AIRBNB DATABASE PRESENTATION

PHASE 2 - IMPLEMENTATION

By Tom Schmaeling

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.

- I encourage the reader to run the test commands in their own environment for better readability.

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

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.
- The primary key's now have more descriptive names.
- Small additional changes include:
  - 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

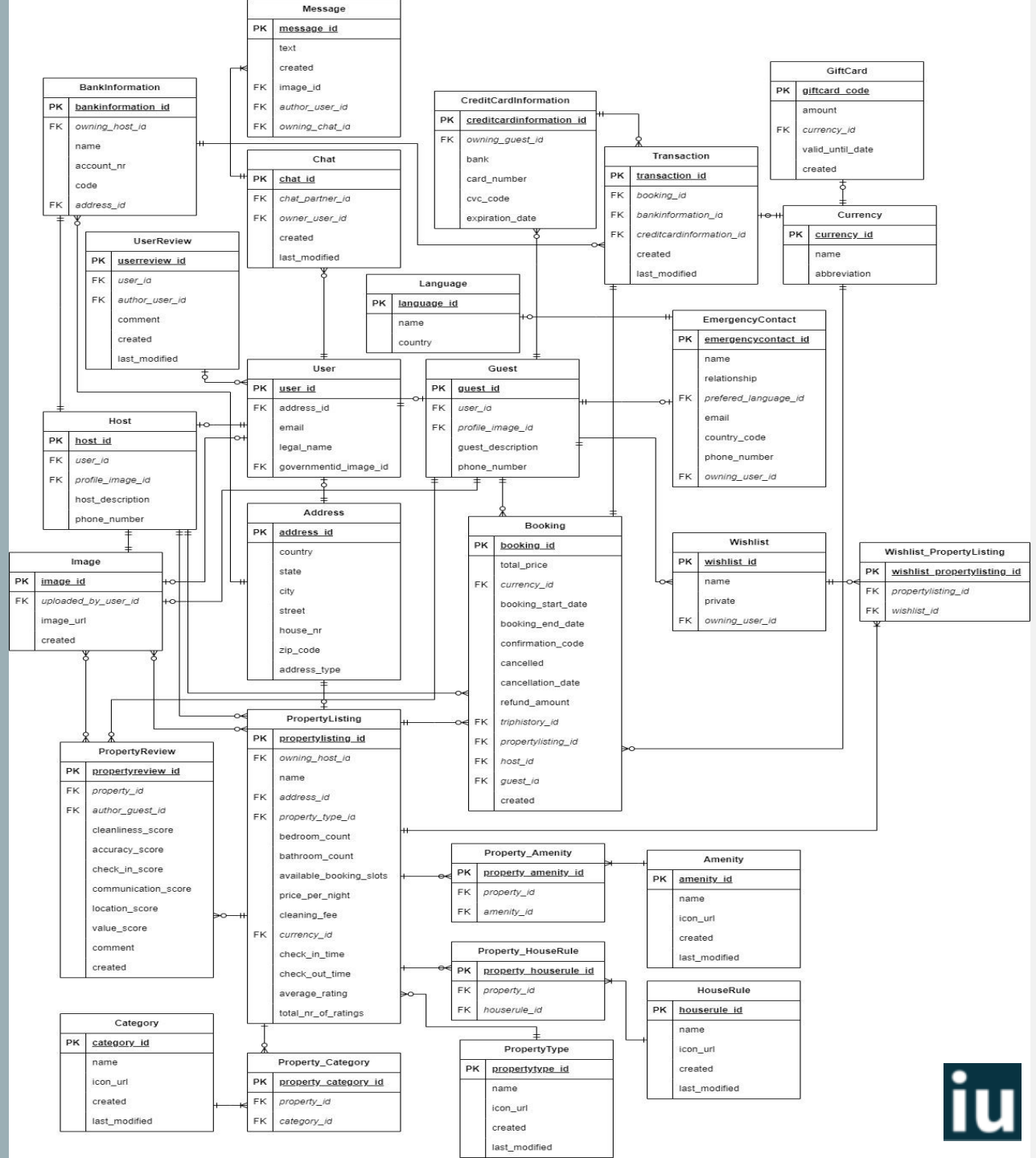
STRUCTURE

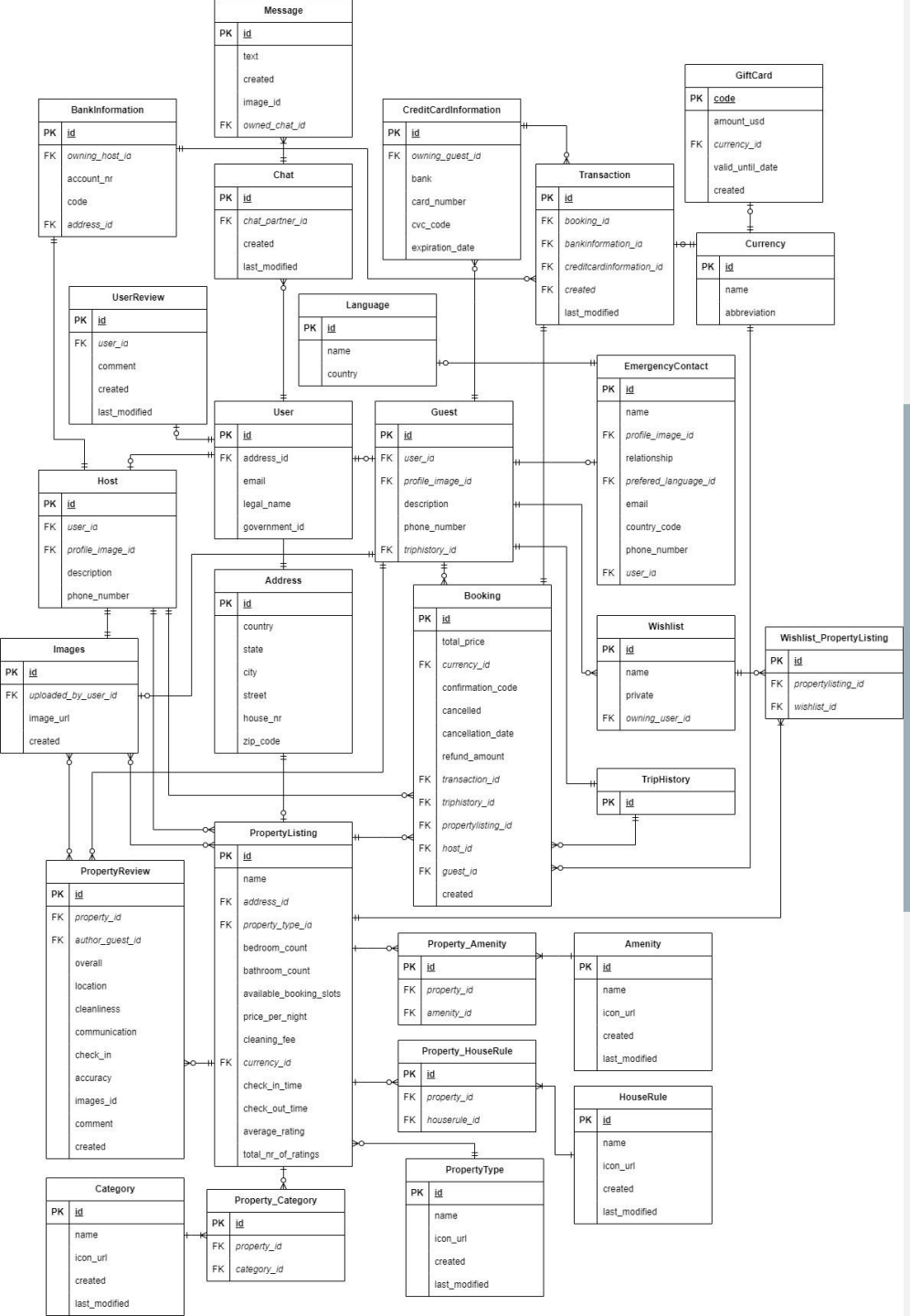
There are two main “blocks” of information the database needs to support: Users and Properties.

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.





STRUCTURE CHANGES OVERVIEW

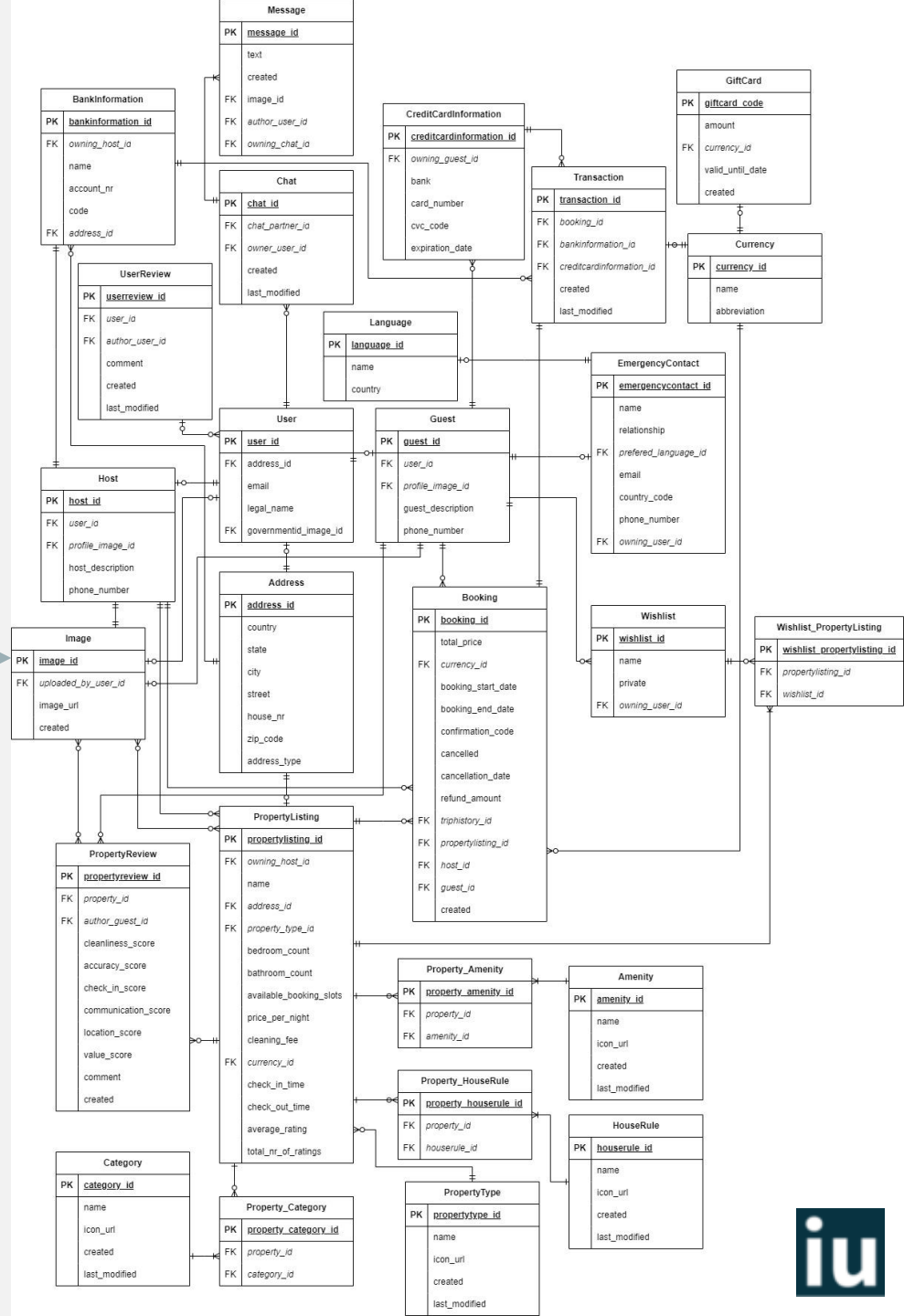


TABLE - USER

User	
PK	user_id
FK	address_id
	email
	legal_name
FK	governmentid_image_id

Tested via ‘Guest’ and ‘Host’ test cases

- 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

```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

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

- 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 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.

```
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_image_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.com/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

```sql
/* 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;
```

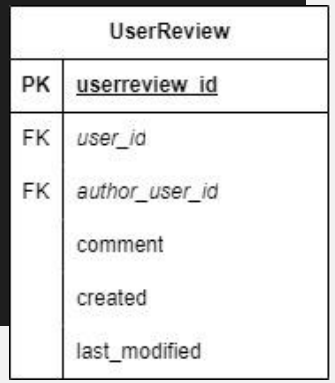
Host	
PK	host_id
FK	*user_id*
FK	*profile_image_id*
	host_description
	phone_number

```
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/governmentid_image21.jpg | 9876543229 | 123 | 22 | United Kingdom |
+---------+--------------+---------------------+---------+---------+-------------+----------+-----------+
1 row in set (0.00 sec)
```

- 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 for
- The ‘triphistory_id’ is used to link together
- The test case is very similar to that of the previous guest class, adjusting where relevant relationships change compared to the previous slide.

TABLE - USERREVIEW

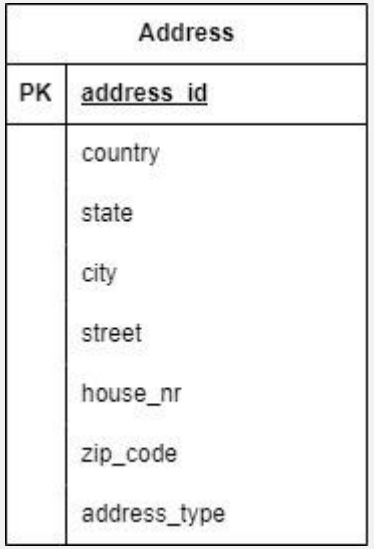
```sql
/* UserReview data from related to explicit user
- quite simple as the userreview shares the attribute 'user_id' which can be used for a natural join,
otherwise a left join on user_id could be used when only specific data is desired
*/
CREATE VIEW iu_userreviews_from_user_view AS
SELECT *
FROM 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;
```



- 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.

```
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)
```

TABLE - ADDRESS



Tested via multiple other test cases

- 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 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.

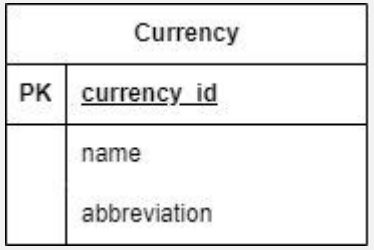
TABLE - IMAGE

Image	
PK	image_id
FK	*uploaded_by_user_id*
	image_url
	created

Tested via multiple other test cases

- 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.

TABLE - CURRENCY



'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.
- This table is very simple and does not require complicated testing, a simple select all test can be found in the test.sql file.

TABLE - LANGUAGE

Language	
PK	language_id
	name
	country

'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.
- 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

```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

- 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 holds 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.

```
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

```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;
```

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)
```

- 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 test the relationship of ‘Chat’ and ‘Message’ tables by returning all data entries of messages for a given

TABLE - EMERGENCYCONTACT

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

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

- The emergency contact 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.
- All the information regarding the contact should be self-explanatory.
- The 'owning_user_id' refers to the user that owns it.
- 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.

TABLE - WISHLIST

```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*

- 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 chat table and other related tables in conjunction. The second test checks the relation of a chat and its messages.

```
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)
```

TABLE – WISHLIST_PROPERTYLISTING

```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*

- 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.

```
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

```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)
```

- 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 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 from all tables that have a relationship with the propertylisting table. (Either via property or host id)

TABLE - PROPERTYREVIEW

```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;
```

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

- 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)
- 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.

```
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)
```

TABLE – PROPERTY_X

*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

```
/* 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;
```

```
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)
```

- 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 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.

TABLE – AMENITY/CATEGORY/HOUSERULE

Category	
PK	category_id
	name
	icon_url
	created
	last_modified

Amenity	
PK	amenity_id
	name
	icon_url
	created
	last_modified

HouseRule	
PK	houserule_id
	name
	icon_url
	created
	last_modified

```
305   /* Amenity data */
306   SELECT * FROM Amenity;
307
308   /* Category data */
309   SELECT * FROM Category;
310
311   /* HouseRule data  */
312   SELECT * FROM HouseRule;
```

Simple tables, which are tested in previous test case.

- 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 other test cases, resulting in these simple queries, which check for content.

TABLE - PROPERTYTYPE

PropertyType	
PK	propertytype_id
	name
	icon_url
	created
	last_modified

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

- 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 table is already tested in the propertylisting test case, the content itself is once again tested via a simple select all statement in the test.sql file.

TABLE - BOOKING

```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;
```

```
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)
```

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

- 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 cases displays that the relationships work as intended, and the related data is properly drawn into one coherent data entry.

TABLE - BANKINFORMATION

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

Tested via multiple other test cases

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

TABLE - CREDITCARDINFORMATION

CreditCardInformation	
PK	creditcardinformation_id
FK	*owning_guest_id*
	bank
	card_number
	cvc_code
	expiration_date

Tested via multiple other test cases

- As stated in the problem statement, Credit Card Information is only relevant for guests.
- The 'CreditCardInformation' table references its address as well its owning host user via ids.
- The rest of the information given in this table is example data relevant to Banks.
- Counterpart to the previous table, both are also tested in the test cases of the 'Transaction' table in the following slide.

TABLE - TRANSACTION

```sql
/* 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

- 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.

```
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)
```

TABLE - GIFTCARD

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

- The 'GiftCard' table contains information relevant to gift cards.
- The 'amount' attribute saves the value of gift card in us dollars.
- The 'currency_id' attribute can then be used to convert it, if necessary
- The GiftCard table is a very simple, which is 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

Additional information is given in the Pebble Pad submission as well as in code via comments.