

# **Phase 2 – Implementation Documents**

# AIRBNB DATABASE PRESENTATION

PHASE 2 - IMPLEMENTATION

By Tom Schmäling

# TABLE OF CONTENTS

1. Introduction
2. Changes compared to Phase I
3. General Database Structure
4. Examining Individual Table

# INTRODUCTION

This presentation is a part of phase 2 and intends to provide extensive documentation of the database structure, its 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

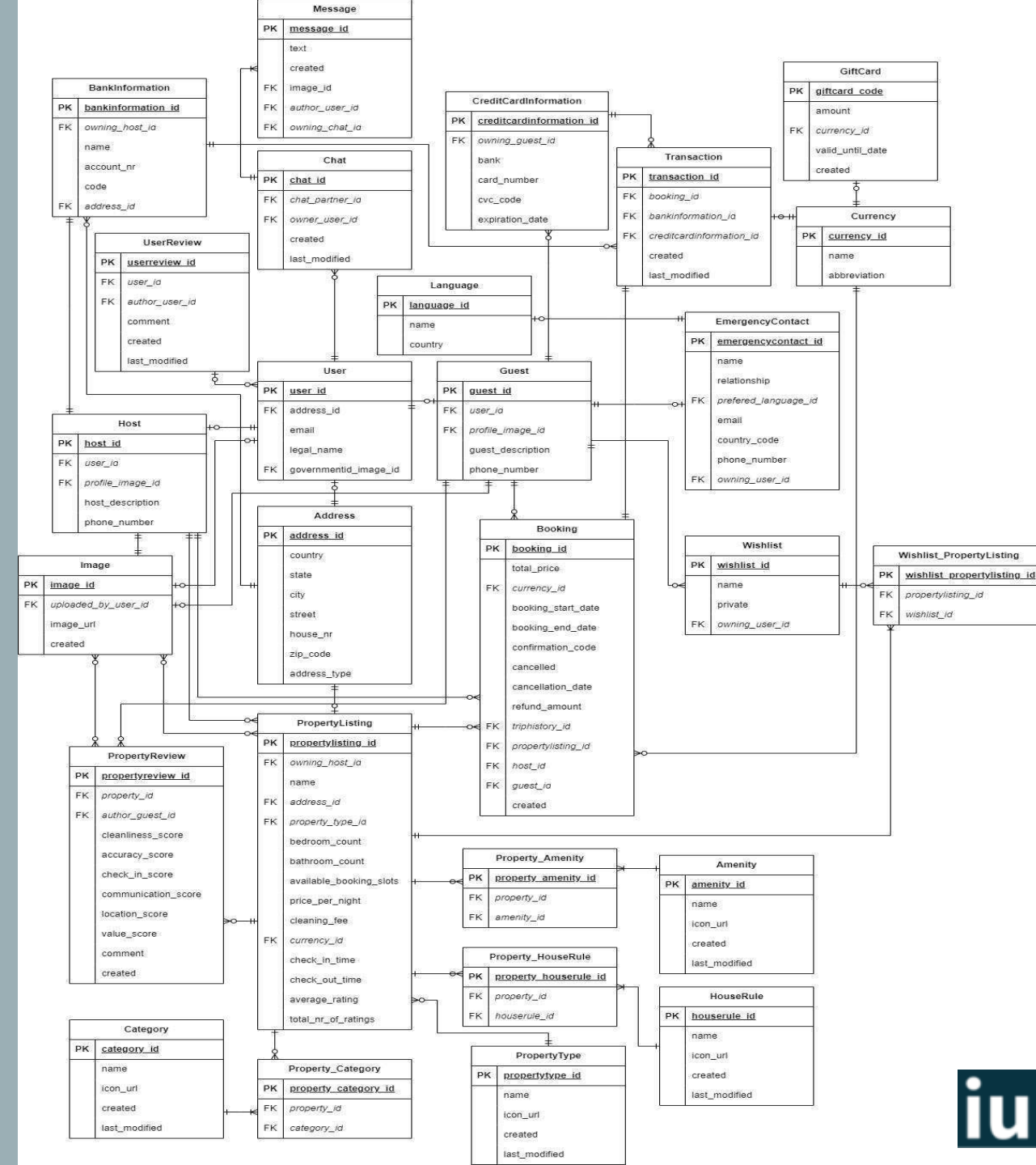
# STRUCTURE

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

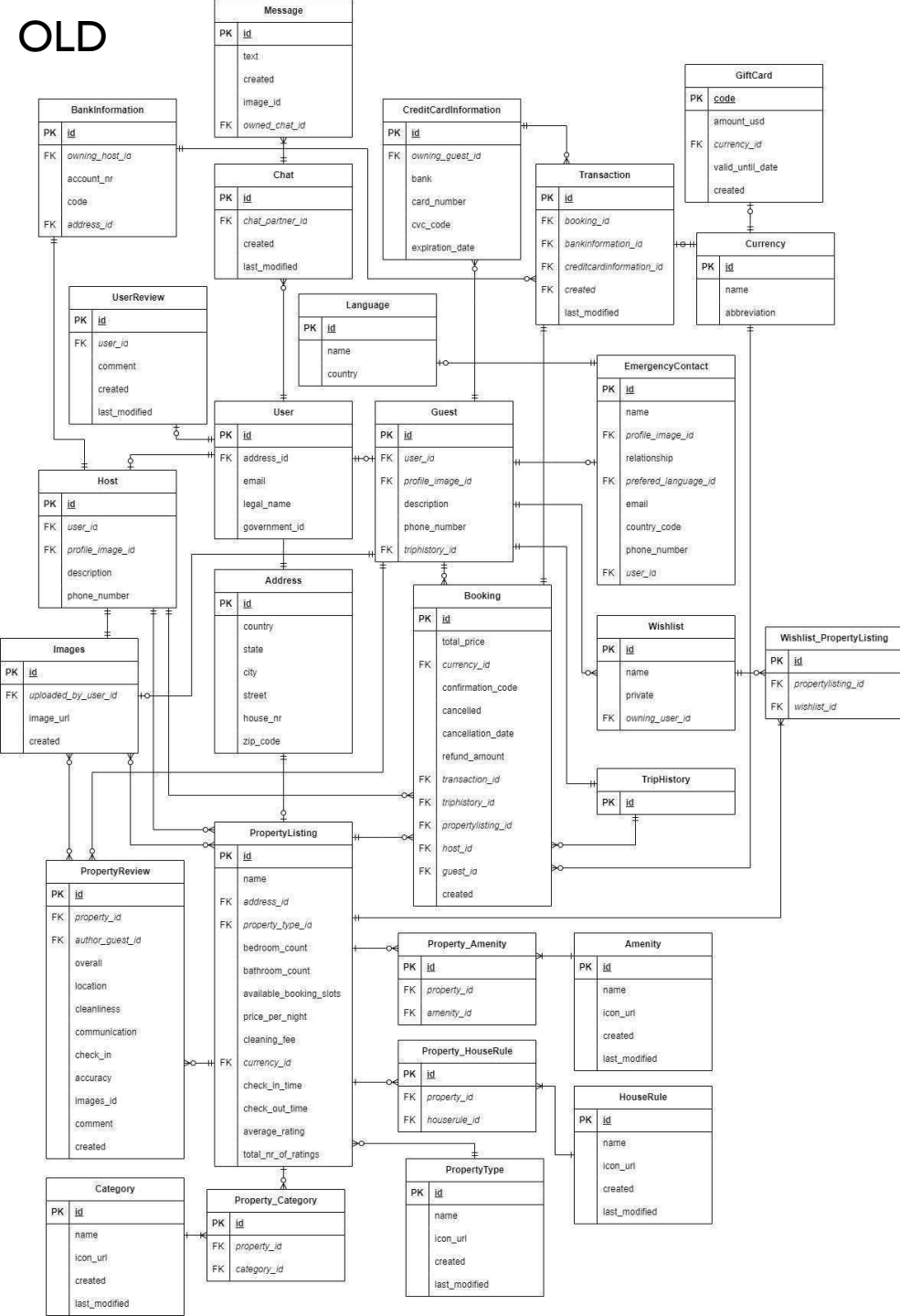
There are multiple tables facilitating each of these data sets, and their attributes.

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.

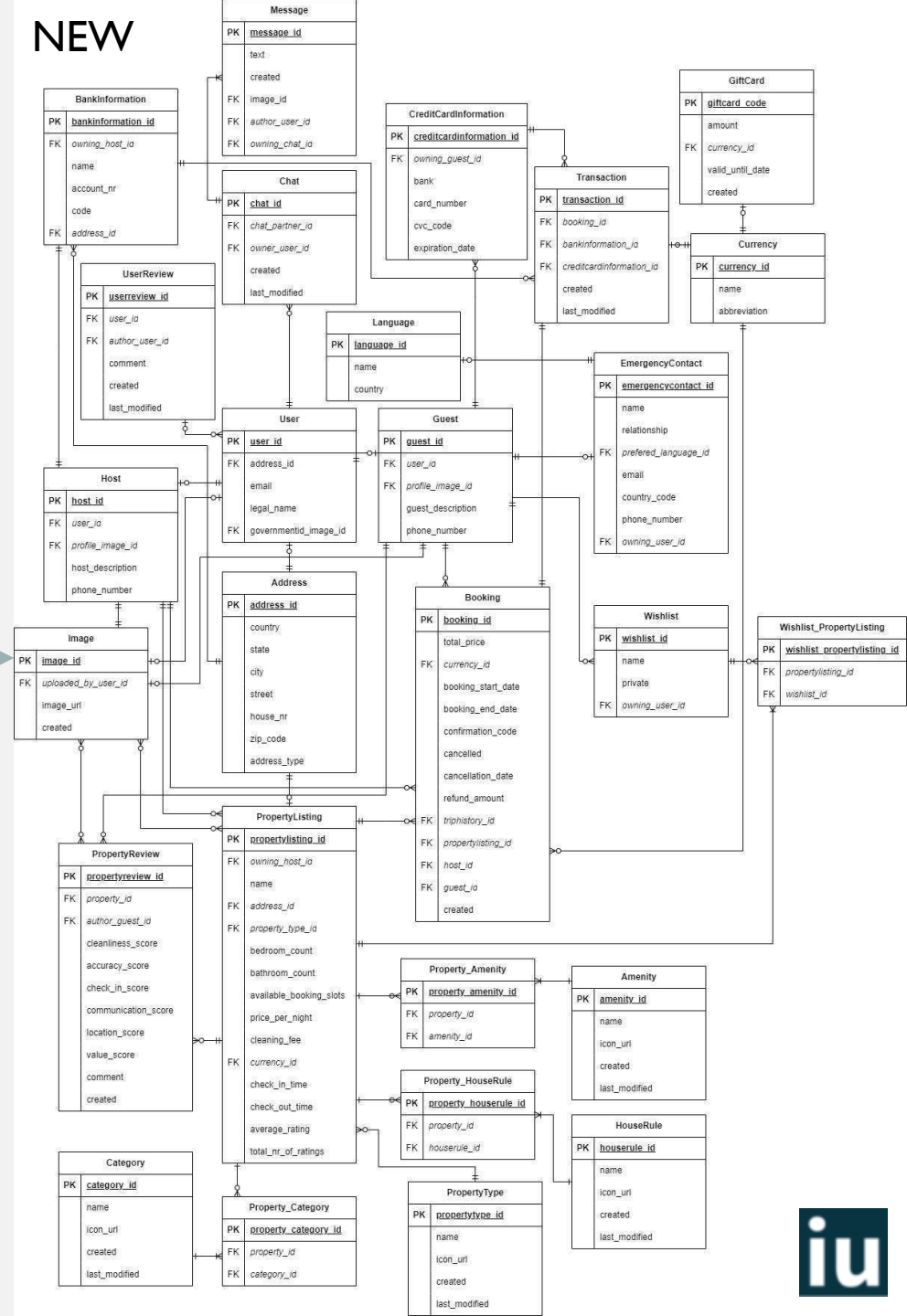


OLD



# 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

INSERT

## Test Case

- 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.

CREATE

```

10  /* relevant guest data
11  - it could be argued that this view should be based on the guest_id instead of the user_id but this
12  is an easy change and depends on how the view is to be used in practice
13  - as it is this view serves as a demonstration for how the connected data can be queried by composing
14  a complex select statement into a view for easier use.
15  */
16  CREATE VIEW iu_userguest_view AS
17  SELECT
18      U.user_id,
19      U.legal_name,
20      U.email,
21      A.country,
22      A.state,
23      A.city,
24      A.street,
25      A.house_nr,
26      A.zip_code,
27      G.guest_description,
28      G.phone_number,
29      IProfile.image_url AS profile_image_url,
30      IGovernID.image_url AS governmentid_image_url,
31      EC.name AS emergency_contact_name,
32      EC.relationship AS emergency_contact_relationship,
33      EC.email AS emergency_contact_email,
34      EC.country_code AS emergency_contact_country_code,
35      EC.phone_number AS emergency_contact_phone_number,
36      CCI.bank AS credit_card_bank,
37      CCI.card_number AS credit_card_number,
38      CCI.cvc_code AS credit_card_cvc,
39      CCI.expiration_date AS credit_card_expiration_date
40  FROM
41      User U
42  JOIN Address A ON U.address_id = A.address_id
43  JOIN Guest G ON U.user_id = G.user_id
44  JOIN Image IProfile ON G.profile_image_id = IProfile.image_id
45  JOIN Image IGovernID ON U.governmentid_image_id = IGovernID.image_id
46  LEFT JOIN EmergencyContact EC ON U.user_id = EC.owning_user_id
47  LEFT JOIN CreditCardInformation CCI ON G.guest_id = CCI.owning_guest_id;
48
49  -- usage of view
50  SELECT * FROM iu_userguest_view WHERE user_id = 1;

```

Guest	
PK	<u>guest_id</u>
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_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 nomad. I love traveling...	+49987654321	https://airbnb.com/images/profile_image1.jpg	https://airbnb.com/images/governmentid_image1.jpg	Anna Mustermann	Family	anna@example.com	123456789	Sparda Bank	3353422819762527	123	2026-10-31

1 row in set (0.00 sec)

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

COUNT

# TABLE - HOST

- 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.

```
851 /*
852 User, Host, Image, BankInformation, Address (SQL Transaction)
853
854 the structure of the host insert statements is very similar to the guest as they share a lot of similarities
855
856 * i will only comment the first transaction as the rest follow the same schematic
857 */
858 -- Host 1
859 START TRANSACTION;
860
861 /* Address for the Host, saved for later use */
862 INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
863 VALUES ("United Kingdom", "England", "London", "123 Main St", 123, "SW1A 1AA", "host");
864 SET @address_id = LAST_INSERT_ID();
865
866 /* User (super) class of the Host, saved for later use */
867 INSERT INTO User (address_id, email, legal_name)
868 VALUES (@address_id, "william.turner@hotmail.com", "William Turner");
869 SET @user_id = LAST_INSERT_ID();
870
871 /* Profile Image of the Host, saved for later use */
872 INSERT INTO Image (uploaded_by_user_id, image_url)
873 VALUES (@user_id, "https://airbnb.com/images/governmentid_image21.jpg");
874 SET @image_governmentid_id = LAST_INSERT_ID();
875 UPDATE User SET governmentid_image_id = @image_governmentid_id WHERE user_id = @user_id;
876
877 /* government id image of the Host (User) */
878 INSERT INTO Image (uploaded_by_user_id, image_url)
879 VALUES (@user_id, "https://airbnb.com/images/profile_image21.jpg");
880 SET @image_profile_id = LAST_INSERT_ID();
881
882 /* Host data, using the previously inserted data */
883 INSERT INTO Host (user_id, profile_image_id, host_description, phone_number)
884 VALUES (@user_id, @image_profile_id, "Host description for host 1", "9876123469");
885 SET @host_id = LAST_INSERT_ID();
886
887 /* Bank Address - this information should likely be inserted differently in an actual production environment, but this works for now */
888 INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)
889 VALUES ("United Kingdom", "England", "London", "456 Oak St", 456, "SW1A 1AA", "bank");
890 SET @bankaddress_id = LAST_INSERT_ID();
891
892 /* BankInformation data, using the previously inserted address data */
893 INSERT INTO BankInformation (owning_host_id, name, account_nr, code, address_id)
894 VALUES (@host_id, "Barclays Bank plc", "9876543229", "123", @bankaddress_id);
895
896 COMMIT;
```

INSERT

CREATE

```
44 CREATE TABLE Host (
45     host_id INT AUTO_INCREMENT PRIMARY KEY,
46     user_id INT UNIQUE NOT NULL, /*FK*/
47     profile_image_id INT UNIQUE NOT NULL, /*FK*/
48     host_description TEXT NOT NULL,
49     phone_number VARCHAR(16)
50 );
283 /* Host constraints */
284 ALTER TABLE Host
285 ADD CONSTRAINT fk_host_user
286 FOREIGN KEY (user_id)
287 REFERENCES User(user_id)
288 ON DELETE CASCADE ON UPDATE RESTRICT;
289
290 ALTER TABLE Host
291 ADD CONSTRAINT fk_host_image
292 FOREIGN KEY (profile_image_id)
293 REFERENCES Image(image_id)
294 ON DELETE CASCADE ON UPDATE RESTRICT;
```

```
mysql> SELECT * FROM iu_userhost_view WHERE user_id = 21;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | legal_name | email | phone_number | profile_image_url | country | state | city | street | house_nr | zip_code | governmentid_image_url |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 21 | William Turner | william.turner@hotmail.com | 9876123469 | https://airbnb.com/images/profile_image21.jpg | United Kingdom | England | London | 123 Main St | 123 | SW1A 1AA | https://airbnb.com/images/governmentid_image21.jpg |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Test Case

```
53 /* relevant host data
54 - it could be argued that this view should be based on the host_id instead of the user_id but this
55 is an easy change and depends on how the view is to be used in practice
56 - as it is this view serves as a demonstration for how the connected data can be queried by composing
57 a complex select statement into a view for easier use.
58 */
59 CREATE VIEW iu_userhost_view AS
60 SELECT
61     U.user_id,
62     U.legal_name,
63     U.email,
64     A.country,
65     A.state,
66     A.city,
67     A.street,
68     A.house_nr,
69     A.zip_code,
70     H.host_description,
71     H.phone_number,
72     IProfile.image_url AS profile_image_url,
73     IGovernID.image_url AS governmentid_image_url,
74     BI.account_nr AS bank_account_nr,
75     BI.code AS bank_code,
76     BI.address_id AS bank_address_id,
77     BIA.country AS bank_address_country
78 FROM
79     User U
80 JOIN Address A ON U.address_id = A.address_id
81 JOIN Host H ON U.user_id = H.user_id
82 JOIN Image IProfile ON H.profile_image_id = IProfile.image_id
83 JOIN Image IGovernID ON U.governmentid_image_id = IGovernID.image_id
84 LEFT JOIN BankInformation BI ON H.host_id = BI.owning_host_id
85 JOIN Address BIA ON BI.address_id = BIA.address_id;
86
87 -- usage example of view
88 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 COUNT(*) FROM UserReview;
+-----+
| COUNT(*) |
+-----+
|      20 |
+-----+
1 row in set (0.00 sec)
```

COUNT

# TABLE - USERREVIEW

## INSERT

```
1527  /* UserReview
1528  - The UserReview is written by a host to the profile of a guest who stayed at their property
1529  - The author therefore has to be a host, while the user the review is written to has to be a guest
1530  - There are no other restrictions except that both id's have to be valid
1531  */
1532  -- UserReview 1
1533  INSERT INTO UserReview (user_id, author_user_id, comment)
1534  VALUES (1, 21, "They were a lovely guest, we hope to meet you again some time!");
```

## CREATE

```
60  CREATE TABLE UserReview (
61      userreview_id INT AUTO_INCREMENT PRIMARY KEY,
62      user_id INT NOT NULL, /*FK*/
63      author_user_id INT NOT NULL, /*FK*/
64      comment VARCHAR(2000),
65      created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
66      last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
67
68      FOREIGN KEY (user_id) REFERENCES User(user_id),
69      FOREIGN KEY (author_user_id) REFERENCES User(user_id)
70  );
```

## Test Case

- 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.

```
156  /* UserReview data from related to explicit user
157  - quite simple as the userreview shares the attribute 'user_id' which can be used for a natural join,
158  otherwise a left join on user_id could be used when only specific data is desired
159  */
160  CREATE VIEW iu_userreviews_from_user_view AS
161  SELECT *
162  FROM User
163  NATURAL JOIN UserReview;
164  /* another test example for only the UserReview data without User data:
165  SELECT UR.*
166  FROM User
167  NATURAL JOIN UserReview UR;
168  */
169  -- usage example of view
170  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

```
16 CREATE TABLE Address (
17     address_id INT AUTO_INCREMENT PRIMARY KEY,
18     country VARCHAR(128) NOT NULL,
19     state VARCHAR(128) NOT NULL,
20     city VARCHAR(128) NOT NULL,
21     street VARCHAR(128) NOT NULL,
22     house_nr INT NOT NULL,
23     zip_code VARCHAR(10) NOT NULL,
24     address_type VARCHAR(32)
25 );
```

Address	
PK	<u>address_id</u>
	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

```
35 CREATE TABLE Image (
36   image_id INT AUTO_INCREMENT PRIMARY KEY,
37   uploaded_by_user_id INT NOT NULL, /*FK*/
38   image_url varchar(1024) NOT NULL,
39   created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
40
41   FOREIGN KEY (uploaded_by_user_id) REFERENCES User(user_id)
42 );
```

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

```
201 CREATE TABLE Currency (
202     currency_id INT AUTO_INCREMENT PRIMARY KEY,
203     name VARCHAR(64) NOT NULL,
204     abbreviation VARCHAR(3)
205 );
```

Currency	
PK	<u>currency_id</u>
	name
	abbreviation

INSERT

```
47 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	<u>language_id</u>
	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.



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

COUNT

# TABLE - CHAT

- 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

```
1613 /* Chat
1614 - the chat table is hard to make sense of without the frontend, the chat table holds the ids of the two users chatting,
1615 the owner being the initiator of the chat
1616 - the id's are referenced by the messages to string together the whole chat
1617 - important is that the user id's have to be valid and they cannot be identical, as a user can't chat with themselves
1618 - the partner/owner naming is just for context and does not have to be regarded when requesting a chat
1619 (otherwise duplicates would be needed, which makes no sense)
1620 */
1621 INSERT INTO Chat (chat_partner_id, owner_user_id) VALUES (1, 2);
```

```
242 CREATE TABLE Chat (
243     chat_id INT AUTO_INCREMENT PRIMARY KEY,
244     chat_partner_id INT, /*FK*/
245     owner_user_id INT, /*FK*/
246     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
247     last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
248 );
```

CREATE

```
465 /* Chat constraints */
466 ALTER TABLE Chat
467 ADD CONSTRAINT fk_chat_partner
468 FOREIGN KEY (chat_partner_id)
469 REFERENCES User(user_id)
470 ON DELETE CASCADE ON UPDATE CASCADE;
471
474 ALTER TABLE Chat
475 ADD CONSTRAINT fk_chat_owner
476 FOREIGN KEY (owner_user_id)
477 REFERENCES User(user_id)
478 ON DELETE CASCADE ON UPDATE CASCADE;
```

```
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)
```

Test Case

```
259 /* Chat & Message data
260 - two examples, the first one will display general details of the chat, the second text will
261 return all messages of a given chat
262 */
263 CREATE VIEW iu_chat_details_view AS
264 SELECT
265     C.*,
266     UO.legal_name AS owning_user_name,
267     UP.legal_name AS partner_guest_name,
268     COUNT(M.message_id) AS message_count
269 FROM Chat C
270 JOIN User UO ON C.owner_user_id = UO.user_id
271 JOIN User UP ON C.chat_partner_id = UP.user_id
272 LEFT JOIN Message M ON M.owning_chat_id = C.chat_id
273 GROUP BY C.chat_id, UO.legal_name, UP.legal_name;
274
275 -- usage example of view
276 SELECT * FROM iu_chat_details_view WHERE chat_id = 1;
277
278 CREATE VIEW iu_chat_messages_view AS
279 SELECT
280     M.message_id,
281     M.text,
282     M.image_id,
283     C.chat_id AS owning_chat_id_ref
284 FROM Message M
285 JOIN Chat C ON C.chat_id = M.owning_chat_id;
286
287 -- usage example of view
288 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



# TABLE - MESSAGE

## Test Case

- 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.

```
259 /* Chat & Message data
260 - two examples, the first one will display general details of the chat, the second text will
261 return all messages of a given chat
262 */
263 CREATE VIEW iu_chat_details_view AS
264 SELECT
265     C.*,
266     UO.legal_name AS owning_user_name,
267     UP.legal_name AS partner_guest_name,
268     COUNT(M.message_id) AS message_count
269 FROM Chat C
270 JOIN User UO ON C.owner_user_id = UO.user_id
271 JOIN User UP ON C.chat_partner_id = UP.user_id
272 LEFT JOIN Message M ON M.owning_chat_id = C.chat_id
273 GROUP BY C.chat_id, UO.legal_name, UP.legal_name;
274
275 -- usage example of view
276 SELECT * FROM iu_chat_details_view WHERE chat_id = 1;
277
278 CREATE VIEW iu_chat_messages_view AS
279 SELECT
280     M.message_id,
281     M.text,
282     M.image_id,
283     C.chat_id AS owning_chat_id_ref
284 FROM Message M
285 JOIN Chat C ON C.chat_id = M.owning_chat_id;
286
287 -- usage example of view
288 SELECT * FROM iu_chat_messages_view WHERE owning_chat_id_ref = 1;
```

```
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)
```

Message	
PK	message_id
	text
	created
FK	image_id
FK	author_user_id
FK	owning_chat_id

## INSERT

```
1643 /* Message
1644 - the messages hold the individual text of a message and their owning user, they reference the owning chat,
1645 - their timestamp is used to recreate the chat when required
1646 */
1647 INSERT INTO Message (text, author_user_id, owning_chat_id) VALUES ("Content for message 1", 1, 1);
1648
1649 /* examples with images - the same user_id must be used for the image and the chat message */
1650
1651 INSERT INTO Image (uploaded_by_user_id, image_url)
1652 VALUES (1, "https://airbnb.com/images/chat_message_image1.jpg");
1653
1654 SET @message_image_id = LAST_INSERT_ID();
1655
1656 INSERT INTO Message (image_id, author_user_id, owning_chat_id) VALUES (@message_image_id, 9, 12);
```

## CREATE

```
250 CREATE TABLE Message (
251     message_id INT AUTO_INCREMENT PRIMARY KEY,
252     text TEXT,
253     image_id INT, /*FK*/
254     author_user_id INT, /*FK*/
255     owning_chat_id INT, /*FK*/
256     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
257 );
258
259 /* Message constraints */
260 ALTER TABLE Message
261 ADD CONSTRAINT fk_message_owningchat
262 FOREIGN KEY (owning_chat_id)
263 REFERENCES Chat(chat_id)
264 ON DELETE CASCADE ON UPDATE CASCADE;
```

```
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

```
184 CREATE TABLE EmergencyContact (
185     emergencycontact_id INT AUTO_INCREMENT PRIMARY KEY,
186     name VARCHAR(255) NOT NULL,
187     relationship VARCHAR(255),
188     preferred_language_id INT DEFAULT 1, /*FK*/
189     email VARCHAR(255),
190     country_code VARCHAR (32),
191     phone_number VARCHAR(15) NOT NULL,
192     owning_user_id INT NOT NULL /*FK*/
193 )
194
195 /* EmergencyContact constraints */
196
407 ALTER TABLE EmergencyContact
408 ADD CONSTRAINT fk_econtact_language
409 FOREIGN KEY (preferred_language_id)
410 REFERENCES Language(language_id)
411 ON DELETE CASCADE ON UPDATE CASCADE;
412
413 ALTER TABLE EmergencyContact
414 ADD CONSTRAINT fk_econtact_user
415 FOREIGN KEY (owning_user_id)
416 REFERENCES User(user_id)
417 ON DELETE CASCADE ON UPDATE CASCADE;
```

EmergencyContact	
PK	<u>emergencycontact_id</u>
	name
	relationship
FK	preferred_language_id
	email
	country_code
	phone_number
FK	owning_user_id

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

INSERT

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

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

COUNT

# TABLE - WISHLIST

- 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 checks the relation of a wishlist and its propertylistings.

INSERT

```
1837 /* Wishlist
1838 - the wishlist is the relation that stores the data relevant to the wishlists
1839 - the wishlist is referenced by the wishlist_propertylisting relation to link the properties in the list with the list itself
1840 - similar to the chat/messages, it is hard to visualize the wishlist tables without the frontend
1841 */
1842 INSERT INTO Wishlist (name, private, owning_user_id) VALUES ("wishlist 1", TRUE, 1);
```

CREATE

```
171 CREATE TABLE Wishlist (
172     wishlist_id INT AUTO_INCREMENT PRIMARY KEY,
173     name VARCHAR(255) NOT NULL,
174     private BOOLEAN NOT NULL DEFAULT TRUE,
175     owning_user_id INT NOT NULL /*FK*/
176 );
386 /* Wishlist constraints */
387 ALTER TABLE Wishlist
388 ADD CONSTRAINT fk_wishlist_guest
389 FOREIGN KEY (owning_user_id)
390 REFERENCES Guest(guest_id)
391 ON DELETE CASCADE ON UPDATE CASCADE;
```

Test Case

```
229 /* Wishlist data
230 - test case that shows the wishlist, user and property listing data to proves the proper
231 implementation of links between them. Goal is to see the relationship of a user owning
232 a wishlist, which in turn 'owns' (multiple) propertylistings.
233 */
234 CREATE VIEW iu_wishlist_details_view AS
235 SELECT
236     W.*,
237     U.legal_name AS owning_user_name,
238     PL.name AS property_listing_name
239 FROM
240     Wishlist W
241 JOIN User U ON W.owning_user_id = U.user_id
242 JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
243 JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;
244
245 -- usage example of view
246 SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;
247
248 -- this view gets all data regarding the property listings that are in a given wishlist
249 CREATE VIEW iu_wishlist_propertylistings_view AS
250 SELECT
251     W.wishlist_id,
252     PL.*
253 FROM Wishlist W
254 JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
255 JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;
256
257 -- usage example of view
258 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_name |
+-----+-----+-----+-----+-----+-----+
| 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

## Test Case

- 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.

## INSERT

```
1864  /* Wishlist_propertylisting
1865  - this is a relation table that links the wishlists and the properties together
1866  - the wishlist_id references the previous Wishlist relation
1867  */
1868  INSERT INTO Wishlist_PropertyListing (propertylisting_id, wishlist_id) VALUES (1, 1);
```

## CREATE

```
178 CREATE TABLE Wishlist_PropertyListing (
179     wishlist_propertylisting_id INT AUTO_INCREMENT PRIMARY KEY,
180     propertylisting_id INT, /*FK*/
181     wishlist_id INT /*FK*/
182 );

393 /* Wishlist_PropertyListing constraints */
394 ALTER TABLE Wishlist_PropertyListing
395 ADD CONSTRAINT fk_wishlist_property
396 FOREIGN KEY (propertylisting_id)
397 REFERENCES PropertyListing(propertylisting_id)
398 ON DELETE CASCADE ON UPDATE CASCADE;
399
400 ALTER TABLE Wishlist_PropertyListing
401 ADD CONSTRAINT fk_wishlist_wishlist
402 FOREIGN KEY (wishlist_id)
403 REFERENCES Wishlist(wishlist_id)
404 ON DELETE CASCADE ON UPDATE CASCADE;
```

```
229 /* Wishlist data
230 - test case that shows the wishlist, user and property listing data to proves the proper
231 implementation of links between them. Goal is to see the relationship of a user owning
232 a wishlist, which in turn 'owns' (multiple) propertylistings.
233 */
234 CREATE VIEW iu_wishlist_details_view AS
235 SELECT
236     W.*,
237     U.legal_name AS owning_user_name,
238     PL.name AS property_listing_name
239 FROM
240     Wishlist W
241 JOIN User U ON W.owning_user_id = U.user_id
242 JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
243 JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;

244
245 -- usage example of view
246 SELECT * FROM iu_wishlist_details_view WHERE wishlist_id = 1;
247
248 -- this view gets all data regarding the property listings that are in a given wishlist
249 CREATE VIEW iu_wishlist_propertylistings_view AS
250 SELECT
251     W.wishlist_id,
252     PL.*
253 FROM Wishlist W
254 JOIN Wishlist_PropertyListing WPL ON W.wishlist_id = WPL.wishlist_id
255 JOIN PropertyListing PL ON WPL.propertylisting_id = PL.propertylisting_id;
256
257 -- usage example of view
258 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)



```
mysql> SELECT COUNT(*) FROM PropertyListing;
```

COUNT(*)
20

1 row in set (0.01 sec)

COUNT

# TABLE - PROPERTYLISTING

## Test Case

- 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

```
72 CREATE TABLE PropertyListing (  
73     propertylisting_id INT AUTO_INCREMENT PRIMARY KEY,  
74     owning_host_id INT NOT NULL, /*FK*/  
75     name VARCHAR(255) NOT NULL,  
76     address_id INT UNIQUE NOT NULL, /*FK*/  
77     property_type_id INT, /*FK*/  
78     bedroom_count INT,  
79     bathroom_count INT,  
80     available_booking_slots INT,  
81     price_per_night DECIMAL(10,2),  
82     currency_id INT NOT NULL DEFAULT 1, /*FK*/  
83     check_in_time TIME DEFAULT '14:00',  
84     check_out_time TIME DEFAULT '10:00',  
85     average_rating DECIMAL(2,1) DEFAULT 0, /* should get updated by the application when a new review is published, is initially 0 */  
86     total_nr_of_ratings INT DEFAULT 0, /* should get updated/incremented by the application when a new review is published, is initially 0 */  
87  
88     FOREIGN KEY (owning_host_id) REFERENCES Host(host_id)  
89 );
```

## INSERT

```
1690 /* PropertyListing  
1691 - the propertylisting is referenced by the previously created tables for the relevant category, amenities, and house rules  
1692 - the structure of the inserts is fairly simple, we first insert the address data and then use the insert id for the propertylisting  
1693 - for the other attributes we simply enter semi-random dummy data for testing/demonstration purposes  
1694 */  
1695 -- PropertyListing 1  
1696 INSERT INTO Address (country, state, city, street, house_nr, zip_code, address_type)  
1697 VALUES ("United States", "New York", "Brooklyn", "Elm Street", "1234", "11201", "property");  
1698 SET @property_address_id = LAST_INSERT_ID();  
1699 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)  
1700 VALUES (1, "Cozy Studio Apartment", @property_address_id, 1, 1, 1, 2, 50.00, 1, "15:00:00", "11:00:00");
```

```
91 /* relevant property listing data  
92 - this first view shows the data attributes that somewhat directly relate to the propertylisting while  
93 the the following view will serve as an example for one of the tables that is linked via another table.  
94 In that case we will use the amenities table to demonstrate the proper relationship between a  
95 propertylisting and amenities, categories and house rules.  
96 - includes property_type relation test  
97 */  
98 CREATE VIEW iu_propertylisting_view AS  
99 SELECT  
100     PL.propertylisting_id,  
101     PL.name AS propertylisting_name,  
102     PT.name AS property_type,  
103     PL.price_per_night,  
104     C.name AS currency,  
105     A.country,  
106     A.state,  
107     A.city,  
108     A.street,  
109     A.house_nr,  
110     A.zip_code,  
111     H.host_id AS host_id,  
112     HU.legal_name AS host_name  
113 FROM  
114     PropertyListing PL  
115 JOIN Address A ON PL.address_id = A.address_id  
116 JOIN Host H ON PL.owning_host_id = H.host_id  
117 JOIN User HU ON H.user_id = HU.user_id  
118 JOIN Currency C ON PL.currency_id = C.currency_id  
119 JOIN PropertyType PT ON PL.property_type_id = PT.property_type_id;  
120  
121 -- usage example of view  
122 SELECT * FROM iu_propertylisting_view WHERE propertylisting_id = 1;  
123 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

```
91 CREATE TABLE PropertyReview (
92     propertyreview_id INT AUTO_INCREMENT PRIMARY KEY,
93     property_id INT NOT NULL, /*FK*/
94     author_guest_id INT NOT NULL, /*FK*/
95     cleanliness_score INT,
96     accuracy_score INT,
97     check_in_score INT,
98     communication_score INT,
99     location_score INT,
100     value_score INT,
101     comment VARCHAR(2000),
102     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
103 );
```

```
310 ALTER TABLE PropertyReview
311 ADD CONSTRAINT fk_review_property
312 FOREIGN KEY (property_id)
313 REFERENCES PropertyListing(propertylisting_id)
314 ON DELETE CASCADE ON UPDATE CASCADE;
315
316 ALTER TABLE PropertyReview
317 ADD CONSTRAINT fk_review_author
318 FOREIGN KEY (author_guest_id)
319 REFERENCES Guest(guest_id)
320 ON DELETE CASCADE ON UPDATE CASCADE;
```

```
163 /* PropertyReviews are very similar to the UserReviews above, i will therefore not go into much detail
164 about the structure of the query / test case
165 - the reason i constrained the query is that it would otherwise be very bloated with less readability
166 */
167 CREATE VIEW iu_propertyreviews_view AS
168 SELECT
169     PL.propertylisting_id, -- left in to show that PL.id and PR. id are equal -> functioning relation
170     PL.name,
171     PR.*
172 FROM PropertyListing PL
173 JOIN PropertyReview PR ON PL.propertylisting_id = PR.property_id;
174
175 -- usage example of view
176 SELECT * FROM iu_propertyreviews view WHERE propertylisting id = 1;
```

Test Case

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

INSERT

```
2055 /* Property Review
2056 - the propertyreview table is very simple, in that it only contains two foreign keys, which are easy to explain
2057 - the property_id is the property the review is about, while the author_guest_id is the guest who published the review
2058 - the other attributes are simple and obvious
2059 */
2060 -- PropertyReview 1
2061 INSERT INTO PropertyReview (property_id, author_guest_id, cleanliness_score, accuracy_score, check_in_score, communication_score, location_score, value_score, comment)
2062 VALUES (1, 1, 4, 4, 5, 4, 5, 4, "We had a wonderful stay at this place!");
```

```
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		Property_Amenity		Property_HouseRule	
PK	<u>property_category_id</u>	PK	<u>property_amenity_id</u>	PK	<u>property_houserule_id</u>
FK	property_id	FK	property_id	FK	property_id
FK	category_id	FK	amenity_id	FK	houserule_id

```
133  /* this test case shows the selected amenities for a certain propertylisting
134  - this same structure will work for categories and houserules as well, which will not be included as
135  they function identically to this view (with different names, etc.)
136  - i have essentially decided to view amenities, categories and houserules as an equivalence class
137  and chose Amenity as the representative. this can reduce testing workload/effort
138  (this can/could be reasoned as trying to reduce testing costs for example)
139  */
140  CREATE VIEW iu_propertylisting_amenities_view AS
141  SELECT
142      PL.propertylisting_id,
143      PL.name AS property_name,
144      A.name AS amenity_name
145  FROM PropertyListing PL
146  JOIN Property_Amenity PA ON PL.propertylisting_id = PA.property_id
147  JOIN Amenity A ON PA.amenity_id = A.amenity_id;
148
149  -- usage example of view
150  SELECT * FROM iu_propertylisting_amenities_view WHERE propertylisting_id = 1;
```

Test Case

```
119  CREATE TABLE Property_Amenity (
120      property_amenity_id INT AUTO_INCREMENT PRIMARY KEY,
121      property_id INT, /*FK*/
122      amenity_id INT /*FK*/
123  );
135  /* Property_Amenity constraints */
136  ALTER TABLE Property_Amenity
137  ADD CONSTRAINT fk_amenity_property
138  FOREIGN KEY (property_id)
139  REFERENCES PropertyListing(propertylisting_id)
140  ON DELETE CASCADE ON UPDATE CASCADE;
143  ALTER TABLE Property_Amenity
144  ADD CONSTRAINT fk_amenity_amenity
145  FOREIGN KEY (amenity_id)
146  REFERENCES Amenity(amenity_id)
147  ON DELETE CASCADE ON UPDATE CASCADE;
```

INSERT

CREATE

```
1917  /* Property_Amenity
1918  - this relation links the properties with their respective amenities
1919  - each propertylisting will essentially have multiple entries with its id in this table,
1920  each pointing at a the different amenities selected in the propertylisting
1921  */
1922  INSERT INTO Property_Amenity (property_id, amenity_id) VALUES (1,1);
```

```
mysql> SELECT * FROM iu_propertylisting_amenities_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

```
125 CREATE TABLE Amenity (  
126     amenity_id INT AUTO_INCREMENT PRIMARY KEY,  
127     name VARCHAR(255),  
128     icon_url VARCHAR(1024),  
129     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
130     last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
131 );
```

### INSERT

```
1890 /* Amenity  
1891 - this relation is essentially a list of all the amenities that are available  
1892 - this table will not be changed often, and if it does then it will likely be an admin  
1893 - i selected some example amenities that actually exist in the airbnb application to serve as dummy data  
1894 */  
1895 INSERT INTO Amenity (name, icon_url) VALUE ("Kitchen", "https://airbnb.com/images/amenity_icon1.ico");
```

Category		Amenity		HouseRule	
PK	<u>category_id</u>	PK	<u>amenity_id</u>	PK	<u>houserule_id</u>
	name		name		name
	icon_url		icon_url		icon_url
	created		created		created
	last_modified		last_modified		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 test case on previous slide.



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

\*explained in the comment of the insert statement (2<sup>nd</sup> bullet point)

# TABLE - PROPERTYTYPE

## CREATE

```
147 CREATE TABLE PropertyType (
148     propertytype_id INT AUTO_INCREMENT PRIMARY KEY,
149     name VARCHAR(255),
150     icon_url VARCHAR(1024),
151     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
152     last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
153 );
```

PropertyType	
PK	<u>propertytype_id</u>
	name
	icon_url
	created
	last_modified

## INSERT

```
2044 /* PropertyType
2045 - refer to the property_amenity relation above for an explanation
2046 - this is the only table i cannot insert 20 data entries with a good conciseness as there are not enough propertytypes that exist in the application.
2047 i hope it is acceptable to not make add duplicates or unsensible entries
2048 */
2049 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 ‘PropertyType’ 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.

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

COUNT

# TABLE - BOOKING

- 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

```
155 CREATE TABLE Booking (
156   booking_id INT AUTO_INCREMENT PRIMARY KEY,
157   total_price DECIMAL(10,2) NOT NULL,
158   currency_id INT DEFAULT 1, /*FK*/
159   booking_start_date TIMESTAMP NOT NULL,
160   booking_end_date TIMESTAMP NOT NULL,
161   confirmation_code VARCHAR(16) NOT NULL,
162   cancelled BOOLEAN NOT NULL,
163   cancellation_date DATE,
164   refund_amount DECIMAL(10,2),
165   propertylisting_id INT NOT NULL, /*FK*/
166   host_id INT NOT NULL, /*FK*/
167   guest_id INT NOT NULL, /*FK*/
168   created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
169 );
```

```
361 /* Booking constraints */
362 ALTER TABLE Booking
363 ADD CONSTRAINT fk_booking_currency
364 FOREIGN KEY (currency_id)
365 REFERENCES Currency(currency_id)
366 ON DELETE CASCADE ON UPDATE CASCADE;
367
368 ALTER TABLE Booking
369 ADD CONSTRAINT fk_booking_propertylisting
370 FOREIGN KEY (propertylisting_id)
371 REFERENCES Propertylisting(propertylisting_id)
372 ON DELETE CASCADE ON UPDATE CASCADE;
373
374 ALTER TABLE Booking
375 ADD CONSTRAINT fk_booking_host
376 FOREIGN KEY (host_id)
377 REFERENCES Host(host_id)
378 ON DELETE CASCADE ON UPDATE CASCADE;
379
380 ALTER TABLE Booking
381 ADD CONSTRAINT fk_booking_guest
382 FOREIGN KEY (guest_id)
383 REFERENCES Guest(guest_id)
384 ON DELETE CASCADE ON UPDATE CASCADE;
```

## Test Case

```
187 /* Booking data
188 - this test case serves the purpose of testing the relationships relevant to a booking table entry
189 */
190 CREATE VIEW iu_booking_view AS
191 SELECT
192   B.booking_id AS booking_id,
193   B.propertylisting_id,
194   PL.name AS propertylisting_name,
195   H.host_id AS host_id,
196   HU.legal_name AS host_legal_name,
197   G.guest_id AS guest_id,
198   GU.legal_name AS guest_legal_name
199 FROM Booking B
200 JOIN Propertylisting PL ON B.propertylisting_id = PL.propertylisting_id
201 JOIN Host H ON B.host_id = H.host_id
202 JOIN User HU ON HU.user_id = H.user_id
203 JOIN Guest G ON B.guest_id = G.guest_id
204 JOIN User GU ON GU.user_id = G.user_id;
205
206 -- usage examples of view (either look up booking via the transaction, or propertylisting)
207 SELECT * FROM iu_booking_view WHERE booking_id = 1;
208 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

## INSERT

```
2141 /* Booking
2142 - the booking relation holds data related to the booking of a property by a guest
2143 - the cancellation data is obviously optional and can be NULL until updated if a cancellation occurs
2144 */
2145 -- Booking 1
2146 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)
2147 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 * 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)
```



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

COUNT

## TABLE - BANKINFORMATION

### CREATE

```
233 CREATE TABLE BankInformation (
234     bankinformation_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
235     owning_host_id INT NOT NULL, /*FK*/
236     name VARCHAR(128) NOT NULL,
237     account_nr VARCHAR(32) NOT NULL,
238     code VARCHAR(64) NOT NULL,
239     address_id INT NOT NULL /*FK*/
240 );
```

```
452 /* BankInformation constraints */
453 ALTER TABLE BankInformation
454 ADD CONSTRAINT fk_bankinfo_host
455 FOREIGN KEY (owning_host_id)
456 REFERENCES Host(host_id)
457 ON DELETE CASCADE ON UPDATE CASCADE;
458
459 ALTER TABLE BankInformation
460 ADD CONSTRAINT fk_bankinfo_address
461 FOREIGN KEY (address_id)
462 REFERENCES Address(address_id)
463 ON DELETE CASCADE ON UPDATE CASCADE;
```

BankInformation	
PK	<u>bankinformation_id</u>
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

```
224 CREATE TABLE CreditCardInformation (
225     creditcardinformation_id INT AUTO INCREMENT PRIMARY KEY,
226     owning_guest_id INT NOT NULL, /*FK*/
227     bank VARCHAR(128) NOT NULL,
228     card_number VARCHAR(64) NOT NULL,
229     cvc_code VARCHAR(3) NOT NULL,
230     expiration_date DATE NOT NULL
231 );
```

```
445 /* CreditCardInformation constraints */
446 ALTER TABLE CreditCardInformation
447 ADD CONSTRAINT fk_ccinformation_guest
448 FOREIGN KEY (owning_guest_id)
449 REFERENCES Guest(guest_id)
450 ON DELETE CASCADE ON UPDATE CASCADE;
```

CreditCardInformation	
PK	<u>creditcardinformation_id</u>
FK	owning_guest_id
	bank
	card_number
	cvc_code
	expiration_date

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.



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

COUNT

# TABLE - TRANSACTION

- 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.

Test Case

```
215 CREATE TABLE Transaction (
216   transaction_id INT AUTO_INCREMENT PRIMARY KEY,
217   booking_id INT NOT NULL, /*FK*/
218   bankinformation_id INT, /*FK*/
219   creditcardinformation_id INT, /*FK*/
220   created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
221   last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
222 );
```

CREATE

```
426 /* Transaction constraints */
427 ALTER TABLE Transaction
428 ADD CONSTRAINT fk_transaction_booking
429 FOREIGN KEY (booking_id)
430 REFERENCES Booking(booking_id)
431 ON DELETE CASCADE ON UPDATE CASCADE;
432
433 ALTER TABLE Transaction
434 ADD CONSTRAINT fk_transaction_bankinformation
435 FOREIGN KEY (bankinformation_id)
436 REFERENCES BankInformation(bankinformation_id)
437 ON DELETE CASCADE ON UPDATE CASCADE;
438
439 ALTER TABLE Transaction
440 ADD CONSTRAINT fk_transaction_creditcard
441 FOREIGN KEY (creditcardinformation_id)
442 REFERENCES CreditCardInformation(creditcardinformation_id)
443 ON DELETE CASCADE ON UPDATE CASCADE;
```

```
210 /* Transaction, CreditCard, Bankinformation data
211 - Test case that shows the relation between transactions with payment informations, such as
212 credit cards or banks and the corresponding booking
213 - (this test case will be seen as sufficient for testing the proper implementation of the tables that are
214 a part of it)
215 */
216 CREATE VIEW iu_transaction_view AS
217 SELECT
218   T.*,
219   B.booking_id AS Booking_id_ref,
220   CCI.creditcardinformation_id AS creditcard_id,
221   CCI.card_number AS creditcard_number,
222   BI.bankinformation_id AS bankinfo_id,
223   BI.account_nr AS bank_account_number
224 FROM Transaction T
225 JOIN Booking B ON T.booking_id = B.booking_id
226 JOIN CreditCardInformation CCI ON T.creditcardinformation_id = CCI.creditcardinformation_id
227 JOIN BankInformation BI ON T.bankinformation_id = BI.bankinformation_id;
228
229 -- usage examples of view (either look up data via the transaction, or booking)
230 SELECT * FROM iu_transaction_view WHERE transaction_id = 1;
231 SELECT * FROM iu_transaction_view WHERE booking_id = 1;
```

Transaction	
PK	<u>transaction_id</u>
FK	booking_id
FK	bankinformation_id
FK	creditcardinformation_id
	created
	last_modified

INSERT

```
2226 /* Transaction
2227 - this table again does not make a whole lot of sense without the application actually pulling in the relevant data from their respective tables
2228 - the table is intended to be the base lookup table for purchase transaction data
2229 */
2230 -- Transaction 1
2231 INSERT INTO Transaction (booking_id, bankinformation_id, creditcardinformation_id)
2232 VALUES (1, 2, 3);
```

```
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

```
207 CREATE TABLE GiftCard (
208     giftcard_code VARCHAR(16) PRIMARY KEY,
209     amount DECIMAL(10,2) NOT NULL,
210     currency_id INT NOT NULL DEFAULT 1, /*FK*/
211     valid_until_date TIMESTAMP NOT NULL,
212     created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
213 );
```

GiftCard	
PK	giftcard_code
FK	amount
	currency_id
	valid_until_date
	created

### INSERT

```
80 INSERT INTO GiftCard (giftcard_code, amount, currency_id, valid_until_date)
81 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!