- Optimize full table scans.
- Distinguish between long queries and short queries.
- Choose the right optimization technique for each query type.
- Avoid the pitfalls of ORM frameworks.

At the end of the book, we present the *Ultimate Optimization Algorithm*, which guides a database developer through the process of producing the most performant query.

The Postgres Air Database

Throughout this book, examples are built on one of the databases of a virtual airline company called Postgres Air. This company connects over 600 virtual destinations worldwide, offers about 32,000 direct virtual flights weekly, and has over 100,000 virtual members in its frequent flyer program and many more passengers every week. The company fleet consists of virtual aircraft. As operations are entirely virtual, the company is not affected by the COVID-19 pandemic.

Please note that all data provided in this database is fictional and provided for illustrative purposes only. Although some data appears very realistic (especially descriptions of airports and aircraft), they cannot be used as sources of information about real airports or aircraft. All phone numbers, email addresses, and names are generated.

To install the training database on your local system, please access the shared directory postgres_air_dump using this link: `https://drive.google.com/drive/folders/13F7M80Kf_somnjb-mTYAnh1hW1Y_g4kJ?usp=sharing`

You can also use a QR code as shown in Figure 1.

Figure 1. *QR code to access the database dump*

This shared directory contains data dump of the postgres_air schema in three formats: directory format, default pg_dump format, and compressed SQL format.

The total size of each is about 1.2 GB. Use directory format if you prefer to download smaller files (the max file size is 419 MB). Use SQL format if you want to avoid warnings about object ownership.

For directory format and default format, use pg_restore (*www.postgresql.org/docs/12/app-pgrestore.html*). For SQL format, unzip the file and use psql for restore.

In addition, after you restore the data, you will need to run the script in Listing 1 to create several indexes.

Listing 1. Initial set of indexes

```
SET search_path TO postgres_air;
CREATE INDEX flight_departure_airport ON
flight(departure_airport);
CREATE INDEX flight_scheduled_departure ON postgres_air.flight
(scheduled_departure);
CREATE INDEX flight_update_ts ON postgres_air.flight  (update_ts);
CREATE INDEX booking_leg_booking_id ON postgres_air.booking_leg
(booking_id);
CREATE INDEX booking_leg_update_ts ON postgres_air.booking_leg
(update_ts);
CREATE INDEX account_last_name
  ON account (last_name);
```

We will use this database schema to illustrate the concepts and methods that are covered in this book. You can also use this schema to practice optimization techniques.

This schema contains data that might be stored in an airline booking system. We assume that you have booked a flight online, at least once, so the data structure should be easily understood. Of course, the structure of this database is much simpler than the structure of any real database of this kind.

Anyone who books a flight needs to create an account, which stores login information, first and last names, and contact information. We also store data about frequent flyers, which might or might not be attached to an account. A person who makes a booking can book for several passengers, who might or might not have their accounts in the system. Each booking may include several flights (legs). Before the flight, each traveler is issued a boarding pass with a seat number.

The Entity-Relationship (ER) diagram for this database is presented in Figure 2.

- *airport* stores information about airports and contains the airport's three-character (IATA) code, name, city, geographical location, and time zone.
- *flight* stores information about flights between airports. For each flight, the table stores a flight number, arrival and departure airports, scheduled and actual arrival and departure times, aircraft code, and flight status.
- *account* stores login credentials, the account holder's first and last names, and possibly a reference to a frequent flyer program membership; each account may potentially have multiple phone numbers, which are stored in the *phone* table.
- *frequent_flyer* stores information about membership in the frequent flyer program.
- *booking* contains information about booked trips; each trip may have several booking legs and several passengers.
- *booking_leg* stores individual legs of bookings.
- *passenger* stores information about passengers, linked to each booking. Note that a passenger ID is unique to a single booking; for any other booking, the same person will have a different passenger ID.

- *aircraft* provides the aircraft's description, and the *seat* table stores seat maps for each of aircraft types.
- Finally, the *boarding_pass* table stores information about issued boarding passes.

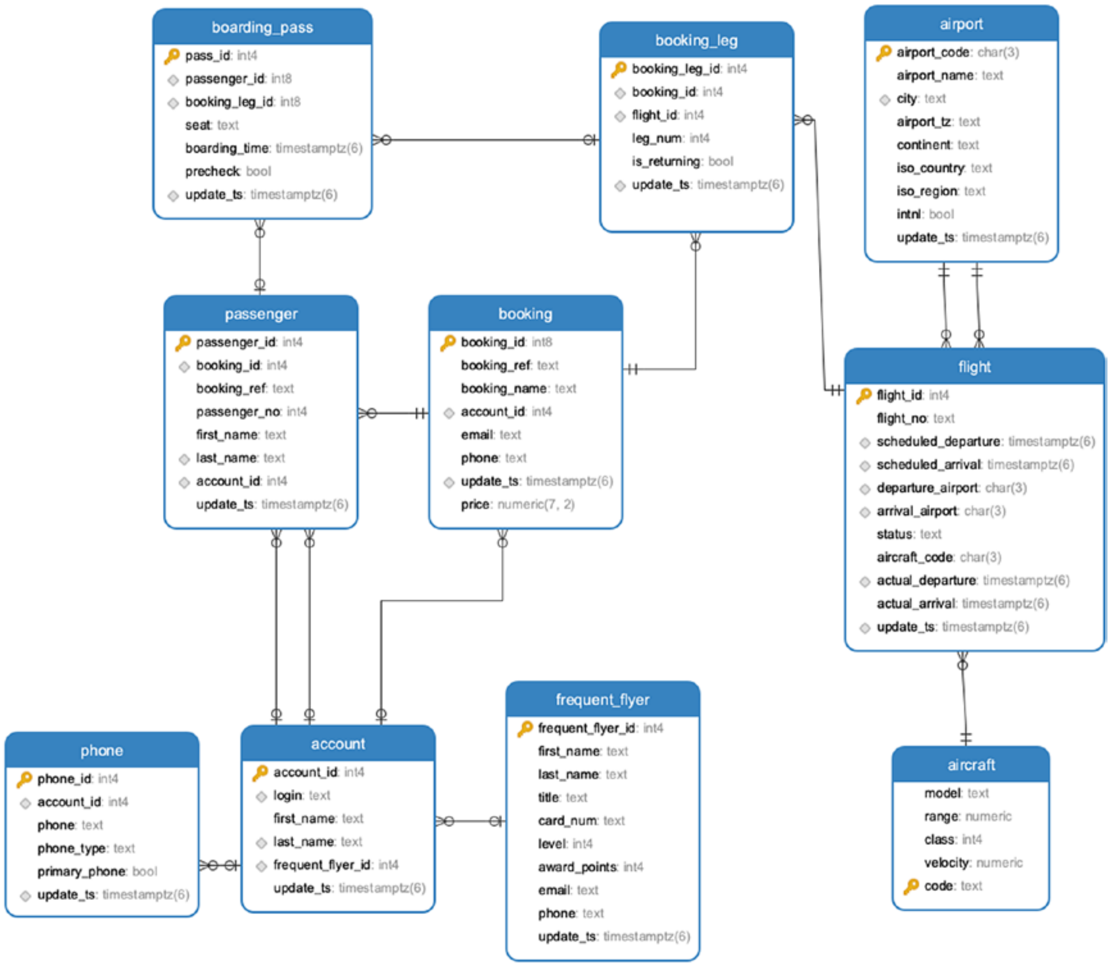

Figure 2. *ER diagram of the booking schema*