# Table of Contents

# Quick Instructions

In this exercise, imagine that Rover has acquired a small pet care start-up. As an analyst, you have been tasked with the responsibility of exploring their database. We have shared a SQLite database with you containing this (fabricated) data. The file is named `db1.sqlite3`.

In order to query the database, you will need an appropriate client. We recommend either using the command line client ( `sqlite3` ), Python, R, or a GUI manager like SQLiteStudio. We've also included CSVs of the tables in this database in case you'd like to use another tool, like Excel. Once you are comfortable connecting to the database, please proceed to the exercises.

You may submit your report in any format you choose; for example, you might submit a Google Doc, a Jupyter notebook, RMarkdown, a PowerPoint presentation, etc. Assume your results will be shared with analysts and partners from marketing and operations. The latter will have a degree of analytical sophistication (i.e., they know some SQL and basic stats) but may need guidance in interpreting raw results or advanced techniques. Use this exercise as an opportunity to show off your communication skills and style.

Lastly, we understand that everyone's circumstances are different so there is no hard deadline; please complete the exercise with whatever timing works best for you. That said, we have one ask: we constantly collect data on this exercise so we can calibrate it and tune it going forward, so if you are willing to share how much total time you put into it, we would be appreciative.

**Submission Requirements**

- For sections II-VI, choose the top 3 sections that interest you and submit your work for those.
- Keep any information that could identify you (e.g., photos, your name) in a separate folder named PII or a separate file named pii.txt so that we can anonymize your work.

*For more detailed instructions, there is an appendix after the exercises. Also, if needed, there is an overview of the database in another appendix.*

# Exercises

## I. Exploring the Database

We begin by asking a few basic questions about the users of this platform. This first exercise is presented with answers so that you can diagnose any issues you might have connecting to or working with the data.

1. How many users have signed up?
   > The answer is `64393`.
2. How many users signed up prior to `2018-02-03`?
   > The answer is `35826`.
3. What percentage of users have added pets?
   > The answer is `80.43%`.
4. Of those users, how many pets have they added on average?
   > The answer is `1.501`.
5. What percentage of pets play well with cats?
   > The answer is `24.85%`.

## II. Conversations and Bookings

Some users can offer pet care services. When an owner needs pet care, they can create a conversation with another user that offers the service they are interested in. After exchanging some messages and possibly meeting in person, that conversation hopefully books. In that case, services are paid for and delivered. Occasionally, some conversations that have booked may be cancelled. Lastly, for uncancelled bookings, both owners and sitters have the option of leaving a review. In the following questions, we explore these concepts.

1. For uncancelled bookings, is the owner or provider more likely to leave a review and which tends to leave better reviews? How would you narrate this finding to a business partner?

## III. Recent Daily Booking Rate

The snapshot of this database was taken on `2018-08-02` at midnight and only contains data reflecting events prior to that date. A junior analyst is investigating daily booking rate during the days prior to the snapshot and is concerned about an apparent downward trend. You are tasked with helping them out.

1. First, let's reproduce their results. They tell you that *daily booking rate* is defined to be the percentage of conversations created each day that eventually book. What is the daily booking rate for each of the 90 days prior to the snapshot? Is there a downward trend?
2. Can you narrate a reason why this trend exists? Is there a reason to be concerned? Please provide additional data and evidence to justify your position.

## IV. Analyzing Take Rate

In order to do the next exercise, you will need to understand the fee structure for this company. Each user has a `fee` associated with their account (recorded on `people_person`). If that user books as an owner, the company charges a service fee (in addition to the booking total) that is a percentage of the booking total (to a maximum of $50). Also, each service has a `fee` amount (recorded on `services_service`). Before a provider receives their payment, the company takes a percentage of the booking total as dictated by that fee. As an example, suppose an owner has a fee amount of 5% and books with a service that has a fee amount of 15%. If the booking was for $100, then the owner would get charged $105 (adding the owner's fee). The $5 owner fee would go to the company. An additional $15 would also go to the company since the service had a 15% fee associated to it. The remaining $85 would go to the provider. To summarize:

| | Amount | Description |
|---|---:|---|
| Booking Total | $100 | e.g., 4 walks at $25/walk |
| Owner Fee | $5 | 5% of the booking total |
| Gross Billings | $105 | charged to the owner |
| Service Fee | $15 | 15% of the booking total |
| Net Revenue | $20 | all fees that go to the company |
| Provider Payment | $85 | earnings for the provider |

1. In each month, what were the gross billings and net revenue?
2. Define *take rate* to be the percentage of gross billings that is net revenue. In the previous example, the take rate is slightly more than 19% since $20/$105 is approximately 0.1905. In each month, what was the aggregate take rate?

3. Did take rate trend up or trend down or remain unchanged over time?
4. If it did change, investigate why and provide an explanation. Be sure to provide additional data/charts/evidence that justify your explaination. Any claims should be backed by data.

## V. New Conversation Flow

Internal documents indicate that this recently acquired company was performing many A/B tests; we would like to investigate one. This platform had a conversation page where owners and service providers could exchange messages as they organized their booking. The team thought this page could use a re-design and set out to improve its UI. A product manager then set up a test to measure the new page's effectiveness. On `2018-04-04` , an A/B test was launched. For those owners who sent a request, they would be randomly assigned to *variant* or *holdout* groups. Those users who are in the *variant* group would see the new conversation flow. However, those in the *holdout* group would see the old conversation flow. Providers would always see the old conversation flow.

1. Did conversations with the new conversation page book at a higher rate?
2. Is it statistically significant?
3. Do you have any reservations about the experiment design? What would you recommend as next steps?

## VI. Search Engine Marketing

Search engine advertising is a huge driver of new user accounts. Users that are aquired through search engine marketing can be identified by looking at `people_person.channel` . These users will have `'Google'` listed there. Historically, this company spent an average of $30 per account to advertise in the 2nd position on Google. However, on `2018-05-04` , they decided to start bidding for the 1st position. Since `2018-05-04` , they have spent `$207180` in total.

1. For each day, determine the count of users that joined and were acquired through Google. Plot this and confirm there is an inflection point on or near `2018-05-04` .
2. How many users were acquired via Google advertising since `2018-05-04` and what was the average cost per account?

3. Estimate how many users would have been acquired had the company not changed its bidding strategy. What would have been the marketing spend in that case?
4. How many additional accounts where created? What was the marginal cost per account for these additional accounts?

## Appendix: Detailed Instructions

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed database engine in the world. If you have experience with MySQL, PostgreSQL, Redshift, or other SQL variants, you should be able to write queries in SQLite. For more information on SQLite, see the following links:

- SQLite
- SQLite commands
- SQLite keywords

As mentioned before, you can query this database in a number of ways.

### Using `sqlite3`

Check to see if you already have `sqlite3` installed by issuing the following command in a terminal window:

```
$ sqlite3
```

If it is installed, you can use `.quit` to to exit to the command prompt. If not, install `sqlite3` using one of the above links. Once installed, you can connect to the database via `sqlite3` by navigating to the appropriate folder and running:

```
$ sqlite3 db1.sqlite3
```

Now, you can are connected to the database and are free to explore. Entering `.help` will give list interface commands (dot commands).

Querying the database can be done by typing your query directly into the client. Your query can be written over many lines by pressing `Enter` but will not execute until you include a semicolon ( `;` ) at the end of the query. It is also useful to turn on headers with `.headers on` and switch to column mode with `.mode columns` .

```
sqlite> .headers on
sqlite> .mode columns
sqlite> select
   ...>  date_joined
   ...>  , first_name
   ...>  , last_name
   ...>  , gender
   ...> from
   ...>  people_person
   ...> limit
   ...>  5
   ...> ;

date_joined                 first_name  last_name   gender
--------------------------  ----------  ----------  ----------
2015-07-05 10:46:39.392280  Selma       Panda       m
2015-07-05 13:28:10.586571  Ronny       Thatcher    f
2015-07-05 16:20:00.300618  Eduardo     Trossbach   f
2015-07-05 08:21:52.076197  Ressie      Zappone     m
2015-07-05 17:02:01.769151  Bernie      Anzaldua    f
```

For complicated queries, it is probably best to store your query in a separate file. If you have a query stored in a file called `query.sql` (which is in the same folder as the database file), execute the query by running `.read query.sql` .

To output the results as a CSV, switch the mode to `csv` and issue a `.once` command specifying the output file. Then, the next time you execute a SQL statement, the results will be written to the output file instead of being displayed in the terminal window.

```
sqlite> .headers on
sqlite> .mode csv
sqlite> .once output.csv
sqlite> .read query.sql
```

The result of the above is a file called `output.csv` which contains the output of the query stored in `query.sql` .

Certainly, there are other commands, options, and customizations that we could detail in these instructions. What we have presented here is the bare minimum to get you started. Feel free to read on your own and experiment. Also, do not hesitate to reach out if you are stuck or have clarifying questions.

## Using Python

If you are familiar with Python and `pandas` , it is very easy to connect to the database and import query results into dataframes. Ensure that `pandas` and `sqlite3` Python packages are installed in you environment. Then, consider the following script:

```
import sqlite3
import pandas as pd

conn = sqlite3.connect("db1.sqlite3")
query = '''
select
  date_joined, first_name, last_name, gender
from
  people_person
limit 5;
'''

df = pd.read_sql_query(query, conn)
```

Following execution, `df` is a `pandas` dataframe that contains the results of `query` . `df` can be manipulated as necessary.

If you prefer to do data analysis in Pyhton, you can simply load the CSVs usings `pandas` :

```
df = pd.read_csv('/path/to/CSV')
```

Feel free to use others tools like `jupyter` notebooks and `matplotlib` to explore the data and present your findings.

## Using R

As with Python, you are invited to use R if you prefer.

Querying the database:

```r
install.packages('DBI')
install.packages('RSQLite')
library('DBI')
db <- dbConnect(RSQLite::SQLite(), "db1.sqlite3")
query <- 'select ... from ... limit 5;'
df <- dbGetQuery(db, query)
```

Loading CSVs:

```r
df <- read.csv('/path/to/CSV')
```

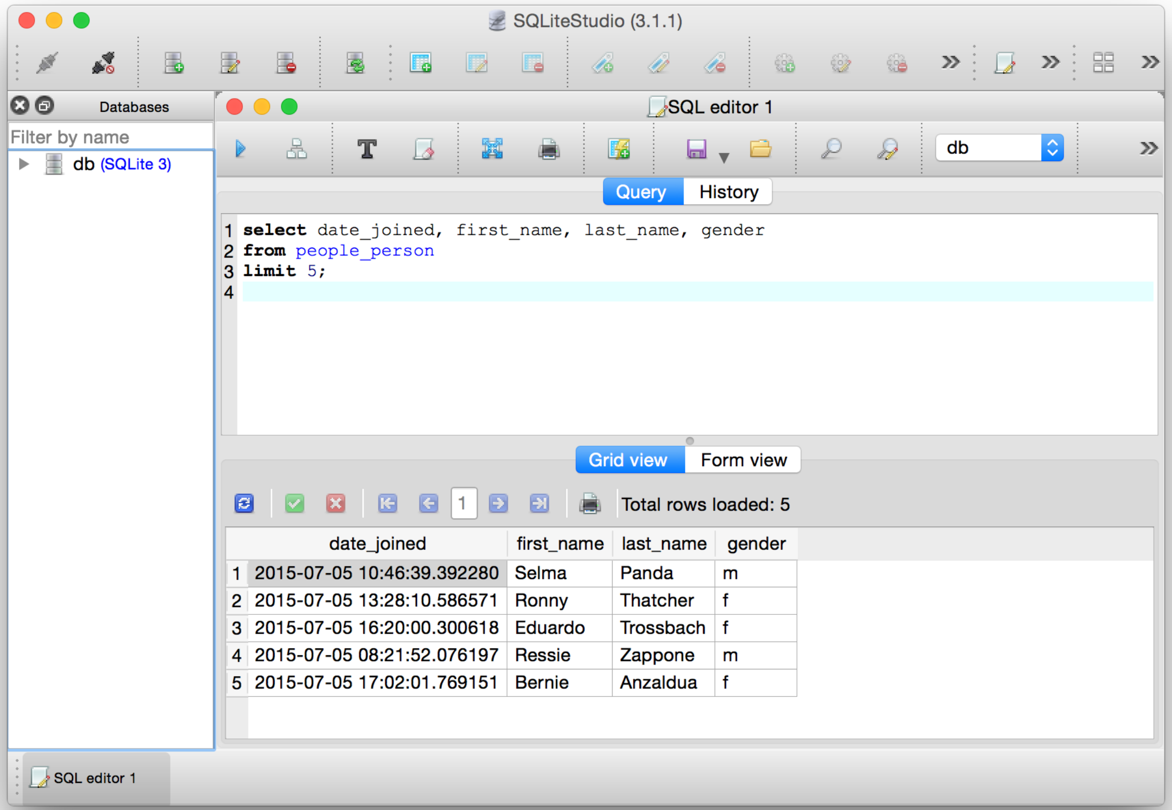
## Using SQLiteStudio

A GUI interface may be easier for some applicants and you are welcome to use one. We suggest SQLiteStudio. As with the previous section, we will provide a minimal explanation on how to get started.

After installing SQLiteStudio, you will need to connect to the database. Click on the *Add a Database* button.

After navigating and choosing the database we provided, choose *OK*. To query the database, click on the *Open SQL Editor* button.

Simply type your query and click on the *Execute Query* button.



```
select date_joined, first_name, last_name, gender
from people_person
limit 5;
```

Total rows loaded: 5

| | date_joined | first_name | last_name | gender |
|---|---|---|---|---|
| 1 | 2015-07-05 10:46:39.392280 | Selma | Panda | m |
| 2 | 2015-07-05 13:28:10.586571 | Ronny | Thatcher | f |
| 3 | 2015-07-05 16:20:00.300618 | Eduardo | Trossbach | f |
| 4 | 2015-07-05 08:21:52.076197 | Ressie | Zappone | m |
| 5 | 2015-07-05 17:02:01.769151 | Bernie | Anzaldua | f |

Lastly, you can export the results of a query by clicking on the *Export* button and following the wizard.

# Appendix: Database Overview

## `people_person`

This table details each user on our site. This table may contain dog owners, dog sitters, or people who have not transacted on our site. Many of the fields on this table are self explanatory but we have detailed a few below.

- `channel` - This field reports how this user discovered our site when they signed up.
- `date_joined` - The timestamp for when this user signed up.
- `fee` - When a user books a service as a dog owner, we charge the owner a separate service fee that takes the form of a percentage of the booking total.

## `people_testsegmentation`

Occasionally, this company would run an A/B test which required that users get placed in two groups. This table provides a log for experiments which require user-level segmentations. Many of the fields on this table are self explanatory but we have detailed a few below.

- `person_id` - This foreign key reports the `people_person` record that was segmented.
- `test_name` - Multiple tests were run on this site and all are logged on this table. Use this column to filter to the correct experiment.
- `test_group` - For the purposes of the experiment in `test_name`, the user given by `person_id` was segmented into the group named in this column (e.g., `holdout`, `variant`, `A`, `B`, etc.).
- `added` - The timestamp reporting the time when this user was segmented.

## `pets_pet`

This table details each pet that a user has added to their profile. One owner may have more than one pet, but not vice versa. Many of the fields on this table are self explanatory but we have detailed a few below.

- `description` - A short (lorem ipsum) description of the pet.
- `plays_cats` - If 1, then this pet plays well with cats.
- `plays_children` - If 1, then this pet plays well with children.
- `plays_dogs` - If 1, then this pet plays well with dogs.
- `spayed_neutered` - If 1, then this pet has been spayed or neutered.
- `house_trained` - If 1, then this pet is house trained.
- `owner_id` - This foreign key reports the `people_person` record for this pet's owner.

## `services_service`

On our site, users may offer pet care services. This table stores a record for each service that is offered. Each user can offer more than one service, but not more than one of each type. Many of the fields on this table are self explanatory but we have detailed a few below.

- `max_dogs` - This number is the maximum number of pets this provider would prefer to care for.
- `fee` - When a user books with a service, we take a percentage of the booking total. This field reports the percentage.
- `provider_id` - This foreign key reports the `people_person` record for this service's provider.
- `added` - A timestamp for when this service became active.
- `price` - The price per unit booked.

## `conversations_conversation`

An owner can book a service provider by starting a conversation with them. This table stores a record for each conversation started on our platform. Many of the fields on this table are self explanatory but we have detailed a few below.

- `start_date` - This is the date for which pet care will first be needed.
- `end_date` - This is the last date for which pet care will be needed.
- `units` - This is the number of units of service that the owner is interested in booking.
- `added` - A timestamp for when this conversation was created.
- `booking_total` - This is the dollar amount (not including the owner's service fee) that this booking would cost.
- `requester_id` - This foreign key reports the `people_person` record for the pet owner that is requesting pet care.
- `service_id` - This foreign key reports the `services_service` record for the service that the pet owner is requesting.
- `booked_at` - If the request is booked, this timestamp reports when that occurred.
- `cancelled_at` - A booked request can be cancelled. In that case, this timestamp reports when that occurred.

## `conversations_conversation_pets`

Since a booking may involve many pets and many pets might have had many bookings, it is necessary to store this many-to-many relationship on a separate table. Many of the fields on this table are self explanatory but we have detailed a few below.

- `conversation_id` - A foreign key to a booking request on the `conversations_converation` table. If this conversation involves caring for more than one pets, then this `conversation_id` will occur on more than one row on this table (once for each pet).
- `pet_id` - A foreign key to a pet that will receive pet care during the corresponding conversation's booking.

## `conversations_message`

Each conversation consists of a series of messages. A conversation may contain many messages, but not vice versa. Many of the fields on this table are self explanatory but we have detailed a few below.

- `conversation_id` - This foreign key reports the conversation in `conversations_conversation` for which this message is apart of.
- `sender_id` - This foreign key reports the user in `people_person` that sent this message.

## `conversations_review`

If a booking occurs, then either participant can leave a review for the experience. This table records those reviews, which consist of a brief statement and a star rating. Many of the fields on this table are self explanatory but we have detailed a few below.

- `conversation_id` - This foreign key reports the booking in `conversations_conversation` for which this review pertains.
- `reviewer_id` - This foreign key reports the user in `people_person` that wrote this review.