATE

```sql
CREATE TABLE Amenity (
    amenity_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    icon_url VARCHAR(1024),
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

INSERT

```sql
/* Amenity
- this relation is essentially a list of all the amenities that are available
- this table will not be changed often, and if it does then it will likely be an admin
- i selected some example amenities that actually exist in the airbnb application to serve as dummy data
*/
INSERT INTO Amenity (name, icon_url) VALUE ("Kitchen", "https://airbnb.com/images/amenity_icon1.ico");
```

```sql
/* Amenity data */
SELECT * FROM Amenity;

/* Category data */
SELECT * FROM Category;

/* HouseRule data */
SELECT * FROM HouseRule;
```

Simple tables, which are tested in test case on previous slide.

```
mysql> SELECT COUNT(*) FROM Propertytype;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

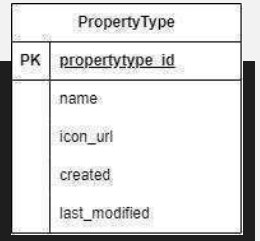
COUNT

*explained in the comment of the insert statement (2nd bullet point)

TABLE - PROPERTYTYPE

CREATE

```sql
CREATE TABLE PropertyType (
    propertytype_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    icon_url VARCHAR(1024),
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```



INSERT

```sql
/* PropertyType
- refer to the property_amenity relation above for an explanation
- this is the only table i cannot insert 20 data entries with a good conciense as there are not enough propertytypes that exist in the application.
i hope it is acceptable to not make add duplicates or unsensible entries
*/
INSERT INTO PropertyType (name, icon_url) VALUES ("Entire place", "https://airbnb.com/images/propertytype_icon1.ico");
```

‘PropertyType’ is tested via the ‘PropertyListing’ table test case.

- Different to the previous three tables, the ‘PrototypeType’ is not in a M:N relation with the PropertyListing table.
- In the Airbnb application, each property is listed based on its type.
- These types are not frequently changed, and if they are, then they are changed by an admin.
- The relevant relationship of this table is already tested in the propertylisting test case, the content itself is once again tested via a simple select all statement at the end of the test.sql file.

TABLE - BOOKING

```
mysql> SELECT COUNT(*) FROM Booking;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

- Once a guest books a property, an entry in the 'Booking' table is made.
- Most of the information listed is obvious and inferred by the attribute name.
- The time frame is stated by the booking start and end dates.
- The optional 'cancelled' Boolean and the relevant 'refund_amount' is only used in case the booking is cancelled.
- The test case displays that the relationships work as intended, and the related data is properly drawn into one coherent data entry.

CREATE

```sql
CREATE TABLE Booking (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    total_price DECIMAL(10,2) NOT NULL,
    currency_id INT DEFAULT 1, /*FK*/
    booking_start_date TIMESTAMP NOT NULL,
    booking_end_date TIMESTAMP NOT NULL,
    confirmation_code VARCHAR(16) NOT NULL,
    cancelled BOOLEAN NOT NULL,
    cancellation_date DATE,
    refund_amount DECIMAL(10,2),
    propertylisting_id INT NOT NULL, /*FK*/
    host_id INT NOT NULL, /*FK*/
    guest_id INT NOT NULL, /*FK*/
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```sql
/* Booking constraints */
ALTER TABLE Booking
ADD CONSTRAINT fk_booking_currency
FOREIGN KEY (currency_id)
REFERENCES Currency(currency_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Booking
ADD CONSTRAINT fk_booking_propertylisting
FOREIGN KEY (propertylisting_id)
REFERENCES PropertyListing(propertylisting_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Booking
ADD CONSTRAINT fk_booking_host
FOREIGN KEY (host_id)
REFERENCES Host(host_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Booking
ADD CONSTRAINT fk_booking_guest
FOREIGN KEY (guest_id)
REFERENCES Guest(guest_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Test Case

```sql
/* Booking data
- this test case serves the purpose of testing the relationships relevant to a booking table entry
*/
CREATE VIEW iu_booking_view AS
SELECT
    B.booking_id AS booking_id,
    B.propertylisting_id,
    PL.name AS propertylisting_name,
    H.host_id AS host_id,
    HU.legal_name AS host_legal_name,
    G.guest_id AS guest_id,
    GU.legal_name AS guest_legal_name
FROM Booking B
JOIN PropertyListing PL ON B.propertylisting_id = PL.propertylisting_id
JOIN Host H ON B.host_id = H.host_id
JOIN User HU ON HU.user_id = H.user_id
JOIN Guest G ON B.guest_id = G.guest_id
JOIN User GU ON GU.user_id = G.user_id;

-- usage examples of view (either look up booking via the transaction, or propertylisting)
SELECT * FROM iu_booking_view WHERE booking_id = 1;
SELECT * FROM iu_booking_view WHERE propertylisting_id = 1;
```

Booking	
PK	booking_id
	total_price
FK	currency_id
	booking_start_date
	booking_end_date
	confirmation_code
	cancelled
	cancellation_date
	refund_amount
FK	triphistory_id
FK	propertylisting_id
FK	host_id
FK	guest_id
	created

```
mysql> SELECT * FROM iu_booking_view WHERE propertylisting_id = 1;
+------------+--------------------+---------------------+---------+-----------------+----------+-----------------+
| booking_id | propertylisting_id | propertylisting_name | host_id | host_legal_name | guest_id | guest_legal_name |
+------------+--------------------+---------------------+---------+-----------------+----------+-----------------+
|          1 |                  1 | Cozy Studio Apartment |       2 | John Doe        |        3 | Dave Adams      |
+------------+--------------------+---------------------+---------+-----------------+----------+-----------------+
1 row in set (0.00 sec)
```

INSERT

```sql
/* Booking
- the booking relation holds data related to the booking of a property by a guest
- the cancellation data is obviously optional and can be NULL until updated if a cancellation occurs
*/
-- Booking 1
INSERT INTO Booking (total_price, currency_id, booking_start_date, booking_end_date, confirmation_code, cancelled, cancellation_date, refund_amount, propertylisting_id, host_id, guest_id)
VALUES (250.00, 1, '2024-03-15 12:00:00', '2024-03-20 10:00:00', 'ABCD1234', FALSE, NULL, NULL, 1, 2, 3);
```

```
mysql> SELECT COUNT(*) FROM BankInformation;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE - BANKINFORMATION

CREATE

```sql
CREATE TABLE BankInformation (
    bankinformation_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    owning_host_id INT NOT NULL, /*FK*/
    name VARCHAR(128) NOT NULL,
    account_nr VARCHAR(32) NOT NULL,
    code VARCHAR(64) NOT NULL,
    address_id INT NOT NULL /*FK*/
);
```

```sql
/* BankInformation constraints */
ALTER TABLE BankInformation
ADD CONSTRAINT fk_bankinfo_host
FOREIGN KEY (owning_host_id)
REFERENCES Host(host_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE BankInformation
ADD CONSTRAINT fk_bankinfo_address
FOREIGN KEY (address_id)
REFERENCES Address(address_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

BankInformation	
PK	bankinformation_id
FK	*owning_host_id*
	name
	account_nr
	code
FK	*address_id*

Tested via multiple other test cases, insert statement happens in the ‘Host’ transaction.

- As mentioned in the problem statement, Bank Information is only relevant for hosts.
- The ‘BankInformation’ table references the bank address as well its owning host user via the respective foreign keys.
- The rest of the information given in this table is example data relevant to Banks.
- This tables relationships are tested via other test cases (such as Host), the content itself is tested once again via a select all query at the end of the test.sql file.

```
mysql> SELECT COUNT(*) FROM CreditCardInformation;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE - CREDITCARDINFORMATION

CREATE

```sql
CREATE TABLE CreditCardInformation (
    creditcardinformation_id INT AUTO_INCREMENT PRIMARY KEY,
    owning_guest_id INT NOT NULL, /*FK*/
    bank VARCHAR(128) NOT NULL,
    card_number VARCHAR(64) NOT NULL,
    cvc_code VARCHAR(3) NOT NULL,
    expiration_date DATE NOT NULL
);
```

CreditCardInformation	
PK	creditcardinformation_id
FK	*owning_guest_id*
	bank
	card_number
	cvc_code
	expiration_date

```sql
/* CreditCardInformation constraints */
ALTER TABLE CreditCardInformation
ADD CONSTRAINT fk_ccinformation_guest
FOREIGN KEY (owning_guest_id)
REFERENCES Guest(guest_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Tested via multiple other test cases, insert statement happens in the ‘Guest’ transaction.

- As stated in the problem statement, Credit Card Information is only relevant for guests.
- The ‘CreditCardInformation’ table references its owning host user via the foreign key.
- The rest of the information given in this table is example data relevant to credit cards.
- Counterpart to the previous table, it is likewise tested in the relevant user table, in this case ‘Guest’.
- Both tables are also tested in the test cases of the ‘Transaction’ table in the following slide.

TABLE - TRANSACTION

```
mysql> SELECT COUNT(*) FROM Transaction;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

- The transaction holds relevant payment information for each booking.
- The table references ‘BankInformation’ and ‘CreditCardInformation’, as well as the booking it was made for.
- The test case can be used to check if the relationships of the transaction table and the respective payment information and ‘booking_id’ are plausible.

CREATE

```
CREATE TABLE Transaction (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    booking_id INT NOT NULL, /*FK*/
    bankinformation_id INT, /*FK*/
    creditcardinformation_id INT, /*FK*/
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```
/* Transaction constraints */
ALTER TABLE Transaction
ADD CONSTRAINT fk_transaction_booking
FOREIGN KEY (booking_id)
REFERENCES Booking(booking_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Transaction
ADD CONSTRAINT fk_transaction_bankinformation
FOREIGN KEY (bankinformation_id)
REFERENCES BankInformation(bankinformation_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Transaction
ADD CONSTRAINT fk_transaction_creditcard
FOREIGN KEY (creditcardinformation_id)
REFERENCES CreditCardInformation(creditcardinformation_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

```
/* Transaction
- this table again does not make a whole lot of sense without the application actually pulling in the relevant data from their respective tables
- the table is intended to be the base lookup table for purchase transaction data
*/
-- Transaction 1
INSERT INTO Transaction (booking_id, bankinformation_id, creditcardinformation_id)
VALUES (1, 2, 3);
```

Test Case

```
/* Transaction, CreditCard, Bankinformation data
- Test case that shows the relation between transactions with payment informations, such as
credit cards or banks and the corresponding booking
- (this test case will be seen as sufficient for testing the proper implementaiton of the tables that are
a part of it)
*/
CREATE VIEW iu_transaction_view AS
SELECT
    T.*,
    B.booking_id AS Booking_id_ref,
    CCI.creditcardinformation_id AS creditcard_id,
    CCI.card_number AS creditcard_number,
    BI.bankinformation_id AS bankinfo_id,
    BI.account_nr AS bank_account_number
FROM Transaction T
JOIN Booking B ON T.booking_id = B.booking_id
JOIN CreditCardInformation CCI ON T.creditcardinformation_id = CCI.creditcardinformation_id
JOIN BankInformation BI ON T.bankinformation_id = BI.bankinformation_id;

-- usage examples of view (either look up data via the transaction, or booking)
SELECT * FROM iu_transaction_view WHERE transaction_id = 1;
SELECT * FROM iu_transaction_view WHERE booking_id = 1;
```

Transaction	
PK	transaction_id
FK	booking_id
FK	bankinformation_id
FK	creditcardinformation_id
	created
	last_modified

```
mysql> SELECT * FROM iu_transaction_view WHERE transaction_id = 1;
+----------------+------------+--------------------+-------------------------+---------------------+---------------------+----------------+---------------+--------------------+-------------+---------------------+
| transaction_id | booking_id | bankinformation_id | creditcardinformation_id | created             | last_modified       | Booking_id_ref | creditcard_id | creditcard_number  | bankinfo_id | bank_account_number |
+----------------+------------+--------------------+-------------------------+---------------------+---------------------+----------------+---------------+--------------------+-------------+---------------------+
|              1 |          1 |                  2 |                       3 | 2024-01-08 10:03:59 | 2024-01-08 10:03:59 |              1 |             3 | 0195402326072019   |           2 | 9876543210          |
+----------------+------------+--------------------+-------------------------+---------------------+---------------------+----------------+---------------+--------------------+-------------+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Giftcard;
+----------+
| COUNT(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)
```

COUNT

TABLE - GIFTCARD

CREATE

```
CREATE TABLE GiftCard (
    giftcard_code VARCHAR(16) PRIMARY KEY,
    amount DECIMAL(10,2) NOT NULL,
    currency_id INT NOT NULL DEFAULT 1, /*FK*/
    valid_until_date TIMESTAMP NOT NULL,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

INSERT

```
INSERT INTO GiftCard (giftcard_code, amount, currency_id, valid_until_date)
VALUES ("TYTYZSNBMAQRTJOY", 15.00, 1, TIMESTAMP("2024-08-31", "00:00:00"));
```

‘GiftCard’ is a simple table, see content test in test.sql

- The ‘GiftCard’ table contains information relevant to gift cards.
- The ‘amount’ attribute in combination with the ‘currency_id’ attribute are used to store the value of the gift card.
- The GiftCard table is quite simple and not as important as other tables to the functionality of the system and therefore does not warrant a complicated test case, content is checked again at the end of the test.sql file.

THANKS FOR READING

I am looking forward to your feedback!

Phase 3 – Finalization

Documents

Abstract - Making of AirBnB Database

Project Overview

This project consisted of three phases, in the conception phase I researched and analyzed the database by inspecting the website itself, the functionalities it provides as well as other implementations that are close in nature to the desired outcome. During the concept, I also invested a lot of time into researching what the requirements for such a database, and pitfalls often cause trouble during implementation.

After creating a solid plan and receiving feedback on the ER diagram creating during the initial conception phase, I set out to implement the tables. During implementation, I iteratively worked on the overall table structure (and ER diagram), the test cases and relevant test data to periodically test the database during implementation.

This has helped me produce a solid database while avoiding major time sinks that may have halted development. More about this during the “Learnings” section of this abstract.

As I received positive feedback after the implementation phase, I mainly focused on improving the overall structure of the submission by refactoring the folder structure of the GitHub repository and making said repository public. I also added and improved readme files for each phase making it as easy as possible to navigate.

Lastly I also carried out finalization tests to make sure the provided sql files run without issues and made a few quality-of-life improvements.

Functionality

As initially mentioned in the concept during phase 1, this database’s main goal is to provide structure which is sensible and easy to use. The idea is almost akin to a modular kit that can be used to build off of.

It should be very user friendly and easy to use in a variety of applications. I tried to achieve this by reducing the number of necessary queries to a minimum and creating template transactions that can be adjusted to fit ones needs. This is the case for both adding and deleting data entries from the database. More about this in the “Learnings section” below.

Another goal was to offer easy deletion or expansion to the system. This is exemplified by how easy it was to delete the triphistory table during the implementation phase.

The actual functionality of the existing tables is talked about in the presentation provided during phase 2. To summarize the overall functionality: The database allows creation of users, the User class is then categorized into the two subclasses Guest & Host.

Data such as images or addresses are created to be usable by many different tables. Addresses are shared by many tables without relying on any of them, the reason for this is that they can be altered easily without having to adjust other tables. The relation between these tables is stored via FK in the respectively linked table, so a user or a property listing would for example have a reference to the address table which can be used when altering the address table.

Images are contained in their own table that stores a bunch of relevant data, the images themselves are stored using a simple URL link. This is because saving images as BLOPs uses a lot of space without providing any real advantages, other than maybe convenience, when compared to storing the images in, for example, a cloud storage and then accessing said file system via the URL stored in the database.

Other important features include saving relevant booking data via multiple N:M relationships, making it easy to adjust available property categories, amenities, house rules, etc.

Meta data

The database consists of 27 tables and 12 views, resulting in a total of 39 entities in the database. The views are used for test cases while the tables form the actual structure of the database.

Running the SQL statements provided in the “metadata.sql” file you can see that the database is also quite small in size, taking up only 1 MB in size. The individual table size also does not exceed 80 KB with the booking table being the largest. This was achieved by researching the minimum necessary size of each individual attribute in the database.

All tables have 20 data entries, some may have more, such as address or user which are required by other tables, resulting in a total of 672 data entries of test data across all tables. I have appended a table showing the individual table sizes and number of data entries below for convenience.

Learnings

I am overall very happy with how the project went, but in this section, I’d like to quickly summarize the mistakes I made during this project, how I fixed them and what I learned from them.

During the conception phase, I made the mistake of not adding a data dictionary right away and having ambiguous naming conventions. In the future I’d like to develop a solid naming convention from the beginning of a project to reduce mistakes such as this, as well as implementing and maintaining a data dictionary right away.

After one of the test phases I noticed that the triphistory table, although helpful may be unnecessary. This resulted in me overhauling some of the dependencies in the system to improve the overall structure, although I am aware that the process of developing such a project is an iterative one, I can’t help but

feel that I could have accounted for this error during the conception phase with some more in-depth planning.

Reflecting on the workflow, the balance between error-prevention and error-correction may have been a little too one-sided towards prevention, spending too time testing for example.

All in all, I am happy with what I have achieved.

Table Sizes:

Table	Size (KB)	Data entries*
booking	80.00	20
propertylisting	64.00	20
transaction	64.00	20
bankinformation	48.00	20
chat	48.00	20
emergencycontact	48.00	20
guest	48.00	20
host	48.00	20
property_amenity	48.00	20
property_category	48.00	20
property_houserule	48.00	20
propertyreview	48.00	20
userreview	48.00	20
wishlist_propertylisting	48.00	20
creditcardinformation	32.00	20
giftcard	32.00	20
image	32.00	85
message	32.00	20
wishlist	32.00	20
address	16.00	80
amenity	16.00	20
category	16.00	20
currency	16.00	20
houserule	16.00	20
language	16.00	20
propertytype	16.00	4
user	16.00	40

*Data entries are irrelevant for size measurement.

Instruction Manual & Documentation

In this instruction manual, I'd like to explain how to install, setup and use this database. A markdown version of this manual can also be found on the GitHub repository.

Structure:

1. Dependencies
2. Setup Environment
3. Database Installation
4. Database Usage
5. Documentation

Dependencies

- This database was built to have as few dependencies as possible. Currently only MySQL 8.1 and a way to access it are necessary. Recommendations follow.
- This database is built using MySQL 8.1, it has not been tested on any other versions, but should in theory work on any newer version as well.
- I recommend using the MySQL 8.1 Community server together with the MySQL 8.1 Command Line Client. This setup is explained below.

Setup Environment

Everything necessary to use this database can be setup using the MySQL community installer. While this may look very long, it's very quick and you can **start testing the database within 5 minutes**.

It is assumed that the Windows operating system will be used, there may be slight differences in case you are using a different OS. In that case, please see the link at the end of this section on how to setup MySQL on different operating systems.

Instructions on the installation process follow:

1. Follow this link: https://downloads.mysql.com/archives/community/, select version 8.1.0 and your operating system in the dropdown menu and download the MSI installer.
2. Run the downloaded Installer, proceed with default settings and keep the option "Run MySQL configurator" ticked before clicking Finish.

3. In the configurator, keep “Type and Networking” default and click on Next. In the Accounts and Roles tab, setup your root password. **Important: remember this password** for later.
4. After setting your Root password, the next two tabs can be left as is, until you proceed to the tab “Apply Configurations”. On this tab click “Execute” and then proceed. On the last tab simply click Finish.
5. You should now have a MySQL server and the command line client installed on your system. If you do not see the command line client, you can also add the PATH to your MySQL installations “bin” folder to your environment variables and use the normal command line.

You can now use the command line to access the MySQL server and start testing the database.

More information can be found in the official documentation: https://dev.mysql.com/doc/mysql-installation-excerpt/8.3/en/.

Database Installation

To test and use the database let’s setup the database structure, triggers, and test data and test case views.

To install these open up the previously installed Command Line Client and enter the password you set during the configuration. You can now start writing MySQL statements.

Important: Read before proceeding, the schema.sql (used to setup the table structure and FKs) overwrites the database if there already exists one with the same name. The name of the database includes my matriculation number (see submission name) so the chance of this happening should be extremely low. In case you want to name the database yourself or already have one that exists, edit lines 12 through 14 in the schema.sql file. This is also necessary for the second way of installing the database.

There are two ways for this installation. **Recommended**: simply copy-paste and run all statements from each file on the command lines (order is shown below).

Installation order:

1. schema.sql
2. triggers.sql (optional)
3. data.sql (optional)

4. test.sql (optional)

The other way if you have set up the environment variable is to open the command line in the folder, and then adjust and run the following command for each of the files above.

mysql -u your_username -p your_database_name < file.sql

Database Usage

Now the database structure, including test data and test cases should be installed. To use them either use normal MySQL syntax or use one of the test cases listed in the following Documentation section.

To insert data into the database, simply adjust one of the insert example statements found in the “data.sql” file for example.

Documentation

Database Overview

To gain an overview of each of the tables, it is recommended to look at the provided PowerPoint presentation. The presentation includes explanations about the general use and function of the table as well as its relationship to other tables.

File Overview

It is recommended to open up and browse the files for more information regarding specific test cases or data. All statement blocks are documented using comments.

Schema

This file contains all statements necessary to create the table structure of the database. It also creates the database and can therefore be run right away after opening the MySQL command line. No data is being created using this file.

Triggers

This file can be omitted without any influence on the process and only serves as an example of how conditions could be added to this project in a modular and future-proof way.

Data

This file includes the bulk of the statements. This file includes 2000+ (includes docs & white space) lines of sql statements that serve to fill the database with properly linked, semi-realistic test data.

Metadata

This file includes the statements that were used to provide the metadata mentioned in the submission accompanying abstract. This file can be executed as is but can also be ignored as it is not relevant to the implementation of the database.

Test

Test.sql includes test cases for each table, putting focus on more important tables that are relevant to the main functionality of the database. The provided primarily test the integrity of the relationship between tables and the related data stored within them. This means that test cases will select data entries that store data over multiple related tables. By viewing this information, we can confirm that data was properly inserted into these tables. (This file also includes examples which will run during installation). Additionally, equivalence classes have been used to reduce the workload to a reasonable amount without reducing the achieved result.

List of test case views

This list merely serves as an overview of all test cases, so you know what to search for. Documentation can be found via the comments in the test case file. (This pdf would otherwise get quite bloated.) Example use cases for each of the test case will also be provided.

- iu_userguest_view
    - SELECT * FROM iu_userguest_view WHERE user_id = 1;
- iu_userhost_view
    - SELECT * FROM iu_userhost_view WHERE user_id = 21;
- iu_propertylisting_view
    - SELECT * FROM iu_propertylisting_view WHERE propertylisting_id = 1;
    - SELECT * FROM iu_propertylisting_view WHERE host_id = 1;
- iu_propertylisting_amentities_view
    - SELECT * FROM iu_propertylisting_amentities_view WHERE propertylisting_id = 1;
- iu_userreviews_view

    - SELECT * FROM iu_userreviews_view WHERE user_id = 1;
- iu_propertyreviews_view
    - SELECT * FROM iu_propertyreviews_view WHERE propertylisting_id = 1;
- iu_booking_view
    - SELECT * FROM iu_booking_view WHERE booking_id = 1;
    - SELECT * FROM iu_booking_view WHERE propertylisting_id = 1;
- iu_transaction_view
    - SELECT * FROM iu_transaction_view WHERE transaction_id = 1;
    - SELECT * FROM iu_transaction_view WHERE booking_id = 1;
- iu_wishlist_details_view
    - SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;
- iu_wishlist_propertylistings_view
    - SELECT * FROM iu_wishlist_propertylistings_view WHERE wishlist_id = 1;
- iu_chat_details_view
    - SELECT * FROM iu_chat_details_view WHERE chat_id = 1;
- iu_chat_messages_view
    - SELECT * FROM iu_chat_messages_view WHERE owning_chat_id_ref = 1;

To delete from the database, use simple DELETE FROM statements and delete all relevant data from the tables. I recommend writing transactions to protect transactions to protect data integrity. To see the relevant tables that need their data deleted, please view the ER diagram and write your transactions that way.

Example for deleting all data relevant to a certain user:

START TRANSACTION;
DELETE FROM UserReview WHERE user_id = 1;
DELETE FROM Booking WHERE guest_id = 1 OR host_id = 1;
DELETE FROM Host WHERE user_id = 1;
DELETE FROM Guest WHERE user_id = 1;
DELETE FROM Image WHERE uploaded_by_user_id = 1;
DELETE FROM Address WHERE address_id IN (SELECT address_id FROM User WHERE user_id = 1);
DELETE FROM User WHERE user_id = 1;
COMMIT;