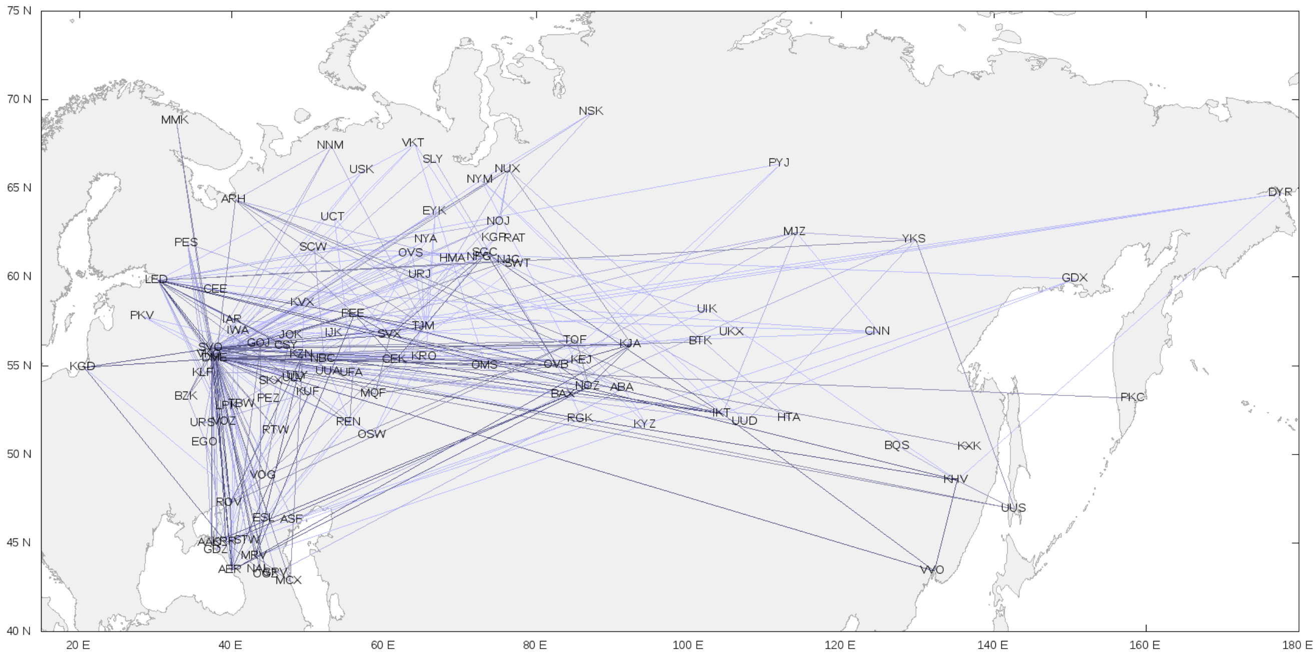
Appendix L. Demo Database “Airlines”

This is an overview of a demo database for PostgreSQL. This appendix describes the database schema, which consists of eight tables and several views. The subject field of this database is airline flights in Russia. You can download the database from *our website*. See Section L.1 for details.

Figure L.1. Airlines in Russia



You can use this database for various purposes, such as:

- learning SQL language on your own
- preparing books, manuals, and courses on SQL
- showing Postgres Pro features in stories and articles

When developing this demo database, we pursued several goals:

- Database schema must be simple enough to be understood without extra explanations.
- At the same time, database schema must be complex enough to allow writing meaningful queries.
- The database must contain true-to-life data that will be interesting to work with.

This demo database is distributed under the *PostgreSQL license*.

You can send us your feedback to *edu@postgrespro.ru*.

L.1. Installation

The demo database is available at *edu.postgrespro.ru* in three flavors, which differ only in the data size:

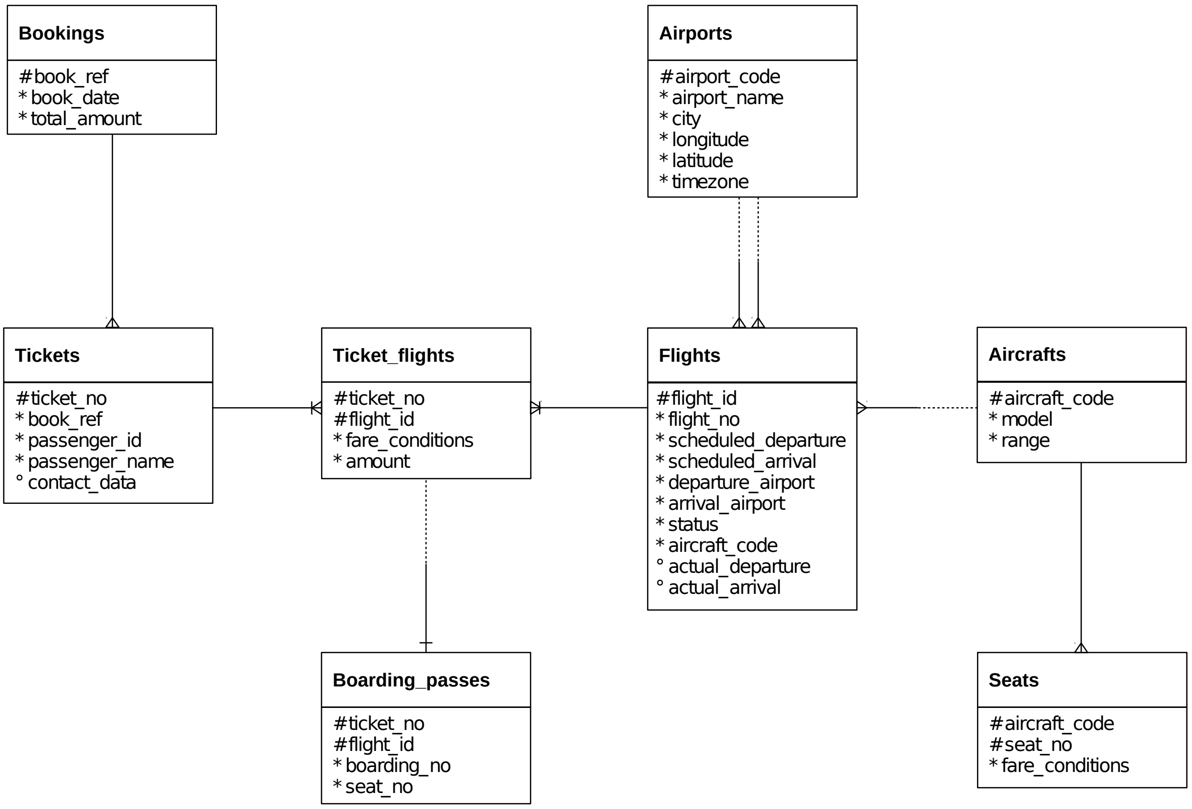
- *demo_small.zip* (21 MB) — flight data for one month (DB size is 265 MB)
- *demo_medium.zip* (62 MB) — flight data for three months (DB size is 666 MB)
- *demo_big.zip* (232 MB) — flight data for one year (DB size is 2502 MB)

The small database is good for writing queries, and it will not take up much disk space. The large database can help you understand the query behavior on large data volumes and consider query optimization.

The files include an SQL script that creates the `demo` database and fills it with data (virtually, it is a backup copy created with the pg_dump utility). Note that if the `demo` database already exists, it will be deleted and recreated! The owner of the `demo` database will be the DBMS user who run the script.

L.2. Schema Diagram

Figure L.2. Bookings Schema Diagram



L.3. Schema Description

The main entity is a booking (`bookings`).

One booking can include several passengers, with a separate ticket (`tickets`) issued to each passenger. A ticket has a unique number and includes information about the passenger. As such, the passenger is not a separate entity. Both the passenger's name and identity document number can change over time, so it is impossible to uniquely identify all the tickets of a particular person; for simplicity, we can assume that all passengers are unique.

The ticket includes one or more flight segments (`ticket_flights`). Several flight segments can be included into a single ticket if there are no non-stop flights between the points of departure and destination (connecting flights), or if it is a round-trip ticket. Although there is no constraint in the schema, it is assumed that all tickets in the booking have the same flight segments.

Each flight (`flights`) goes from one airport (`airports`) to another. Flights with the same flight number have the same points of departure and destination, but differ in departure date.

At flight check-in, the passenger is issued a boarding pass (`boarding_passes`), where the seat number is specified. The passenger can check in for the flight only if this flight is included into the ticket. The flight-seat combination must be unique to avoid issuing two boarding passes for the same seat.

The number of seats (`seats`) in the aircraft and their distribution between different travel classes depends on the model of the aircraft (`aircrafts`) performing the flight. It is assumed that every aircraft model has only one cabin configuration. Database schema does not check that seat numbers in boarding passes have the corresponding seats in the aircraft (such verification can be done using table triggers, or at the application level).

L.4. Schema Objects

L.4.1. List of Relations

```
       Name      |     Type      |  Small | Medium |   Big  |     Description
-----------------+---------------+--------+--------+--------+-------------------
 aircrafts       | table         |  16 kB |  16 kB |  16 kB | Aircraft
 airports        | table         |  48 kB |  48 kB |  48 kB | Airports
 boarding_passes | table         |  31 MB | 102 MB | 427 MB | Boarding passes
 bookings        | table         |  13 MB |  30 MB | 105 MB | Bookings
 flights         | table         |   3 MB |   6 MB |  19 MB | Flights
 flights_v       | view          |   0 kb |   0 kB |   0 kB | Flights
 routes          | mat. view     | 136 kB | 136 kB | 136 kB | Routes
 seats           | table         |  88 kB |  88 kB |  88 kB | Seats
 ticket_flights  | table         |  64 MB | 145 MB | 516 MB | Flight segments
 tickets         | table         |  47 MB | 107 MB | 381 MB | Tickets
```

L.4.2. Table `bookings.aircrafts`

Each aircraft model is identified by its three-digit code (`aircraft_code`). The table also includes the name of the aircraft model (`model`) and the maximal flying distance, in kilometers (`range`).

```
    Column     |  Type   | Modifiers    |             Description
---------------+---------+--------------+-----------------------------------
 aircraft_code | char(3) | NOT NULL     | Aircraft code, IATA
 model         | text    | NOT NULL     | Aircraft model
 range         | integer | NOT NULL     | Maximal flying distance, km
Indexes:
    PRIMARY KEY, btree (aircraft_code)
Check constraints:
    CHECK (range > 0)
Referenced by:
    TABLE "flights" FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts(aircraft_code)
    TABLE "seats" FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts(aircraft_code) ON DELETE CASCADE
```

L.4.3. Table `bookings.airports`

An airport is identified by a three-letter code (`airport_code`) and has a name (`airport_name`).

There is no separate entity for the city, but there is a city name (`city`) to identify the airports of the same city. The table also includes longitude (`longitude`), latitude (`latitude`), and the time zone (`timezone`).

```
   Column     |  Type   | Modifiers    |           Description
--------------+---------+--------------+-------------------------------
 airport_code | char(3) | NOT NULL     | Airport code
 airport_name | text    | NOT NULL     | Airport name
 city         | text    | NOT NULL     | City
 longitude    | float   | NOT NULL     | Airport coordinates: longitude
 latitude     | float   | NOT NULL     | Airport coordinates: latitude
 timezone     | text    | NOT NULL     | Airport time zone
```

```
Indexes:
    PRIMARY KEY, btree (airport_code)
Referenced by:
    TABLE "flights" FOREIGN KEY (arrival_airport)
        REFERENCES airports(airport_code)
    TABLE "flights" FOREIGN KEY (departure_airport)
        REFERENCES airports(airport_code)
```

L.4.4. Table `bookings.boarding_passes`

At the time of check-in, which opens twenty-four hours before the scheduled departure, the passenger is issued a boarding pass. Like the flight segment, the boarding pass is identified by the ticket number and the flight number.

Boarding passes are assigned sequential numbers (`boarding_no`), in the order of check-ins for the flight (this number is unique only within the context of a particular flight). The boarding pass specifies the seat number (`seat_no`).

```
   Column    |     Type    | Modifiers    |         Description
-------------+-------------+--------------+--------------------------
 ticket_no   | char(13)    | NOT NULL     | Ticket number
 flight_id   | integer     | NOT NULL     | Flight ID
 boarding_no | integer     | NOT NULL     | Boarding pass number
 seat_no     | varchar(4)  | NOT NULL     | Seat number
Indexes:
    PRIMARY KEY, btree (ticket_no, flight_id)
    UNIQUE CONSTRAINT, btree (flight_id, boarding_no)
    UNIQUE CONSTRAINT, btree (flight_id, seat_no)
Foreign-key constraints:
    FOREIGN KEY (ticket_no, flight_id)
        REFERENCES ticket_flights(ticket_no, flight_id)
```

L.4.5. Table `bookings.bookings`

Passengers book tickets for themselves, and, possibly, for several other passengers, in advance (`book_date`, not earlier than one month before the flight). The booking is identified by its number (`book_ref`, a six-position combination of letters and digits).

The `total_amount` field stores the total cost of all tickets included into the booking, for all passengers.

```
   Column     |      Type     | Modifiers    |         Description
--------------+---------------+--------------+---------------------------
 book_ref     | char(6)       | NOT NULL     | Booking number
 book_date    | timestamptz   | NOT NULL     | Booking date
 total_amount | numeric(10,2) | NOT NULL     | Total booking cost
Indexes:
    PRIMARY KEY, btree (book_ref)
Referenced by:
    TABLE "tickets" FOREIGN KEY (book_ref) REFERENCES bookings(book_ref)
```

L.4.6. Table `bookings.flights`

The natural key of the `bookings.flights` table consists of two fields — `flight_no` and `scheduled_departure`. To make foreign keys for this table more compact, a surrogate key is used as the primary key (`flight_id`).

A flight always connects two points — the airport of departure (`departure_airport`) and arrival (`arrival_airport`). There is no such entity as a “connecting flight”: if there are no non-stop flights from one airport to another, the ticket simply includes several required flight segments.

Each flight has a scheduled date and time of departure (`scheduled_departure`) and arrival (`scheduled_arrival`). The actual departure time (`actual_departure`) and arrival time (`actual_arrival`) can differ: the difference is usually not very big, but sometimes can be up to several hours if the flight is delayed.

Flight status (`status`) can take one of the following values:

`Scheduled`

The flight is available for booking. It happens one month before the planned departure date; before that time, there is no entry for this flight in the database.

`On Time`

The flight is open for check-in (in twenty-four hours before the scheduled departure) and is not delayed.

`Delayed`

The flight is open for check-in (in twenty-four hours before the scheduled departure) but is delayed.

`Departed`

The aircraft has already departed and is airborne.

`Arrived`

The aircraft has reached the point of destination.

`Cancelled`

The flight is canceled.

```
        Column       |     Type    | Modifiers    |         Description
---------------------+-------------+--------------+-----------------------------
 flight_id           | serial      | NOT NULL     | Flight ID
 flight_no           | char(6)     | NOT NULL     | Flight number
 scheduled_departure | timestamptz | NOT NULL     | Scheduled departure time
 scheduled_arrival   | timestamptz | NOT NULL     | Scheduled arrival time
 departure_airport   | char(3)     | NOT NULL     | Airport of departure
 arrival_airport     | char(3)     | NOT NULL     | Airport of arrival
 status              | varchar(20) | NOT NULL     | Flight status
 aircraft_code       | char(3)     | NOT NULL     | Aircraft code, IATA
 actual_departure    | timestamptz |              | Actual departure time
 actual_arrival      | timestamptz |              | Actual arrival time
Indexes:
    PRIMARY KEY, btree (flight_id)
    UNIQUE CONSTRAINT, btree (flight_no, scheduled_departure)
Check constraints:
    CHECK (scheduled_arrival > scheduled_departure)
    CHECK ((actual_arrival IS NULL)
       OR  ((actual_departure IS NOT NULL AND actual_arrival IS NOT NULL)
            AND (actual_arrival > actual_departure)))
    CHECK (status IN ('On Time', 'Delayed', 'Departed',
                      'Arrived', 'Scheduled', 'Cancelled'))
Foreign-key constraints:
    FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts(aircraft_code)
    FOREIGN KEY (arrival_airport)
        REFERENCES airports(airport_code)
    FOREIGN KEY (departure_airport)
        REFERENCES airports(airport_code)
Referenced by:
    TABLE "ticket_flights" FOREIGN KEY (flight_id)
        REFERENCES flights(flight_id)
```

L.4.7. Table bookings.seats

Seats define the cabin configuration of each aircraft model. Each seat is defined by its number (`seat_no`) and has an assigned travel class (`fare_conditions`): `Economy`, `Comfort` or `Business`.

```
      Column     |     Type    | Modifiers    |      Description
-----------------+-------------+--------------+--------------------
 aircraft_code   | char(3)     | NOT NULL     | Aircraft code, IATA
 seat_no         | varchar(4)  | NOT NULL     | Seat number
 fare_conditions | varchar(10) | NOT NULL     | Travel class
Indexes:
    PRIMARY KEY, btree (aircraft_code, seat_no)
Check constraints:
    CHECK (fare_conditions IN ('Economy', 'Comfort', 'Business'))
Foreign-key constraints:
    FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts(aircraft_code) ON DELETE CASCADE
```

L.4.8. Table bookings.ticket_flights

A flight segment connects a ticket with a flight and is identified by their numbers.

Each flight has its cost (`amount`) and travel class (`fare_conditions`).

```
     Column      |      Type      | Modifiers    |    Description
-----------------+----------------+--------------+---------------------
 ticket_no       | char(13)       | NOT NULL     | Ticket number
 flight_id       | integer        | NOT NULL     | Flight ID
 fare_conditions | varchar(10)    | NOT NULL     | Travel class
 amount          | numeric(10,2)  | NOT NULL     | Travel cost
Indexes:
    PRIMARY KEY, btree (ticket_no, flight_id)
Check constraints:
    CHECK (amount >= 0)
    CHECK (fare_conditions IN ('Economy', 'Comfort', 'Business'))
Foreign-key constraints:
    FOREIGN KEY (flight_id) REFERENCES flights(flight_id)
    FOREIGN KEY (ticket_no) REFERENCES tickets(ticket_no)
Referenced by:
    TABLE "boarding_passes" FOREIGN KEY (ticket_no, flight_id)
        REFERENCES ticket_flights(ticket_no, flight_id)
```

L.4.9. Table bookings.tickets

A ticket has a unique number (`ticket_no`) that consists of 13 digits.

The ticket includes a passenger ID (`passenger_id`) — the identity document number, — their first and last names (`passenger_name`), and contact information (`contact_data`).

Neither the passenger ID, nor the name is permanent (for example, one can change the last name or passport), so it is impossible to uniquely identify all tickets of a particular passenger.

```
     Column     |     Type    | Modifiers    |          Description
----------------+-------------+--------------+-----------------------------
 ticket_no      | char(13)    | NOT NULL     | Ticket number
 book_ref       | char(6)     | NOT NULL     | Booking number
 passenger_id   | varchar(20) | NOT NULL     | Passenger ID
 passenger_name | text        | NOT NULL     | Passenger name
 contact_data   | jsonb       |              | Passenger contact information
Indexes:
    PRIMARY KEY, btree (ticket_no)
Foreign-key constraints:
    FOREIGN KEY (book_ref) REFERENCES bookings(book_ref)
Referenced by:
    TABLE "ticket_flights" FOREIGN KEY (ticket_no) REFERENCES tickets(ticket_no)
```

L.4.10. View `bookings.flights_v`

There is a `flights_v` view over the `flights` table that provides additional information:

- Details about the airport of departure — `departure_airport`, `departure_airport_name`, `departure_city`
- Details about the airport of arrival — `arrival_airport`, `arrival_airport_name`, `arrival_city`
- Local departure time — `scheduled_departure_local`, `actual_departure_local`
- Local arrival time — `scheduled_arrival_local`, `actual_arrival_local`
- Flight duration — `scheduled_duration`, `actual_duration`.

```
          Column           |    Type     |              Description
---------------------------+-------------+--------------------------------------
 flight_id                 | integer     | Flight ID
 flight_no                 | char(6)     | Flight number
 scheduled_departure       | timestamptz | Scheduled departure time
 scheduled_departure_local | timestamp   | Scheduled departure time,
                           |             | local time at the point of departure
 scheduled_arrival         | timestamptz | Scheduled arrival time
 scheduled_arrival_local   | timestamp   | Scheduled arrival time,
                           |             | local time at the point of destination
 scheduled_duration        | interval    | Scheduled flight duration
 departure_airport         | char(3)     | Departure airport code
 departure_airport_name    | text        | Departure airport name
 departure_city            | text        | City of departure
 arrival_airport           | char(3)     | Arrival airport code
 arrival_airport_name      | text        | Arrival airport name
 arrival_city              | text        | City of arrival
 status                    | varchar(20) | Flight status
 aircraft_code             | char(3)     | Aircraft code, IATA
 actual_departure          | timestamptz | Actual departure time
 actual_departure_local    | timestamp   | Actual departure time,
                           |             | local time at the point of departure
 actual_arrival            | timestamptz | Actual arrival time
 actual_arrival_local      | timestamp   | Actual arrival time,
                           |             | local time at the point of destination
 actual_duration           | interval    | Actual flight duration
```

L.4.11. Materialized View `bookings.routes`

The `bookings.flights` table contains some redundancies, which you can use to single out route information (flight number, airports of departure and destination) that does not depend on the exact flight dates.

Such information constitutes the `routes` materialized view.

```
        Column          |   Type    |              Description
------------------------+-----------+-------------------------------------
```

```
 flight_no               | char(6)   | Flight number
 departure_airport       | char(3)   | Departure airport code
 departure_airport_name  | text      | Departure airport name
 departure_city          | text      | City of departure
 arrival_airport         | char(3)   | Arrival airport code
 arrival_airport_name    | text      | Arrival airport name
 arrival_city            | text      | City of arrival
 aircraft_code           | char(3)   | Aircraft code, IATA
 duration                | interval  | Flight duration
 days_of_week            | integer[] | Days of the week on which flights are performed
```

L.4.12. Function `now`

The demo database contains “snapshots” of data — similar to a backup copy of a real system captured at some point in time. For example, if a flight has the `Departed` status, it means that the aircraft had already departed and was airborne at the time of the backup copy.

The “snapshot” time is saved in the `bookings.now()` function. You can use this function in demo queries for cases where you would use the `now()` function in a real database.

In addition, the return value of this function determines the version of the demo database. The latest version available is of October 13, 2016.

L.5. Usage

L.5.1. Schema `bookings`

The `bookings` schema contains all objects of the demo database. It means that when you access database objects, you either have to explicitly specify the schema name (for example: `bookings.flights`), or modify the `search_path` configuration parameter beforehand (for example: `SET search_path = bookings, public;`).

However, for the `bookings.now` function, you always have to specify the schema to distinguish this function from the standard `now` function.

L.5.2. Sample Queries

To better understand the contents of the demo database, let's take a look at the results of several simple queries.

The results displayed below were received on a small database version (demo_small) of October 13, 2016. If the same queries return different data on your system, check your demo database version (using the `bookings.now` function). Some minor deviations may be caused by the difference between your local time and Moscow time, or your locale settings.

All flights are operated by several types of aircraft:

```
SELECT * FROM aircrafts;
```

```
 aircraft_code |        model        | range
---------------+---------------------+-------
 773           | Boeing 777-300      | 11100
 763           | Boeing 767-300      |  7900
 SU9           | Sukhoi SuperJet-100 |  3000
 320           | Airbus A320-200     |  5700
 321           | Airbus A321-200     |  5600
 319           | Airbus A319-100     |  6700
 733           | Boeing 737-300      |  4200
 CN1           | Cessna 208 Caravan  |  1200
 CR2           | Bombardier CRJ-200  |  2700
```

```
(9 rows)
```

For each aircraft type, a separate list of seats is supported. For example, in a small Cessna 208 Caravan, one can select the following seats:

```
SELECT   a.aircraft_code,
         a.model,
         s.seat_no,
         s.fare_conditions
FROM     aircrafts a
         JOIN seats s ON a.aircraft_code = s.aircraft_code
WHERE    a.model = 'Cessna 208 Caravan'
ORDER BY s.seat_no;
```

```
 aircraft_code |       model        | seat_no | fare_conditions
---------------+--------------------+---------+-----------------
 CN1           | Cessna 208 Caravan | 1A      | Economy
 CN1           | Cessna 208 Caravan | 1B      | Economy
 CN1           | Cessna 208 Caravan | 2A      | Economy
 CN1           | Cessna 208 Caravan | 2B      | Economy
 CN1           | Cessna 208 Caravan | 3A      | Economy
 CN1           | Cessna 208 Caravan | 3B      | Economy
 CN1           | Cessna 208 Caravan | 4A      | Economy
 CN1           | Cessna 208 Caravan | 4B      | Economy
 CN1           | Cessna 208 Caravan | 5A      | Economy
 CN1           | Cessna 208 Caravan | 5B      | Economy
 CN1           | Cessna 208 Caravan | 6A      | Economy
 CN1           | Cessna 208 Caravan | 6B      | Economy
(12 rows)
```

Bigger aircraft have more seats of various travel classes:

```
SELECT   s2.aircraft_code,
         string_agg (s2.fare_conditions || '(' || s2.num::text || ')',
                     ', ') as fare_conditions
FROM     (
          SELECT   s.aircraft_code, s.fare_conditions, count(*) as num
          FROM     seats s
          GROUP BY s.aircraft_code, s.fare_conditions
          ORDER BY s.aircraft_code, s.fare_conditions
         ) s2
GROUP BY s2.aircraft_code
ORDER BY s2.aircraft_code;
```

```
 aircraft_code |             fare_conditions
---------------+-----------------------------------------
 319           | Business(20), Economy(96)
 320           | Business(20), Economy(120)
 321           | Business(28), Economy(142)
 733           | Business(12), Economy(118)
 763           | Business(30), Economy(192)
 773           | Business(30), Comfort(48), Economy(324)
 CN1           | Economy(12)
 CR2           | Economy(50)
 SU9           | Business(12), Economy(85)
(9 rows)
```

The demo database contains the list of airports of almost all major Russian cities. Most cities have only one airport. The exceptions are:

```
SELECT   a.airport_code as code,
         a.airport_name,
         a.city,
         a.longitude,
         a.latitude,
         a.timezone
FROM     airports a
WHERE    a.city IN (
            SELECT   aa.city
            FROM     airports aa
            GROUP BY aa.city
            HAVING   COUNT(*) > 1
         )
ORDER BY a.city, a.airport_code;
```

```
 code |    airport_name     |   city    | longitude | latitude  |   timezone
------+---------------------+-----------+-----------+-----------+---------------
 DME  | Домодедово          | Москва    | 37.906111 | 55.408611 | Europe/Moscow
 SVO  | Шереметьево         | Москва    | 37.414589 | 55.972642 | Europe/Moscow
 VKO  | Внуково             | Москва    | 37.261486 | 55.591531 | Europe/Moscow
 ULV  | Баратаевка          | Ульяновск |   48.2267 | 54.268299 | Europe/Samara
 ULY  | Ульяновск-Восточный | Ульяновск |   48.8027 |    54.401 | Europe/Samara
(5 rows)
```

To learn about your flying options from one point to another, it is convenient to use the `routes` materialized view that aggregates information on all flights. For example, here are the destinations where you can get from Volgograd on specific days of the week, with flight duration:

```
SELECT r.arrival_city as city,
       r.arrival_airport as airport_code,
       r.arrival_airport_name as airport_name,
       r.days_of_week,
       r.duration
FROM   routes r
WHERE  r.departure_city = 'Волгоград';
```

```
      city      | airport_code |  airport_name  |  days_of_week   | duration
----------------+--------------+----------------+-----------------+----------
 Москва         | SVO          | Шереметьево    | {1,2,3,4,5,6,7} | 01:15:00
 Челябинск      | CEK          | Челябинск      | {1,2,3,4,5,6,7} | 01:50:00
 Ростов-на-Дону | ROV          | Ростов-на-Дону | {1,2,3,4,5,6,7} | 00:30:00
 Москва         | VKO          | Внуково        | {1,2,3,4,5,6,7} | 01:10:00
 Чебоксары      | CSY          | Чебоксары      | {1,2,3,4,5,6,7} | 02:45:00
 Томск          | TOF          | Богашёво       | {3}             | 03:50:00
(6 rows)
```

The database was formed at the moment returned by the `bookings.now()` function:

```
SELECT bookings.now() as now;
```

```
          now
------------------------
 2016-10-13 17:00:00+03
```

In relation to this moment, all flights are classified as past and future flights:

```
SELECT   status,
         count(*) as count,
         min(scheduled_departure) as min_scheduled_departure,
         max(scheduled_departure) as max_scheduled_departure
FROM     flights
GROUP BY status
ORDER BY min_scheduled_departure;
```

```
  status   | count | min_scheduled_departure | max_scheduled_departure
-----------+-------+-------------------------+-------------------------
 Arrived   | 16707 | 2016-09-13 00:50:00+03  | 2016-10-13 16:25:00+03
 Cancelled |   414 | 2016-09-16 10:35:00+03  | 2016-11-12 19:55:00+03
 Departed  |    58 | 2016-10-13 08:55:00+03  | 2016-10-13 16:50:00+03
 Delayed   |    41 | 2016-10-13 14:15:00+03  | 2016-10-14 16:25:00+03
 On Time   |   518 | 2016-10-13 16:55:00+03  | 2016-10-14 17:00:00+03
 Scheduled | 15383 | 2016-10-14 17:05:00+03  | 2016-11-12 19:40:00+03
(6 rows)
```

Let's find the next flight from Ekaterinburg to Moscow. The `flight` table is not very convenient for such queries, as it does not include information on the cities of departure and arrival. That is why we will use the `flights_v` view:

```
\x
SELECT   f.*
FROM     flights_v f
WHERE    f.departure_city = 'Екатеринбург'
AND      f.arrival_city = 'Москва'
AND      f.scheduled_departure > bookings.now()
ORDER BY f.scheduled_departure
LIMIT    1;
```

```
-[ RECORD 1 ]-------------+-----------------------
flight_id                 | 10927
flight_no                 | PG0226
scheduled_departure       | 2016-10-14 07:10:00+03
scheduled_departure_local | 2016-10-14 09:10:00
scheduled_arrival         | 2016-10-14 08:55:00+03
scheduled_arrival_local   | 2016-10-14 08:55:00
scheduled_duration        | 01:45:00
departure_airport         | SVX
departure_airport_name    | Кольцово
departure_city            | Екатеринбург
arrival_airport           | SVO
arrival_airport_name      | Шереметьево
arrival_city              | Москва
status                    | On Time
aircraft_code             | 773
actual_departure          |
actual_departure_local    |
actual_arrival            |
actual_arrival_local      |
actual_duration           |
```

Note that the `flights_v` view shows both Moscow time and local time at the airports of departure and arrival.

L.5.3. Bookings

Each booking can include several tickets, one for each passenger. The ticket, in its turn, can include several flight segments. The complete information about the booking is stored in three tables: `bookings`, `tickets`, and `ticket_flights`.

Let's find several most expensive bookings:

```
SELECT   *
FROM     bookings
ORDER BY total_amount desc
LIMIT    10;

 book_ref |       book_date        | total_amount
----------+------------------------+--------------
 3B54BB   | 2016-09-02 16:08:00+03 |   1204500.00
 3AC131   | 2016-09-28 00:06:00+03 |   1087100.00
 65A6EA   | 2016-08-31 05:28:00+03 |   1065600.00
 D7E9AA   | 2016-10-06 04:29:00+03 |   1062800.00
 EF479E   | 2016-09-30 14:58:00+03 |   1035100.00
 521C53   | 2016-09-05 08:25:00+03 |    985500.00
 514CA6   | 2016-09-24 04:07:00+03 |    955000.00
 D70BD9   | 2016-09-02 11:47:00+03 |    947500.00
 EC7EDA   | 2016-08-30 15:13:00+03 |    946800.00
 8E4370   | 2016-09-25 01:04:00+03 |    945700.00
(10 rows)
```

Let's take a look at the tickets included into the booking with code `521C53`:

```
SELECT ticket_no,
       passenger_id,
       passenger_name
FROM   tickets
WHERE  book_ref = '521C53';

   ticket_no   | passenger_id |   passenger_name
---------------+--------------+--------------------
 0005432661914 | 8234 547529  | IVAN IVANOV
 0005432661915 | 2034 201228  | ANTONINA KUZNECOVA
(2 rows)
```

If we would like to know, which flight segments are included into Antonina Kuznecova's ticket, we can use the following query:

```
SELECT   to_char(f.scheduled_departure, 'DD.MM.YYYY') as when,
         f.departure_city || '(' || f.departure_airport || ')' as departure,
         f.arrival_city || '(' || f.arrival_airport || ')' as arrival,
         tf.fare_conditions as class,
         tf.amount
FROM     ticket_flights tf
         JOIN flights_v f ON tf.flight_id = f.flight_id
WHERE    tf.ticket_no = '0005432661915'
ORDER BY f.scheduled_departure;

    when    |     departure     |      arrival      |  class   |  amount
------------+-------------------+-------------------+----------+-----------
 26.09.2016 | Москва(SVO)       | Анадырь(DYR)      | Business | 185300.00
 30.09.2016 | Анадырь(DYR)      | Хабаровск(KHV)    | Business |  92200.00
 01.10.2016 | Хабаровск(KHV)     | Благовещенск(BQS) | Business |  18000.00
 06.10.2016 | Благовещенск(BQS)  | Хабаровск(KHV)     | Business |  18000.00
 10.10.2016 | Хабаровск(KHV)     | Анадырь(DYR)       | Economy  |  30700.00
 15.10.2016 | Анадырь(DYR)       | Москва(SVO)        | Business | 185300.00
(6 rows)
```

As we can see, high booking cost is explained by multiple long-haul flights in business class.

Some of the flight segments in this ticket have earlier dates than the `bookings.now()` return value: it means that these flights had already happened. The last flight had not happened yet at the time of the database creation. After the check-in, a boarding pass with the allocated seat number is issued. We can check the exact seats occupied by Antonina (note the outer left join with table `boarding_passes`):

```
SELECT   to_char(f.scheduled_departure, 'DD.MM.YYYY') as when,
         f.departure_city || '(' || f.departure_airport || ')' as departure,
         f.arrival_city || '(' || f.arrival_airport || ')' as arrival,
         f.status,
         bp.seat_no
FROM     ticket_flights tf
         JOIN flights_v f ON tf.flight_id = f.flight_id
         LEFT JOIN boarding_passes bp ON tf.flight_id = bp.flight_id
                                     AND tf.ticket_no = bp.ticket_no
WHERE    tf.ticket_no = '0005432661915'
ORDER BY f.scheduled_departure;
```

```
    when    |     departure      |      arrival       |  status   | seat_no
------------+--------------------+--------------------+-----------+---------
 26.09.2016 | Москва(SVO)        | Анадырь(DYR)       | Arrived   | 5C
 30.09.2016 | Анадырь(DYR)       | Хабаровск(KHV)     | Arrived   | 1D
 01.10.2016 | Хабаровск(KHV)     | Благовещенск(BQS) | Arrived   | 2C
 06.10.2016 | Благовещенск(BQS)  | Хабаровск(KHV)     | Arrived   | 2D
 10.10.2016 | Хабаровск(KHV)     | Анадырь(DYR)       | Arrived   | 20B
 15.10.2016 | Анадырь(DYR)       | Москва(SVO)        | Scheduled |
(6 rows)
```

L.5.4. New Booking

Let's try to send Aleksandr Radishchev from Saint Petersburg to Moscow — the route that made him famous. Naturally, he will travel for free and in business class. We have already found a flight for tomorrow, and a return flight a week later.

```
BEGIN;

INSERT INTO bookings (book_ref, book_date, total_amount)
VALUES      ('_QWE12', bookings.now(), 0);

INSERT INTO tickets (ticket_no, book_ref, passenger_id, passenger_name)
VALUES      ('_000000000001', '_QWE12', '1749 051790', 'ALEKSANDR RADISHCHEV');

INSERT INTO ticket_flights (ticket_no, flight_id, fare_conditions, amount)
VALUES      ('_000000000001', 9720, 'Business', 0),
            ('_000000000001', 6662, 'Business', 0);

COMMIT;
```

To avoid conflicts with the range of values present in the database, identifiers are started with an underscore.

We will check in Aleksandr for tomorrow's flight right away:

```
INSERT INTO boarding_passes (ticket_no, flight_id, boarding_no, seat_no)
VALUES      ('_000000000001', 9720, 1, '1A');
```

Now let's check the booking information:

```
SELECT   b.book_ref,
         t.ticket_no,
         t.passenger_id,
         t.passenger_name,
         tf.fare_conditions,
         tf.amount,
         f.scheduled_departure_local,
         f.scheduled_arrival_local,
         f.departure_city || '(' || f.departure_airport || ')' as departure,
         f.arrival_city || '(' || f.arrival_airport || ')' as arrival,
         f.status,
         bp.seat_no
FROM     bookings b
         JOIN tickets t ON b.book_ref = t.book_ref
         JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
         JOIN flights_v f ON tf.flight_id = f.flight_id
         LEFT JOIN boarding_passes bp ON tf.flight_id = bp.flight_id
                                     AND tf.ticket_no = bp.ticket_no
WHERE    b.book_ref = '_QWE12'
ORDER BY t.ticket_no, f.scheduled_departure;

-[ RECORD 1 ]-------------+---------------------
book_ref                  | _QWE12
ticket_no                 | _000000000001
passenger_id              | 1749 051790
passenger_name            | ALEKSANDR RADISHCHEV
fare_conditions           | Business
amount                    | 0.00
scheduled_departure_local | 2016-10-14 08:45:00
scheduled_arrival_local   | 2016-10-14 09:35:00
departure                 | Санкт-Петербург(LED)
arrival                   | Москва(SVO)
status                    | On Time
seat_no                   | 1A
-[ RECORD 2 ]-------------+---------------------
book_ref                  | _QWE12
ticket_no                 | _000000000001
passenger_id              | 1749 051790
passenger_name            | ALEKSANDR RADISHCHEV
fare_conditions           | Business
amount                    | 0.00
scheduled_departure_local | 2016-10-21 09:20:00
scheduled_arrival_local   | 2016-10-21 10:10:00
departure                 | Москва(SVO)
arrival                   | Санкт-Петербург(LED)
status                    | Scheduled
seat_no                   |
```

We hope that these simple examples helped you get an idea of this demo database.