Phase 3 – Finalization

Documents

Abstract - Making of AirBnB Database

Project Overview

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

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

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

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

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

Functionality

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

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

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

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

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

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

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

Meta data

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

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

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

Learnings

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

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

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

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

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

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

Table Sizes:

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

*Data entries are irrelevant for size measurement.

Instruction Manual & Documentation

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

Structure:

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

Dependencies

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

Setup Environment

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

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

Instructions on the installation process follow:

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

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

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

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

Database Installation

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

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

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

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

Installation order:

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

4. test.sql (optional)

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

mysql -u your_username -p your_database_name < file.sql

Database Usage

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

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

Documentation

Database Overview

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

File Overview

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

Schema

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

Triggers

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

Data

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

Metadata

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

Test

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

List of test case views

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

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

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

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

Example for deleting all data relevant to a certain user:

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