Airbnb Data Mart

Development

Table of Contents

Overview of the Database Structure

Database components

The logical structure of the data mart consists of six components which represent all functionality of the application. The components are split up according to their roles in the overall user experience of a certain user type. Thereby three types of component groups can be classified, namely components concerning host users, guest users or both user types.

Components for Guest and Host Users

The first component of the data mart represents basic administration functionality for user management, the creation and deletion of users. The user management is thereby embedded in the “Users” table, and it’s related SQL statements. The second component is the payment system, which core features are realized with the “Transactions” table and the additional multi-currency and multi-payment gateway support spread across the “PaymentOption” and “Currency” table.

Components for Guest Users

The third component is centered around the search functionality, which can be used by a guest to find a suitable lodging. Search queries can be performed by issuing detailed criteria concerning a lodging like guest capacity, but also by making a geographical grid search. The former is handled with the “Lodging” and the latter with the help of the “Location” table. The search functionality also encompasses the search for nearby locations, which is achieved by leveraging the “Location” table. In the scope of this application the nearby locations are restricted to a small selection of sights and public transport locations which are stored in the “PublicTransport” and “Sight” tables. The fourth component for reservation management is represented by the “Booking” table. The logic related to the table stores the reservations and prevents possible conflicts with other reservations for the same lodging. The fifth and last component for the guest user type is the review system, with functionality for storing reviews in the “Review” table and calculation of ratings for the individual lodgings.

Components for Host Users

The sixth component of the application is concerned with the core functionality of a host user. This includes the creation and management of lodgings which is spread over several tables. Thereby the “Lodging” table is the main hub for information about a lodging connecting the “Furnishing”, “Room”, “Rule” and “Policy tables via join tables. An additional relation to the “Location” table connects the lodgings to the geographic search functionality.

Implementation Procedure

Set-Up of the Database Structure

The implementation started with the creation of all tables which are specified in the initial ER-Model. After the setup of the core database structure the next part was the collection of appropriate data for the data mart.

Collection of the Data

The collection started with filling the “Currency” and the “PaymentOption” tables. After that the generation of data for the “Users” table with a small python script was the next step. Following was the extraction of cities, states, countries and continents from the MONDIAL database which features an extensive collection of the aforementioned entities.

With data for this tables set-up, the next step was the scraping of data for lodgings, locations, public transports and sights from Airbnb and Google Maps. The last part of the data collection was generating data for the bookings, transactions, furniture, rooms, policies and rules for the lodgings. After all the data was collected and generated, it was imported into the database.

Creation of the Application Logic

The final step of the implementation was the creation of the SQL statements for controlling the application. The development of the procedures started according to their respective place in the application control-flow. First the procedures for user management were implemented. Following was the creation of the code for lodging creation and management. In the next step the search functionality was set-up and finally the procedures for the booking, review and payment system were implemented.

Users

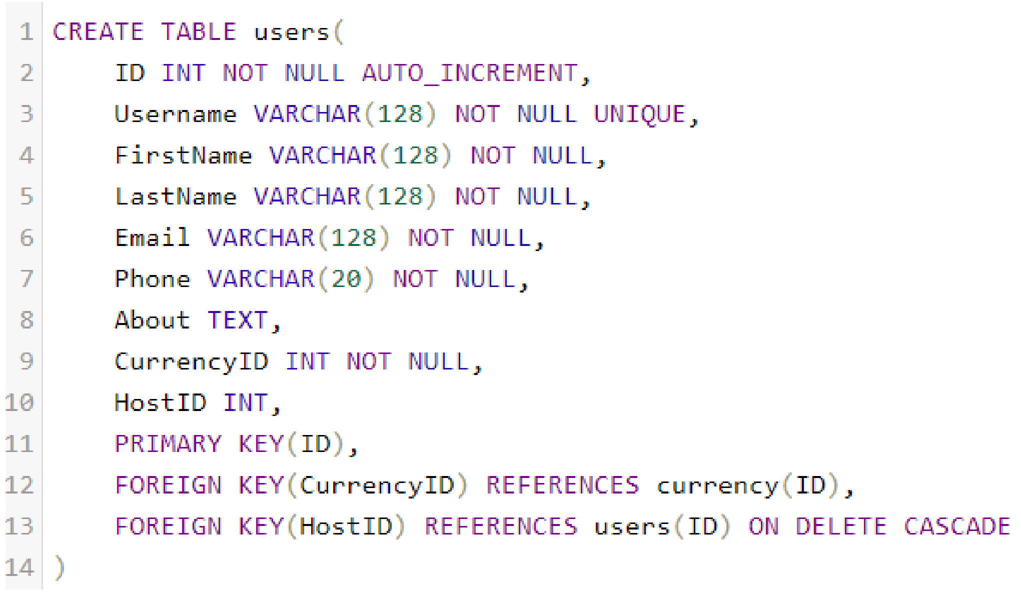
The Users entity is the core of the User Management functionality and features all important information about a user.

The user information consists thereby of:

- A unique username
- First Name
- Last Name
- Email
- Phone Number
- An optional “About” text
- The chosen display currency of the user
- The recursive relation’s ID for distinguishing a host from a guest

The control logic for the user management is thereby embedded in the two procedures related to the “Users” entity. The CreateUser and DeleteAccount procedures.

SQL-Statement



```sql
CREATE TABLE users(
    ID INT NOT NULL AUTO_INCREMENT,
    Username VARCHAR(128) NOT NULL UNIQUE,
    FirstName VARCHAR(128) NOT NULL,
    LastName VARCHAR(128) NOT NULL,
    Email VARCHAR(128) NOT NULL,
    Phone VARCHAR(20) NOT NULL,
    About TEXT,
    CurrencyID INT NOT NULL,
    HostID INT,
    PRIMARY KEY(ID),
    FOREIGN KEY(CurrencyID) REFERENCES currency(ID),
    FOREIGN KEY(HostID) REFERENCES users(ID) ON DELETE CASCADE
)
```

CreateUser Procedure

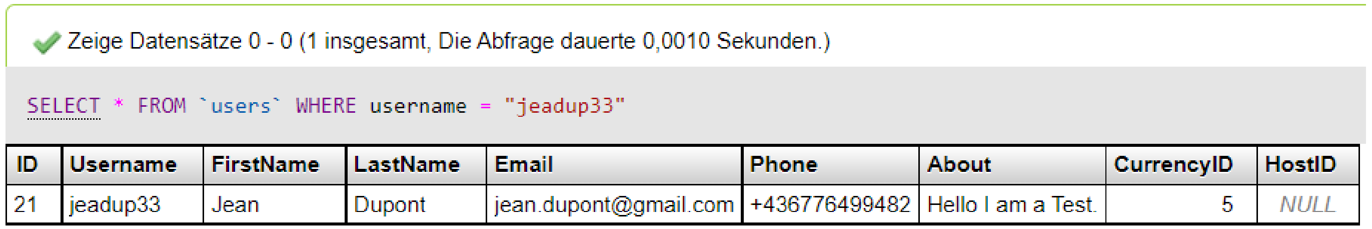
Like the name implies, the CreateUser procedure is used to create a new user account. The procedure takes the users information according to the ER-Model as input parameters and first of all checks if all parameters are issued. After the check, the new user is inserted into the database. It is important to note that the procedure stores each user first as a guest user, which is the standard of the application.

Test Case

To create a user the CreateUser procedure is called with a set of test parameters:

```
SET @p0='jeadup33';
SET @p1='Jean';
SET @p2='Dupont';
SET @p3='jean.dupont@gmail.com';
SET @p4='+436776499482';
SET @p5='Euro';
SET @p6='Hello I am a Test.';
CALL `User_CreateUser`(@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7);
SELECT @p7 AS `message`;
```

The command results in the following new entry in the database:

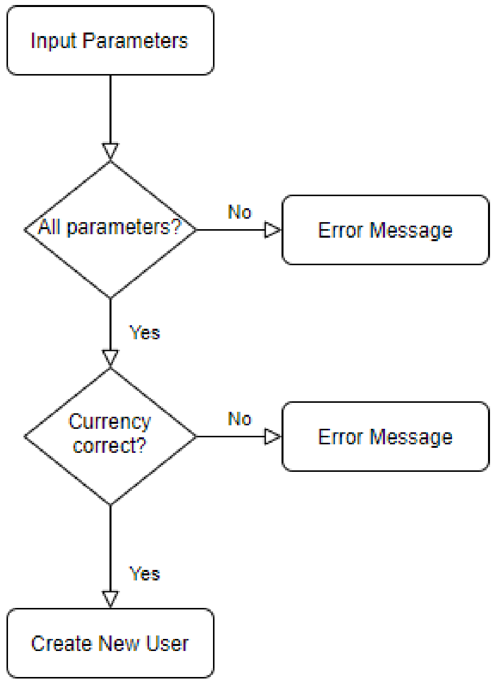


✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0,0010 Sekunden.)

```
SELECT * FROM `users` WHERE username = "jeadup33"
```

ID	Username	FirstName	LastName	Email	Phone	About	CurrencyID	HostID
21	jeadup33	Jean	Dupont	jean.dupont@gmail.com	+436776499482	Hello I am a Test.	5	NULL

Flow Diagram



DeleteAccount Procedure

The DeleteAccount procedure implements the functionality for deleting users. The procedure employs various checks to ensure the integrity of the database after a user is deleted. First it is checked if the issued username parameter can be found in the data mart. After that, the “host” status of the user is queried. If the user is a guest, the procedure checks for unreceived payments and if the check is passed, the user is deleted. For a host user first a check for unclaimed payments is made. If all payments are settled, the procedure first queries all the locations of the host’s lodgings and stores them in a temporary table. Then the host user is deleted and through the foreign key relations all lodgings and related information is deleted from the database. In the last step, the stored locations are also deleted.

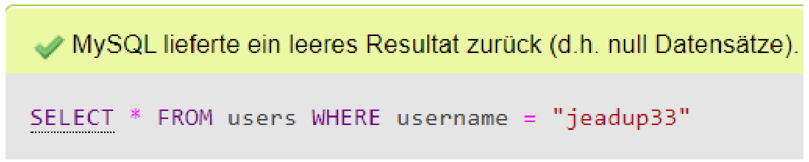
Test Case – User can be deleted

For the test case, the newly created user “jeadup33” is deleted.

The procedure is called with the username as parameter:

```
SET @p0='jeadup33';
CALL `User_DeleteAccount`(@p0, @p1);
SELECT @p1 AS `message`;
```

Checking for the user entry confirms the deletion.



MySQL lieferte ein leeres Resultat zurück (d.h. null Datensätze).

```
SELECT * FROM users WHERE username = "jeadup33"
```

Test Case – User cannot be deleted, because of open transactions

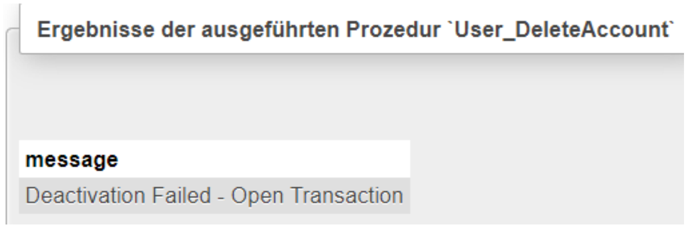
For this test case a guest with open payments is chosen:

```
SELECT users.Username FROM users
INNER JOIN booking ON booking.UsersID = users.ID
INNER JOIN transactions ON transactions.BookingID = booking.ID
WHERE transactions.Received = 0 LIMIT 1;
```

The result is the username “patand16”, which is now used as parameter for the DeleteAccount procedure:

```
SET @p0='patand16'; CALL `User_DeleteAccount`(@p0, @p1); SELECT @p1 AS `message`;
```

and results in an error message:



Ergebnisse der ausgeführten Prozedur `User_DeleteAccount`

```
message
Deactivation Failed - Open Transaction
```

Lodging

The lodging entity is the backbone of the data mart structure and connects all other components.

The lodging entity consists of:

- A description of the lodging
- The category of the lodging, like Hotel or Apartment
- An about text with various information
- The capacity which is the maximum number of allowed guests
- The rating which is the average from all review ratings
- The price per night
- The currency of the host user
- The exact geographical location
- And the host of the lodging

The functionality implemented with the lodging table is also dependent on the several following tables which are used for the storage of the lodging's other attributes and the overall seven related procedures.

SQL-Statement

```sql
CREATE TABLE lodging(
    ID INT NOT NULL AUTO_INCREMENT,
    Description VARCHAR(128) NOT NULL UNIQUE,
    Category VARCHAR(64) NOT NULL,
    About TEXT NOT NULL,
    Capacity INT NOT NULL,
    Rating DECIMAL(2, 1),
    Price DECIMAL(7, 2) NOT NULL,
    CurrencyID INT NOT NULL,
    LocationID INT NOT NULL,
    UsersID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(CurrencyID) REFERENCES currency(ID),
    FOREIGN KEY(LocationID) REFERENCES location(ID),
    FOREIGN KEY(UsersID) REFERENCES users(ID) ON DELETE CASCADE
);
```

Furnishing, Room, Rule and Policy

The following four entities are used to store information which can be used to describe various lodgings. All four tables are based on the same layout, which only stores the description of the corresponding object. The tables are thereby connected to the lodging tables with junction tables. This architecture is chosen, because it reduces the data volume to be stored and also improves the query speed.

Furnishing

The furnishing entity is used to specify furniture and other assets to a lodging, for example WLAN, Microwave or a Shower.

Room

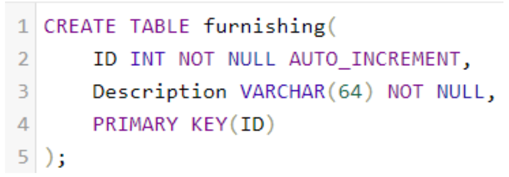
For the information regarding the physical structure of a lodging the room table is employed, in which various possible rooms are predefined.

Rule

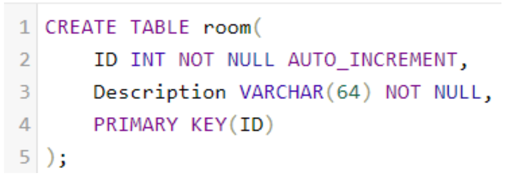
The rule entity holds the possible house rules which a host can assign to a lodging. For example, that no smoking or pets are allowed

Policy

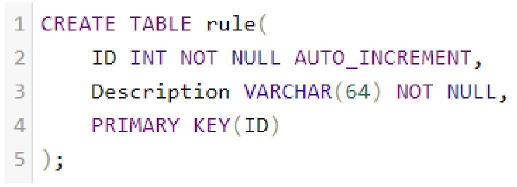
The last of the four attribute tables stores policies for lodgings. For example, information about booking cancellations.



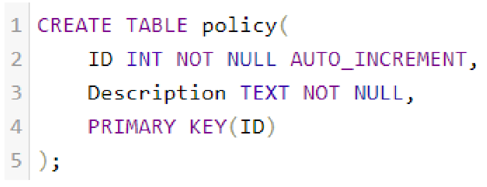
```sql
CREATE TABLE furnishing(
    ID INT NOT NULL AUTO_INCREMENT,
    Description VARCHAR(64) NOT NULL,
    PRIMARY KEY(ID)
);
```



```sql
CREATE TABLE room(
    ID INT NOT NULL AUTO_INCREMENT,
    Description VARCHAR(64) NOT NULL,
    PRIMARY KEY(ID)
);
```



```sql
CREATE TABLE rule(
    ID INT NOT NULL AUTO_INCREMENT,
    Description VARCHAR(64) NOT NULL,
    PRIMARY KEY(ID)
);
```



```sql
CREATE TABLE policy(
    ID INT NOT NULL AUTO_INCREMENT,
    Description TEXT NOT NULL,
    PRIMARY KEY(ID)
);
```

Location

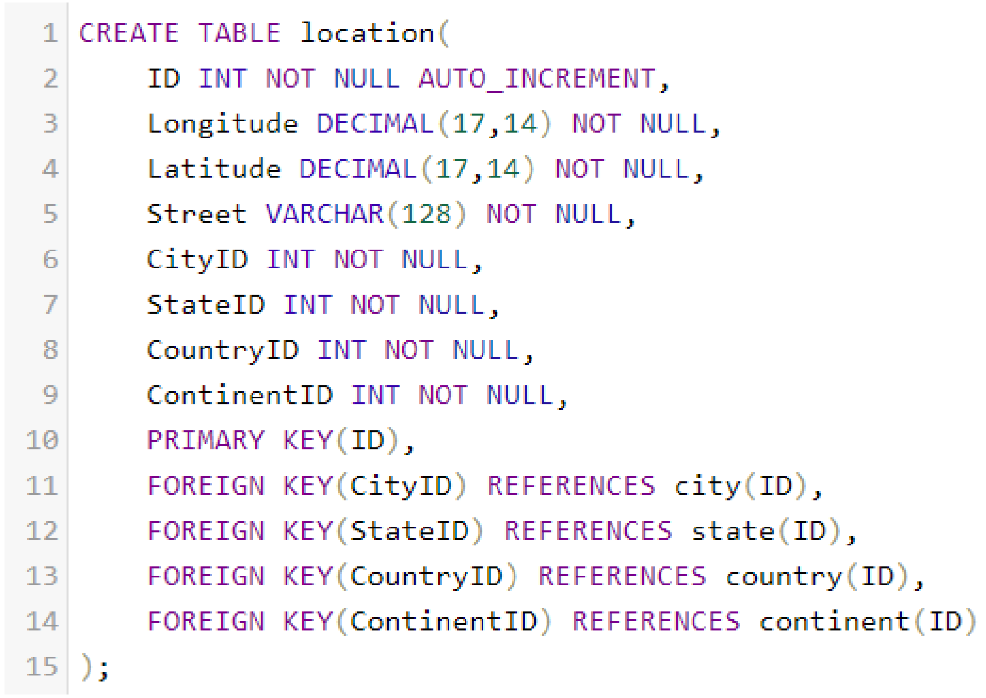
Before the functionality of the lodging management can be examined, first the location entity must be described, as the lodging entity depends on it. The location table is the central storage for geographical information, which connects the lodgings to the geographical search functionality. For adding the longitude and latitude data, Google Maps can be leveraged. To get the coordinates of a location, a right click in Google Maps on the exact location displays the coordinates in the first row of a newly opened small window.

The location entity consists of:

- The longitude of the location in decimal degrees
- The latitude of the location in decimal degrees
- The address of the location in the form of a street name and house number
- The capacity which is the maximum number of allowed guests
- The city in which the location lies
- The location's state
- The location's country
- And the continent

Like for the lodgings, the location entity is depended on several other tables which store the respective cities, states, countries and continents.

SQL-Statement



```sql
CREATE TABLE location(
    ID INT NOT NULL AUTO_INCREMENT,
    Longitude DECIMAL(17,14) NOT NULL,
    Latitude DECIMAL(17,14) NOT NULL,
    Street VARCHAR(128) NOT NULL,
    CityID INT NOT NULL,
    StateID INT NOT NULL,
    CountryID INT NOT NULL,
    ContinentID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(CityID) REFERENCES city(ID),
    FOREIGN KEY(StateID) REFERENCES state(ID),
    FOREIGN KEY(CountryID) REFERENCES country(ID),
    FOREIGN KEY(ContinentID) REFERENCES continent(ID)
);
```

City, State, Country, Continent

This four entities features an extensive collection of cities, states, countries and continents and are used for efficient storage of locations. Like for the lodging's attributes, this architecture was chosen to reduce data volume. All four tables have the same structure which is shown on the right.

City

The city table is used for the storage of cities.

State

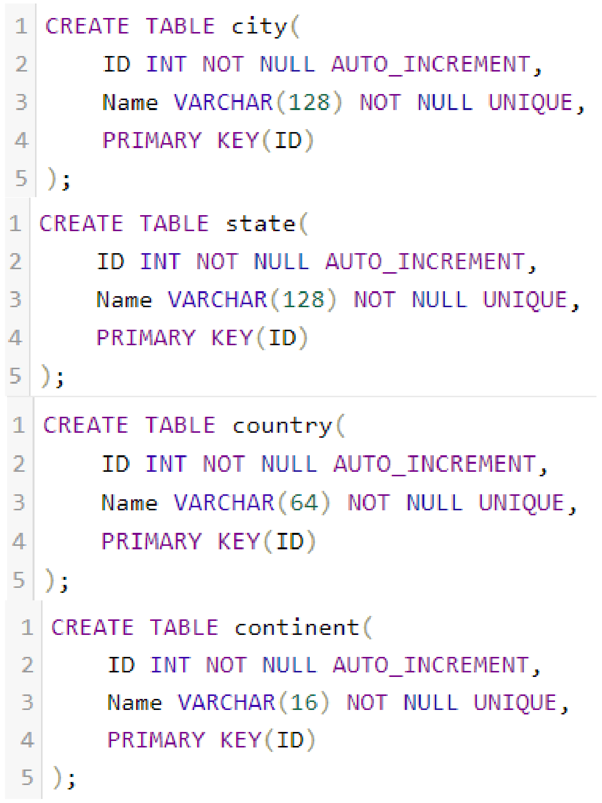
The state entity holds the states and provinces.

Country

The same as the city and state table, but only for countries.

Continent

The last of the four tables stores the continents.



```sql
CREATE TABLE city(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(128) NOT NULL UNIQUE,
    PRIMARY KEY(ID)
);
```

```sql
CREATE TABLE state(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(128) NOT NULL UNIQUE,
    PRIMARY KEY(ID)
);
```

```sql
CREATE TABLE country(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64) NOT NULL UNIQUE,
    PRIMARY KEY(ID)
);
```

```sql
CREATE TABLE continent(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(16) NOT NULL UNIQUE,
    PRIMARY KEY(ID)
);
```

CreateLodging Procedure

Procedure Logic

The CreateLodging procedure is the starting point for the creation of new lodgings. The command takes an extensive set of inputs which are used to supply the core information for the lodging and location entities.

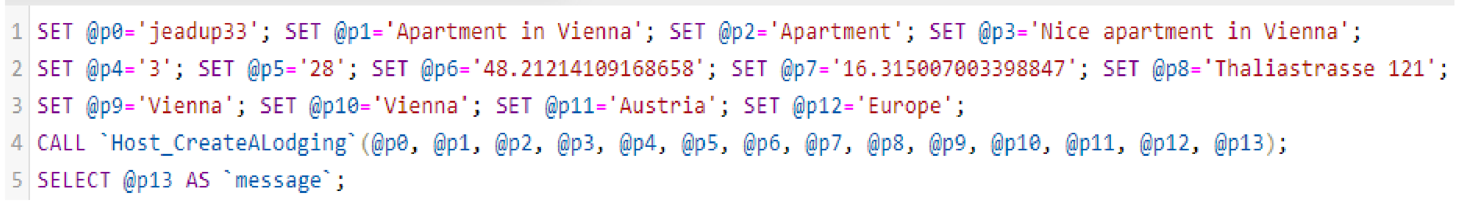
The first step is an initial check if all parameters are issued and if all parameters exist in the data mart. If all checks are passed, the procedure starts a transaction.

First of all, a new location is created with the given parameters. In the second step, the ID of the newly created location is temporarily stored.

Next the new lodging is created and in the last step a check is set-up which updates a user's status from guest to host if the created lodging is the first lodging of the user.

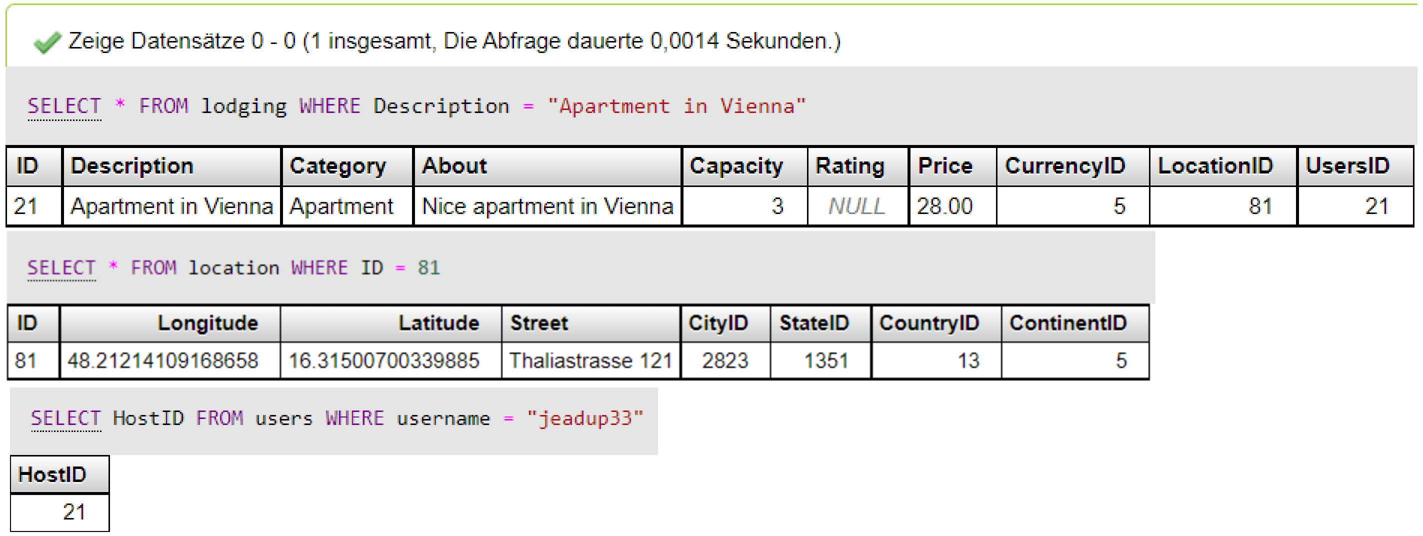
Test Case

To create a new lodging, the CreateLodging procedure is called with a set of test parameters and the newly created user "jeadup33":



```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna'; SET @p2='Apartment'; SET @p3='Nice apartment in Vienna';
SET @p4='3'; SET @p5='28'; SET @p6='48.21214109168658'; SET @p7='16.315007003398847'; SET @p8='Thaliastrasse 121';
SET @p9='Vienna'; SET @p10='Vienna'; SET @p11='Austria'; SET @p12='Europe';
CALL `Host_CreateALodging`(@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7, @p8, @p9, @p10, @p11, @p12, @p13);
SELECT @p13 AS `message`;
```

The command results in the following new entries in the database:



✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0,0014 Sekunden.)

```
SELECT * FROM lodging WHERE Description = "Apartment in Vienna"
```

ID	Description	Category	About	Capacity	Rating	Price	CurrencyID	LocationID	UsersID
21	Apartment in Vienna	Apartment	Nice apartment in Vienna	3	*NULL*	28.00	5	81	21

```
SELECT * FROM location WHERE ID = 81
```

ID	Longitude	Latitude	Street	CityID	StateID	CountryID	ContinentID
81	48.21214109168658	16.31500700339885	Thaliastrasse 121	2823	1351	13	5

```
SELECT HostID FROM users WHERE username = "jeadup33"
```

HostID
21

DeleteLodging Procedure

Procedure Logic

The DeleteLodging procedure is the second part of the lodging administration logic and, like the name implies, is responsible for removing lodgings from the database. The procedure takes two parameters, namely the username and the description of the lodging as input.

Then the procedure starts with an initial check if the entered username is correct, consequently validates if the lodging exists in the database and checks if the user has the right to delete the lodging.

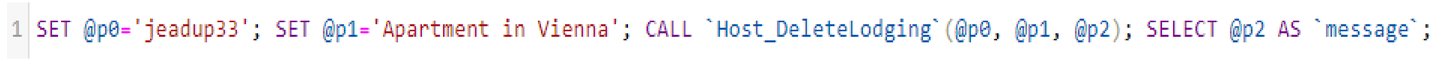
From this point on, there are two possibilities. If there are no bookings for the lodging, there also cannot be open transactions. Therefore, the lodging can be deleted.

Otherwise, if at least one booking exists for the lodging, the procedure checks if all corresponding transactions are closed and deletes the lodging.

Finally, there is a check if the deleted lodging was the last lodging of the host and if this is validated to true, the host status of the user is revoked.

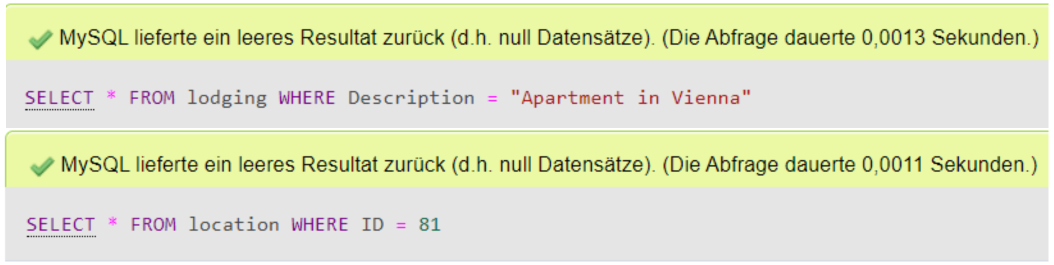
Test Case

To test the procedure first the newly created lodging “Apartment in Vienna” is deleted.

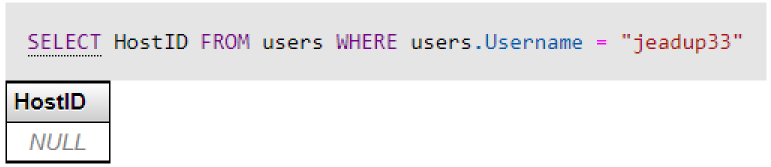


```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna'; CALL `Host_DeleteLodging`(@p0, @p1, @p2); SELECT @p2 AS `message`;
```

A quick search after the lodging and location confirms the deletion.



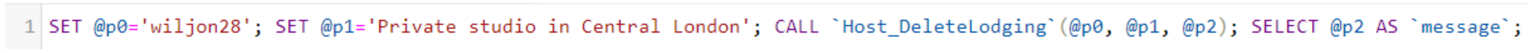
```
✓ MySQL lieferte ein leeres Resultat zurück (d.h. null Datensätze). (Die Abfrage dauerte 0,0013 Sekunden.)

SELECT * FROM lodging WHERE Description = "Apartment in Vienna"

✓ MySQL lieferte ein leeres Resultat zurück (d.h. null Datensätze). (Die Abfrage dauerte 0,0011 Sekunden.)

SELECT * FROM location WHERE ID = 81
```



```
SELECT HostID FROM users WHERE users.Username = "jeadup33"
```

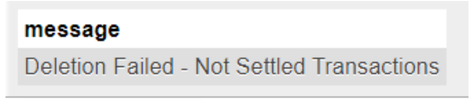
HostID
NULL

To test the control logic for open transactions, the “Private studio in Central London” is set as input parameter. This lodging has unsettled transactions.



```
SET @p0='wiljon28'; SET @p1='Private studio in Central London'; CALL `Host_DeleteLodging`(@p0, @p1, @p2); SELECT @p2 AS `message`;
```

Executing the procedure results in an error message.



message
Deletion Failed - Not Settled Transactions

ManageFurniture, ManagePolicies, ManageRooms and ManageRules Procedures

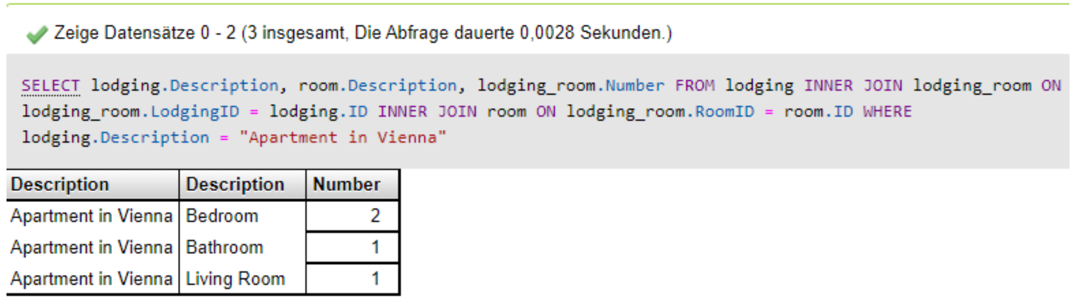
The following procedures are used to add and remove furniture, rooms, policies and rules to a lodging. All four procedures have basically the same inputs, which consists of the host's username, the name of the lodging, the action which can either be "Add" or "Remove" and the items which should be added or removed in a comma separated list. Additionally, the room table takes a comma separated list which must be aligned with the room list and holds the amount of a respective room. Here only the ManageRooms procedure is tested, the other procedures are demonstrated during the test of the search functionality.

Test Case - Adding rooms

For the test case one Bathroom, two Bedrooms and one Living room is added to the "Apartment in Vienna".

```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna';
SET @p2='Bathroom,Bedroom,Living Room'; SET @p3='1,2,1';
SET @p4='Add';
CALL `Host_ManageRooms`(@p0, @p1, @p2, @p3, @p4, @p5);
SELECT @p5 AS `message`;
```

A search for all rooms confirms the results.



✓ Zeige Datensätze 0 - 2 (3 insgesamt, Die Abfrage dauerte 0,0028 Sekunden.)

```
SELECT lodging.Description, room.Description, lodging_room.Number FROM lodging INNER JOIN lodging_room ON
lodging_room.LodgingID = lodging.ID INNER JOIN room ON lodging_room.RoomID = room.ID WHERE
lodging.Description = "Apartment in Vienna"
```

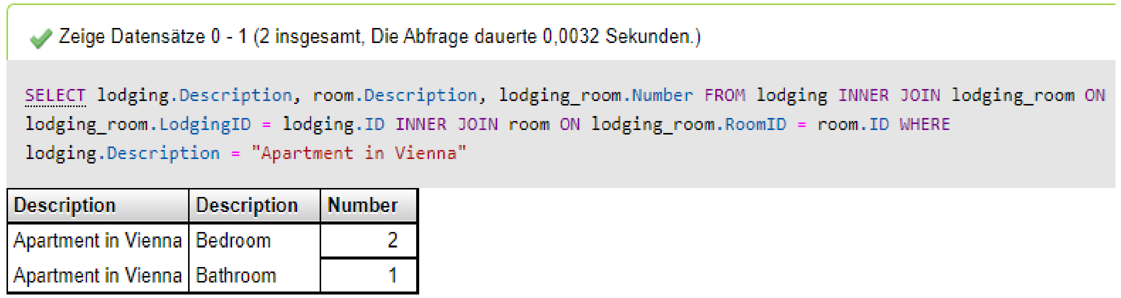
Description	Description	Number
Apartment in Vienna	Bedroom	2
Apartment in Vienna	Bathroom	1
Apartment in Vienna	Living Room	1

Test Case – Removing rooms

For this test case the living room is removed from the apartment.

```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna';
SET @p2='Living Room'; SET @p3=''; SET @p4='Remove';
CALL `Host_ManageRooms`(@p0, @p1, @p2, @p3, @p4, @p5);
SELECT @p5 AS `message`;
```

For removing it is not necessary to specify a number for the room. Executing the same search query as for the "Add" test case gives following results.



✓ Zeige Datensätze 0 - 1 (2 insgesamt, Die Abfrage dauerte 0,0032 Sekunden.)

```
SELECT lodging.Description, room.Description, lodging_room.Number FROM lodging INNER JOIN lodging_room ON
lodging_room.LodgingID = lodging.ID INNER JOIN room ON lodging_room.RoomID = room.ID WHERE
lodging.Description = "Apartment in Vienna"
```

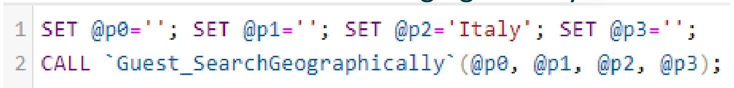
Description	Description	Number
Apartment in Vienna	Bedroom	2
Apartment in Vienna	Bathroom	1

SearchALodging and SearchGeographically Procedure

The first guest user related procedures are used to search for a lodging with the category, capacity and city parameters or with geographical entities like cities, states, countries or continent. It is important that for the SearchGeographically procedure only one entity is chosen, for example only a country and that for the SearchALodging procedure all parameters are issued. Additionally, the SearchALodging procedure displays all lodgings that are greater or equal then the given capacity.

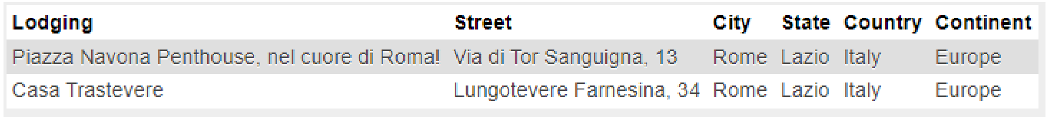
Test Case – Searching for lodgings in Italy

The SearchGeographically procedure can be used with all geographical entities of the data mart. The first search is for lodgings in Italy.



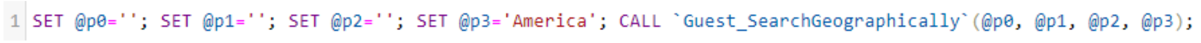
```
SET @p0=''; SET @p1=''; SET @p2='Italy'; SET @p3='';
CALL `Guest_SearchGeographically`(@p0, @p1, @p2, @p3);
```

With the results:



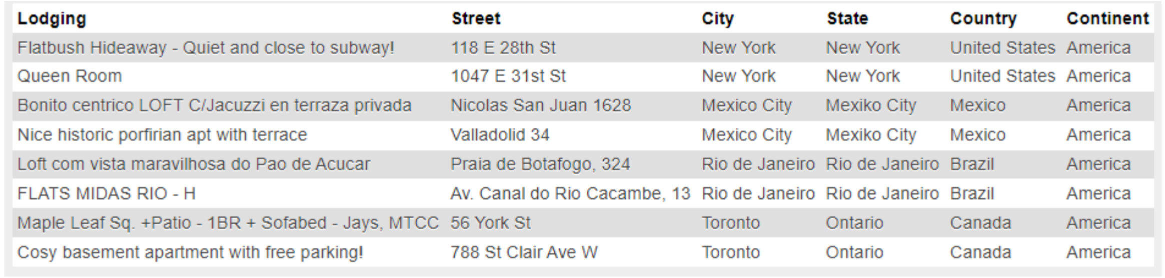
Lodging	Street	City	State	Country	Continent
Piazza Navona Penthouse, nel cuore di Roma!	Via di Tor Sanguigna, 13	Rome	Lazio	Italy	Europe
Casa Trastevere	Lungotevere Farnesina, 34	Rome	Lazio	Italy	Europe

Additionally, a search for lodgings in Amerika is performed.



```
SET @p0=''; SET @p1=''; SET @p2=''; SET @p3='America'; CALL `Guest_SearchGeographically`(@p0, @p1, @p2, @p3);
```

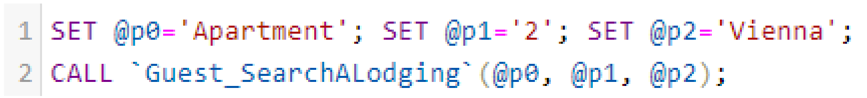
Resulting in:



Lodging	Street	City	State	Country	Continent
Flatbush Hideaway - Quiet and close to subway!	118 E 28th St	New York	New York	United States	America
Queen Room	1047 E 31st St	New York	New York	United States	America
Bonito centrico LOFT C/Jacuzzi en terraza privada	Nicolas San Juan 1628	Mexico City	Mexiko City	Mexico	America
Nice historic porfirian apt with terrace	Valladolid 34	Mexico City	Mexiko City	Mexico	America
Loft com vista maravilhosa do Pao de Acucar	Praia de Botafogo, 324	Rio de Janeiro	Rio de Janeiro	Brazil	America
FLATS MIDAS RIO - H	Av. Canal do Rio Cacambe, 13	Rio de Janeiro	Rio de Janeiro	Brazil	America
Maple Leaf Sq. +Patio - 1BR + Sofabed - Jays, MTCC	56 York St	Toronto	Ontario	Canada	America
Cosy basement apartment with free parking!	788 St Clair Ave W	Toronto	Ontario	Canada	America

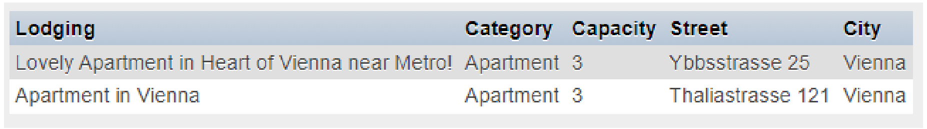
Test Case – Searching for an apartment in Vienna

For testing the SearchALodging procedure, a search for an apartment in Vienna with a capacity for at least two people is conducted.



```
SET @p0='Apartment'; SET @p1='2'; SET @p2='Vienna';
CALL `Guest_SearchALodging`(@p0, @p1, @p2);
```

The search yields two results, an existing entry in the data mart and the newly created test apartment.



Lodging	Category	Capacity	Street	City
Lovely Apartment in Heart of Vienna near Metro!	Apartment	3	Ybbsstrasse 25	Vienna
Apartment in Vienna	Apartment	3	Thaliastrasse 121	Vienna

ShowLodgingDetails Procedure

When a lodging has been found with one of the two search functions, the desired lodging can be viewed in detail with the ShowLodgingDetails procedure. The procedure first checks the input parameters, which are the lodgings description and the username of the guest. Then detailed information, which includes furniture, rooms, rules, policies, the reviews which were left by guests, capacity, category, rating and the price converted into the guest currency, is displayed.

Test Case Preparation

Before testing the procedure, furniture, rules and a policy is added.

For the furniture a shower, microwave and WLAN is added to the test apartment.

```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna'; SET @p2='Shower,Microwave,WLAN';
SET @p3='Add'; CALL `Host_ManageFurniture`(@p0, @p1, @p2, @p3, @p4); SELECT @p4 AS `message`;
```

Additionally, the apartment does not allow pets and kids.

```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna'; SET @p2='No Kids,No Pets';
SET @p3='Add'; CALL `Host_ManageRules`(@p0, @p1, @p2, @p3, @p4); SELECT @p4 AS `message`;
```

And finally, a free cancelation for up to five days before the stay is chosen. It is important to note that for adding a policy the policy ID must be given as parameter, because of the text length.

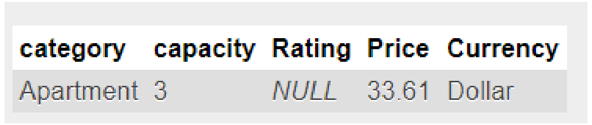
```
SET @p0='jeadup33'; SET @p1='Apartment in Vienna'; SET @p2='2';
SET @p3='Add'; CALL `Host_ManagePolicies`(@p0, @p1, @p2, @p3, @p4);
SELECT @p4 AS `message`;
```

Test Case – Detailed information for the test apartment

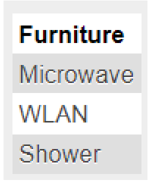
After setting up the test lodging detailed information can be displayed to the user.

```
SET @p0='johwil24'; SET @p1='Apartment in Vienna'; CALL `Guest_ShowLodgingDetails`(@p0, @p1, @p2);
SELECT @p2 AS `message`;
```

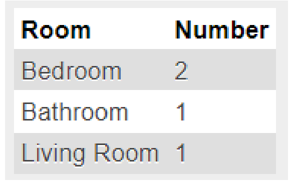
The results are all information that were added previously without a review and a rating, as there are currently are no reviews for the apartment:



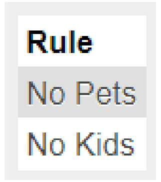
category	capacity	Rating	Price	Currency
Apartment	3	*NULL*	33.61	Dollar



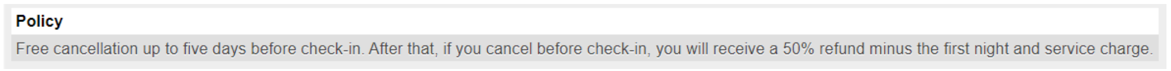
Furniture
Microwave
WLAN
Shower



Room	Number
Bedroom	2
Bathroom	1
Living Room	1



Rule
No Pets
No Kids



Policy
Free cancellation up to five days before check-in. After that, if you cancel before check-in, you will receive a 50% refund minus the first night and service charge.

Sight and PublicTransport

As an additional feature for the guest users, the data mart has built-in functionality for searching nearby locations with coordinates stored in the location table. The locations are thereby limited to famous sights and public transport. In the scope of the data mart project, the sights and public transport entries were limited to two records for each lodging in the test data. There is the possibility to feature a full roaster of all public transport systems, restaurant, sights stores and other locations by leveraging the Google Data Studio software. Nevertheless, the decision was made to only implement a smaller proof-of-concept for the “Nearby Locations System”, because implementing all the available data is outside of the scope of this project.

Sight

The sight entity contains names of famous sights. The sight table is thereby linked to the location table and in turn to the search logic.

SQL-Statement

```sql
CREATE TABLE sight(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(128) NOT NULL UNIQUE,
    LocationID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(LocationID) REFERENCES location(ID)
);
```

PublicTransport

This table is filled with the names of the public transports. The data storage in this entity is intentionally redundant, because when using the Google Data Studio to fill the table, as this would be the case in a production scenario, the public transports all would have different and unique names.

SQL-Statement

```sql
CREATE TABLE publictransport(
    ID INT NOT NULL AUTO_INCREMENT,
    Description VARCHAR(64) NOT NULL,
    LocationID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(LocationID) REFERENCES location(ID)
);
```

SearchNearbyLocations Procedure

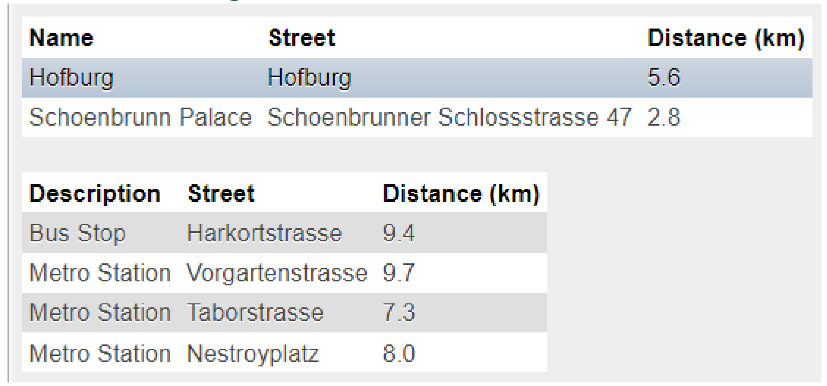
The SearchNeabyLocations procedure implements the logic for searching for locations which are a certain number of kilometers away from a lodging's location. The procedure takes two input parameters, the lodging's name and the maximum distance from the lodging in kilometers that a location is allowed to have. When the procedure is called, first a check is made if the lodging's name is correct. Following the coordinates of the lodging and the city of the lodging are queried and stored. The next step is the calculation of the distance with the coordinates in the location table. For calculation, Pythagoras's theorem is used on an equirectangular projection. This formula can be used for smaller distances and is not as taxing on performance then a more accurate formula would be. Therefore, the search is limited to the boundaries of the city in which the lodging is located.[1] My own testing revealed a distance error of +/- 100 meters on distance smaller then 30km with the tendency to overshoot.

Test Case – Search within a 10 km radius

For the first test case a search within 10km of the test apartment is conducted.

```
SET @p0='Apartment in Vienna'; SET @p1='10'; CALL `Guest_SearchNearbyLocations`(@p0, @p1, @p2);
SELECT @p2 AS `message`;
```

With the following results:



Name	Street	Distance (km)
Hofburg	Hofburg	5.6
Schoenbrunn Palace	Schoenbrunner Schlossstrasse 47	2.8

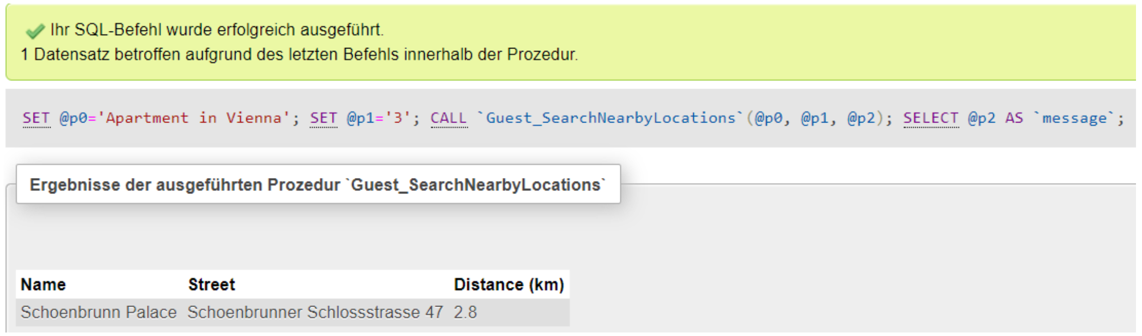
Description	Street	Distance (km)
Bus Stop	Harkortstrasse	9.4
Metro Station	Vorgartenstrasse	9.7
Metro Station	Taborstrasse	7.3
Metro Station	Nestroyplatz	8.0

Test Case – Search within a 3 km radius

To test the distance-based selection, the same search is performed, but only within 3 km radius.

```
SET @p0='Apartment in Vienna'; SET @p1='3'; CALL `Guest_SearchNearbyLocations`(@p0, @p1, @p2);
SELECT @p2 AS `message`;
```

The result confirms the correctness.



✓ Ihr SQL-Befehl wurde erfolgreich ausgeführt.
1 Datensatz betroffen aufgrund des letzten Befehls innerhalb der Prozedur.

```
SET @p0='Apartment in Vienna'; SET @p1='3'; CALL `Guest_SearchNearbyLocations`(@p0, @p1, @p2); SELECT @p2 AS `message`;
```

Ergebnisse der ausgeführten Prozedur `Guest_SearchNearbyLocations`

Name	Street	Distance (km)
Schoenbrunn Palace	Schoenbrunner Schlossstrasse 47	2.8

[1]https://www.movable-type.co.uk/scripts/latlong.html „Equirectangular approximation"

Booking and Transactions

Booking Table

After a user has chosen a lodging that fits all the criteria, the next step is to make a reservation. For handling the reservations, the booking table is used. The table holds information about the arrival and departure date, the guest user who booked the stay and the corresponding lodging.

SQL-Statement

```sql
CREATE TABLE booking(
    ID INT NOT NULL AUTO_INCREMENT,
    Arrival DATE NOT NULL,
    Departure DATE NOT NULL,
    UsersID INT,
    LodgingID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(UsersID) REFERENCES users(ID) ON DELETE SET NULL,
    FOREIGN KEY(LodgingID) REFERENCES lodging(ID) ON DELETE CASCADE
);
```

Transactions Table

Given that the booking table holds the administrative data, the transactions table holds all the financial related data. This includes the calculated total of the stay, the price per night, the currency of the lodging, whether the guest paid his stay or not and also whether the host claimed his money. Additionally, the exchange rates for the guest and lodging currencies at the time of the booking, the chosen payment option and the corresponding booking are stored.

SQL-Statement

```sql
CREATE TABLE transaction(
    ID INT NOT NULL AUTO_INCREMENT,
    Amount DECIMAL(19, 2) NOT NULL,
    Price DECIMAL(19, 2) NOT NULL,
    Currency VARCHAR(64) NOT NULL,
    Received BOOLEAN NOT NULL DEFAULT FALSE,
    Settled BOOLEAN NOT NULL DEFAULT FALSE,
    ExchangeEuroLodging DECIMAL(20, 9) NOT NULL,
    ExchangeEuroGuest DECIMAL(20, 9) NOT NULL,
    PaymentOptionID INT NOT NULL,
    BookingID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(PaymentOptionID) REFERENCES paymentoption(ID),
    FOREIGN KEY(BookingID) REFERENCES booking(ID) ON DELETE CASCADE
);
```

PaymentOption and Currency

The following two tables supplement the booking and transactions entities and form the financial management system of the data mart.

PaymentOption Table

As a supplementary table to the financial logic of the data mart, this entity contains a selection of payment methods and financial service providers.

SQL-Statement

```sql
CREATE TABLE paymentoption(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64) NOT NULL UNIQUE,
    PRIMARY KEY(ID)
);
```

Currency Table

The currency table holds the supported currencies of the data mart. Thereby the name of the currency and the current exchange rate to Euro, which is the systems standard currency, are stored. In a production environment the currency would be updated in certain intervals, for example by external business logic.

SQL-Statement

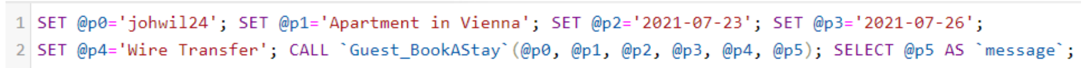
```sql
CREATE TABLE currency(
    ID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64) NOT NULL UNIQUE,
    ExchangeRate DECIMAL(20, 9) NOT NULL,
    PRIMARY KEY(ID)
);
```

BookAStay Procedure

The BookAStay procedure can be used to make a reservation in a lodging. The procedure takes the username of the guest, the lodging for which a reservation should be made, the arrival and departure date and the payment option. After calling the procedure, first various checks are performed. This includes if the lodging, the user and the payment option is issued correctly. In the next step there is a sanity check of the dates. If the arrival date is before the departure date and after the current day, a check looks for possible conflicts with other reservations. If all checks are passed, the booking is inserted into the booking table and an additional entry in the transactions table is made.

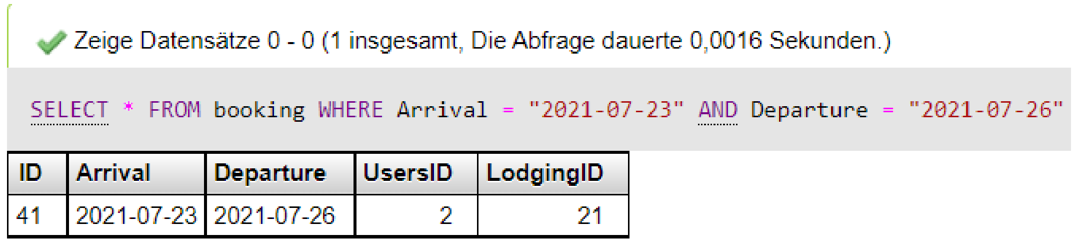
Test Case - Booking a stay in the test apartment

For the test case a reservation in the "Apartment in Vienna" is made.



```
SET @p0='johwil24'; SET @p1='Apartment in Vienna'; SET @p2='2021-07-23'; SET @p3='2021-07-26';
SET @p4='Wire Transfer'; CALL `Guest_BookAStay`(@p0, @p1, @p2, @p3, @p4, @p5); SELECT @p5 AS `message`;
```

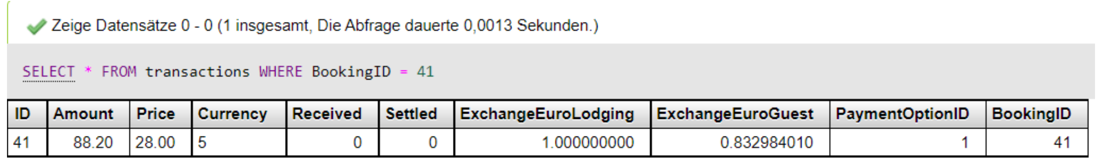
To test the correctness, a query on the booking and transactions table is used.



✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0,0016 Sekunden.)

```
SELECT * FROM booking WHERE Arrival = "2021-07-23" AND Departure = "2021-07-26"
```

ID	Arrival	Departure	UsersID	LodgingID
41	2021-07-23	2021-07-26	2	21



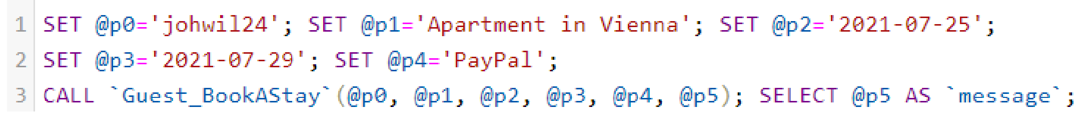
✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0,0013 Sekunden.)

```
SELECT * FROM transactions WHERE BookingID = 41
```

ID	Amount	Price	Currency	Received	Settled	ExchangeEuroLodging	ExchangeEuroGuest	PaymentOptionID	BookingID
41	88.20	28.00	5	0	0	1.000000000	0.832984010	1	41

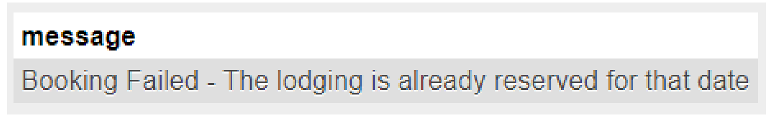
Test Case – Reservation conflict

To test the prevention of overlapping bookings, a second attempt for a reservation from 25.07.2021 to 29.07.2021 is made.



```
SET @p0='johwil24'; SET @p1='Apartment in Vienna'; SET @p2='2021-07-25';
SET @p3='2021-07-29'; SET @p4='PayPal';
CALL `Guest_BookAStay`(@p0, @p1, @p2, @p3, @p4, @p5); SELECT @p5 AS `message`;
```

Consequently, the booking attempt results in an error message.



message
Booking Failed - The lodging is already reserved for that date

Review and WriteAReview Procedure

After a stay, a user can write a review about the lodging. This functionality is implemented with the review entity and the corresponding procedure.

Review Table

The review table is used to store the content, rating, date and the corresponding booking of the review.

SQL-Statement

```sql
CREATE TABLE review(
    ID INT NOT NULL AUTO_INCREMENT,
    Content TEXT NOT NULL,
    Rating DECIMAL(2, 1) NOT NULL,
    BookingID INT NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(BookingID) REFERENCES booking(ID) ON DELETE CASCADE
);
```

WriteAReview Procedure

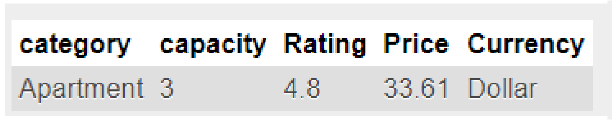
The WriteAReview procedure takes the username, lodging, departure date, the actual review content and the rating given by the user. After checking if all parameters are issued, if there is a booking for the guest at a certain departure date and that this is not a second review for the same stay, the review is inserted and the ratings for the lodging are recalculated.

Test Case – Writing a review for the test stay

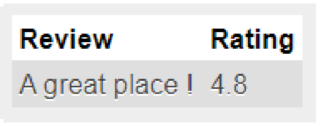
To test the procedure a review is issued for the newly created test booking.

```sql
SET @p0='johwil24'; SET @p1='Apartment in Vienna'; SET @p2='2021-07-26'; SET @p3='A great place !'; SET @p4='4.8';
CALL `Guest_WriteAReview`(@p0, @p1, @p2, @p3, @p4, @p5); SELECT @p5 AS `message`;
```

To verify the creation of the new review, the ShowLodgingDetails procedure is used with the following results:



category	capacity	Rating	Price	Currency
Apartment	3	4.8	33.61	Dollar



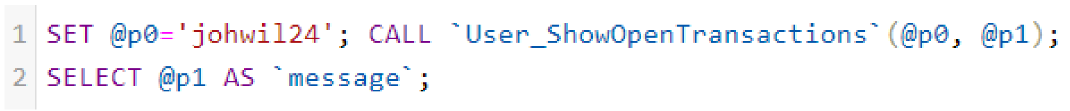
Review	Rating
A great place !	4.8

ShowOpenTransactions Procedure

The last part of this presentation is concerned with the display and management of open transactions. To display open transactions, the ShowOpenTransactions procedure can be used. The procedure takes a username as input and first of all checks if the username is correct. Following is a query that searches for all unreceived (Unpaid) transactions. If the user is a host, there is an additional query to search for unsettled (Not paid out) transactions. It is important to note that for hosts only unsettled transactions are displayed which have been already received and that the amount is changed to exclude the 5 percent platform fee.

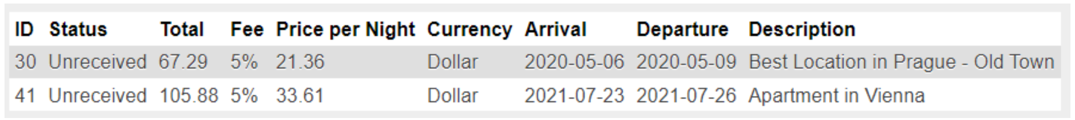
Test Case – Guest with unreceived transactions

The first test case is used to display all unreceived transactions of the guest "johwil24".



```
SET @p0='johwil24'; CALL `User_ShowOpenTransactions`(@p0, @p1);
SELECT @p1 AS `message`;
```

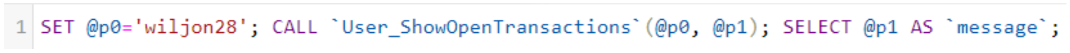
The result are two unreceived transactions, one of them being the test reservation.



ID	Status	Total	Fee	Price per Night	Currency	Arrival	Departure	Description
30	Unreceived	67.29	5%	21.36	Dollar	2020-05-06	2020-05-09	Best Location in Prague - Old Town
41	Unreceived	105.88	5%	33.61	Dollar	2021-07-23	2021-07-26	Apartment in Vienna

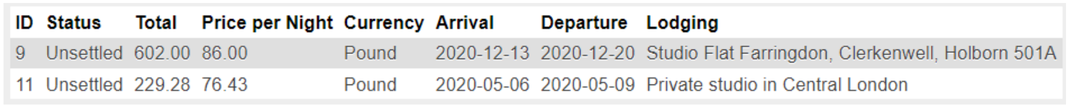
Test Case – Host with unsettled transactions

For the second test case, the open transactions of the user "wiljon28" are queried.



```
SET @p0='wiljon28'; CALL `User_ShowOpenTransactions`(@p0, @p1); SELECT @p1 AS `message`;
```

With the results:



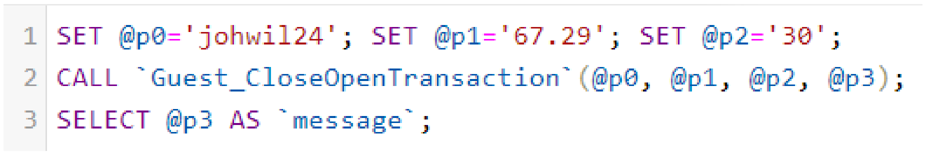
ID	Status	Total	Price per Night	Currency	Arrival	Departure	Lodging
9	Unsettled	602.00	86.00	Pound	2020-12-13	2020-12-20	Studio Flat Farringdon, Clerkenwell, Holborn 501A
11	Unsettled	229.28	76.43	Pound	2020-05-06	2020-05-09	Private studio in Central London

CloseOpenTransactions and SettleOpenTransactions Procedure

The last two procedures are used for changing the received and settled status of a transaction. The procedures can be called manually, although in a production environment the execution would probably be automated. For example, when choosing PayPal as payment option a dedicated API request could automatically set the flags to received. For the host users, the payment could be triggered automatically by other business logic, for example a financial software at certain intervals. This would also ensure that the transaction status in the data mart is in synch with the actual money in the bank account.

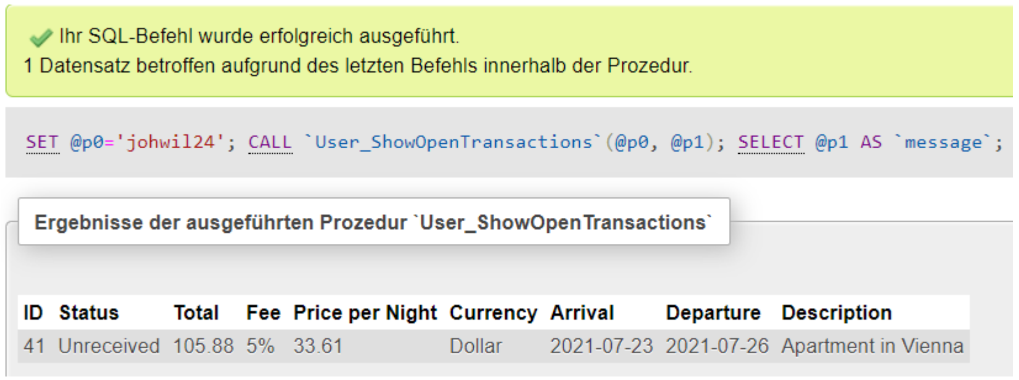
Test Case – Close the transactions of a guest user

For the first test case the first transactions of the user “johwil24” is set to received. For this, the username, the ID and the amount of the transaction must be given as parameters. The procedure checks all the parameters on correctness and then sets the received flag to 1.



```
SET @p0='johwil24'; SET @p1='67.29'; SET @p2='30';
CALL `Guest_CloseOpenTransaction`(@p0, @p1, @p2, @p3);
SELECT @p3 AS `message`;
```

Calling the ShowOpenTransactions procedure again, confirms the action.

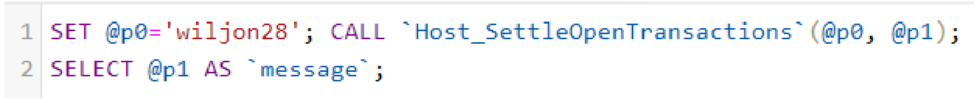


```
Ihr SQL-Befehl wurde erfolgreich ausgeführt.
1 Datensatz betroffen aufgrund des letzten Befehls innerhalb der Prozedur.

SET @p0='johwil24'; CALL `User_ShowOpenTransactions`(@p0, @p1); SELECT @p1 AS `message`;

Ergebnisse der ausgeführten Prozedur `User_ShowOpenTransactions`
```

ID	Status	Total	Fee	Price per Night	Currency	Arrival	Departure	Description
41	Unreceived	105.88	5%	33.61	Dollar	2021-07-23	2021-07-26	Apartment in Vienna

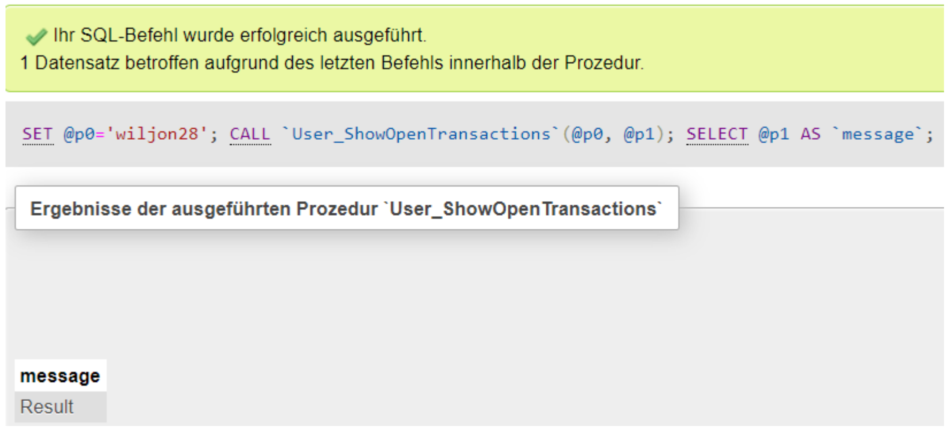
Test Case – Settle open transactions of a host

For the second test case, the open transactions of the user “wiljon28” are settled. The SettleOpenTransactions procedure only needs the username as input.



```
SET @p0='wiljon28'; CALL `Host_SettleOpenTransactions`(@p0, @p1);
SELECT @p1 AS `message`;
```

Also, here the ShowOpenTransactions procedure is called again.



```
Ihr SQL-Befehl wurde erfolgreich ausgeführt.
1 Datensatz betroffen aufgrund des letzten Befehls innerhalb der Prozedur.

SET @p0='wiljon28'; CALL `User_ShowOpenTransactions`(@p0, @p1); SELECT @p1 AS `message`;

Ergebnisse der ausgeführten Prozedur `User_ShowOpenTransactions`
```

message
Result