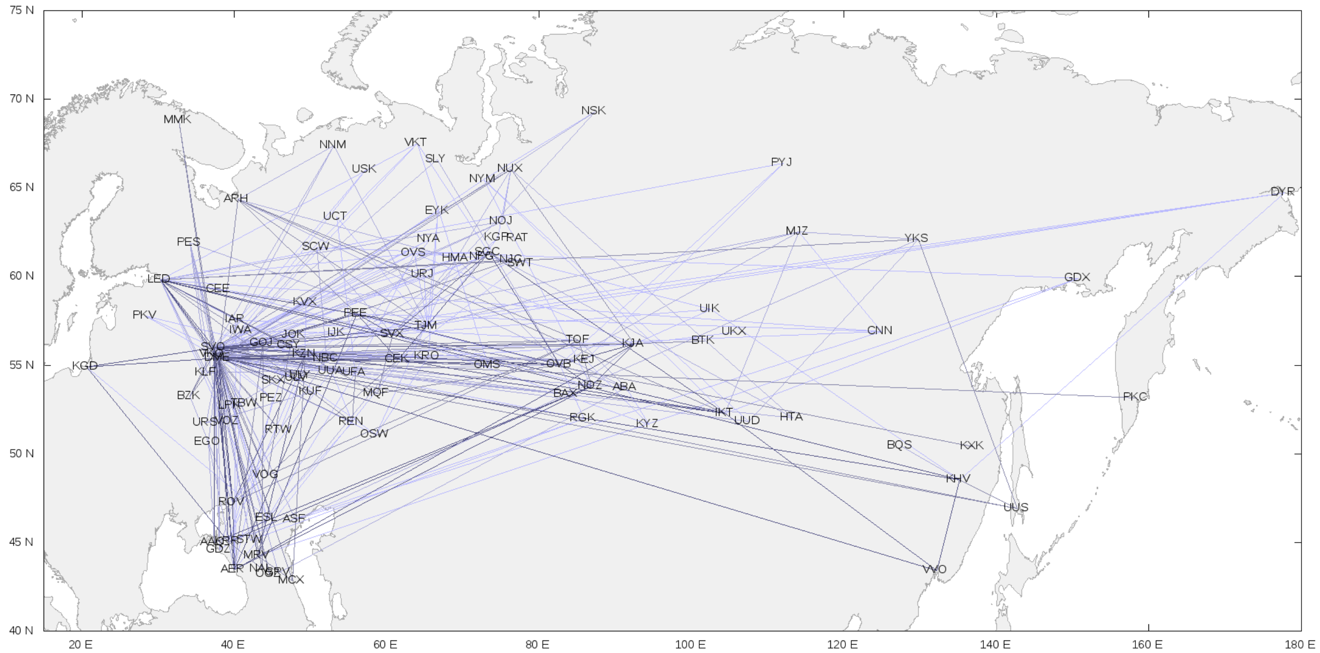
Appendix J. Demo Database “Airlines”

This is an overview of a demo database for Postgres Pro. 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 J.1 for details.

Figure J.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*.

J.1. Installation

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

- *demo-small-en.zip* (21 MB) — flight data for one month (DB size is about 300 MB)
- *demo-medium-en.zip* (62 MB) — flight data for three months (DB size is about 700 MB)
- *demo-big-en.zip* (232 MB) — flight data for one year (DB size is about 2.5 GB)

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.

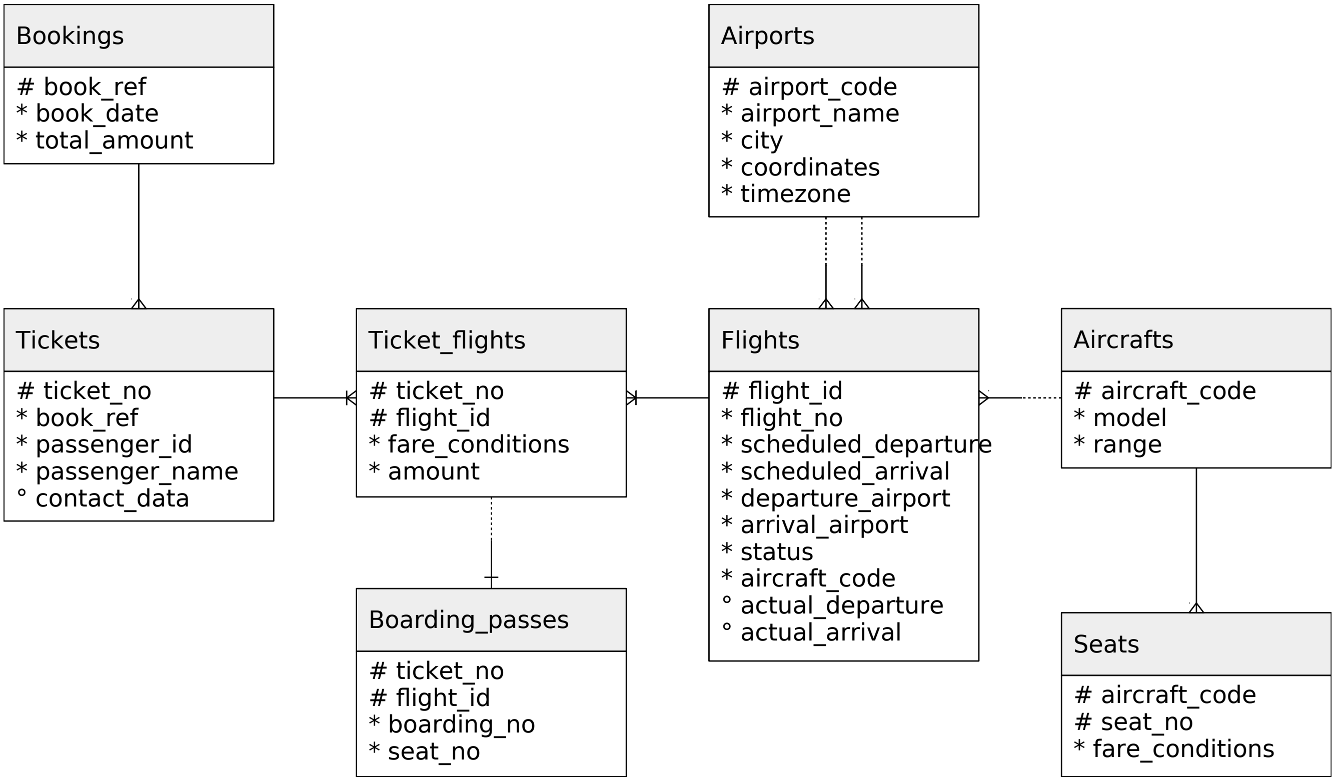
The files include an SQL script that creates the `demo` database and fills it with data (virtually, it is a backup copy created with the pg_dump utility). The owner of the `demo` database will be the DBMS user who runs the script. For example, to create the small database, run the script as the user `postgres` by means of psql:

```
psql -f demo_small_YYYYMMDD.sql -U postgres
```

Note that if the `demo` database already exists, it will be deleted and recreated!

J.2. Schema Diagram

Figure J.2. Bookings Schema Diagram



J.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).

J.4. Schema Objects

J.4.1. List of Relations

```
       Name      |     Type      |
-----------------+---------------+--------+--------+--------+--------------------------
 aircrafts       | view          |        |        |        | Aircraft
 aircrafts_data  | table
 airports        | view
 airports_data   | table         |
 boarding_passes | table         |  31 MB |
 bookings        | table                     30 MB | 105 MB | Bookings
 flights         | table
 flights_v       | view
 routes          | view
 seats           | table         |
 ticket_flights  | table         |  64 MB |
 tickets         | table         |
```

J.4.2. View `bookings.aircrafts`

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

The value of the `model` field is selected according to the chosen language. See Section J.4.15 for details.

```
    Column     |  Type   | Modifiers    |             Description
---------------+---------+--------------+------------------------------------
 aircraft_code | char(3) | not null
 model         | text    | not null
 range         | integer | not null
View definition:
 SELECT ml.aircraft_code,
    ml.model ->> lang() AS model,
    ml.range
   FROM aircrafts_data ml;
```

J.4.3. Table `bookings.aircrafts_data`

This is the base table for the `aircrafts` view. The `model` field of this table contains translations of aircraft models to different languages, in the JSONB format. In most cases, this table is not supposed to be used directly.

```
    Column     |  Type   | Modifiers    |             Description
---------------+---------+--------------+------------------------------------
 aircraft_code | char(3) | not null
 model         | jsonb   | not null
 range         | integer | not null
Indexes:
    PRIMARY KEY, btree (aircraft_code)
Check constraints:
    CHECK (range > 0)
```

```
Referenced by:
    TABLE "flights" FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts_data(aircraft_code)
    TABLE "seats" FOREIGN KEY (aircraft_code)
        REFERENCES aircrafts_data(aircraft_code) ON DELETE CASCADE
```

J.4.4. View 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 view also includes coordinates (coordinates) and the time zone (timezone).

The values of the airport_name and city fields are selected according to the chosen language. See Section J.4.15 for details.

```
   Column     |  Type   | Modifiers    |                 Description
--------------+---------+--------------+--------------------------------------------
 airport_code | char(3) | not null     | Airport code
 airport_name | text                   | Airport name
 city         | text                   | City
 coordinates  | point   | not null     | Airport coordinates (lo
 timezone     | text    | not null
View definition:
 SELECT ml.airport_code,
    ml.airport_name ->> lang() AS airport_name,
    ml.city ->> lang() AS city,
    ml.coordinates,
    ml.timezone
   FROM airports_data ml;
```

J.4.5. Table bookings.airports_data

This is the base table for the airports view. This table contains translations of airport_name and city values to different languages, in the JSONB format. In most cases, this table is not supposed to be used directly.

```
   Column     |  Type   | Modifiers    |                 Description
--------------+---------+--------------+--------------------------------------------
 airport_code | char(3) | not null
 airport_name | jsonb
 city         | jsonb
 coordinates  | point   | not null     | Airport coordinates (lo
 timezone     | text    | not null
Indexes:
    PRIMARY KEY, btree (airport_code)
Referenced by:
    TABLE "flights" FOREIGN KEY (arrival_airport)
        REFERENCES airports_data(airport_code)
    TABLE "flights" FOREIGN KEY (departure_airport)
        REFERENCES airports_data(airport_code)
```

J.4.6. 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
 flight_id   | integer
 boarding_no | integer    | not null
 seat_no     | varchar(4) | not null
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)
```

J.4.7. 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
 book_date    | timestamptz   | not null
 total_amount | numeric(10,2) | not null
Indexes:
    PRIMARY KEY, btree (book_ref)
Referenced by:
    TABLE "tickets" FOREIGN KEY (book_ref) REFERENCES bookings(book_ref)
```

J.4.8. 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
 flight_no           | char(6)
 scheduled_departure | timestamptz | not null     | Scheduled departure time
 scheduled_arrival   | time                        | Scheduled arrival time
 departure_airport   | char                        | Airport of departure
 arrival_airport     | char(3                      | Airport of arrival
 status              | varchar(20) | n            | Flight status
 aircraft_code       | char(3)                     | Aircraft code, IATA
 actual_departure                                  | Actual departure time
 actual_arrival                                    | 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)
```

J.4.9. 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
```

J.4.10. 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                      | 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)
```

J.4.11. 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)     |              | Booking number
 passenger_id   | varchar(20) | not null     | Passenger ID
 passenger_name | text                       | Passenger name
 contact_data   |                            | 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)
```

J.4.12. 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_no                 | char(6)
 scheduled_departure       | timestamptz | Sche
 scheduled_departure_local | timestamp   | Scheduled departure time,
                           |             | local time at the point of departure
 scheduled_arrival         | timestamptz | Sche
 scheduled_arrival_local   |
                           |             | local time
 scheduled_duration        | in
 departure_airport         |
 departure_airport_name    | tex
 departure_city
 arrival_airport
 arrival_airport_name      | t
 arrival_city
 status                    | varchar(20) | Flight
 aircraft_code             | char(3)     | Aircr
 actual_departure          | timestamptz | Actu
 actual_departure_local    |
                           |             | local ti
 actual_arrival            | timestamptz | Actu
 actual_arrival_local
                           |             | local time
 actual_duration           |
```

J.4.13. 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 is shown in the `routes` view.

```
        Column          |   Type    |              Description
------------------------+-----------+-------------------------------------
 flight_no              | char(6)
 departure_airport      | char(3)
 departure_airport_name | text      | Departure airport name
 departure_city         | text
 arrival_airport        | char(3)
 arrival_airport_name   | text      |
 arrival_city           | text      | Ci
 aircraft_code          | char(3)   | Aircr
 duration               | interval  | Flight
 days_of_week           | integer[] | Days of the week on which flights
```

J.4.14. Function `bookings.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 August 15, 2017.

J.4.15. Function `bookings.lang`

Some fields in the demo database are available in English and Russian. Translations to other languages are not provided, but are easy to add. The `bookings.lang` returns the value of the *`bookings.lang`* parameter, that is, the language in which these fields will be displayed.

This function is used in the `aircrafts` and `airports` views and is not intended to be used directly in queries.

J.5. Usage

J.5.1. Schema `bookings`

The `bookings` schema contains all objects of the demo database. When you connect to the database, `search_path` configuration parameter is automatically set to `bookings, public`, so you do not need to specify the schema name explicitly.

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

J.5.2. Translations

By default, values of several translatable fields are shown in Russian. These are `airport_name` and `city` of the `airports` view, as well as `model` of the `aircrafts` view.

You can choose to display these fields in another language (although only the English translation is provided in the demo database). To switch to English, set the *`bookings.lang`* parameter to `en`. It may be convenient to choose the language at the database level:

```
ALTER DATABASE demo SET bookings.lang = en;
```

You have to reconnect to the database for this command to take effect. For other methods of settings configuration parameters, see Section 18.1.

In the examples below, the English language is selected for translatable fields.

J.5.3. 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 August 15, 2017. 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      |
 SU9           | Sukhoi SuperJet-100 |  3000
 320           | Airbus A320-200     |
 321           | Airbus A321-200     |
 319           | Airbus A319-100     |
 733           | Boeing 737-300      |
 CN1           | Cessna 208 Caravan
 CR2           | Bombardier CRJ-200
(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
         string_agg (s2.fare_conditions || '(' || s2.num::text || ')',
                     ', ') as fare_conditions
FROM
          SELECT   s.aircraft_code, s.fare_conditions,
          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(
 321           | Business(28), Economy(
 733           | Business(12), Economy(
 763           | Business(30), Economy(
 773           | Business(30), Comfort(48), Economy(
 CN1           | Economy
 CR2           | Economy
 SU9           | Business(12), Economy
(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.coordinates
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    |           coordinates
------+-------------------------+-----------+-------------------------
 DME  | Domodedovo              | Moscow
      | International Airport
 SVO  | Sheremetyevo
      | International Airport
 VKO  | Vnukovo                 | Mo
      | International Airport
 ULV  | Ulyan vsk               | Ulyanovsk | (48.22669982
      | Baratayevka Airport
 ULY  | Ulyanovsk East Airport | Ulyanovsk | (48.8027000427246,54.4010009765625)
(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 code,
       r.arrival_airport_name as airport_name,
       r.days_of_week,
       r.duration
FROM   rout
WHERE  r.departure_city = 'Volgograd';

    city     | code |            airport_name             |  days_of_week   | duration
-------------+------+-------------------------------------+-----------------+----------
 Moscow      | SVO  | Sheremetyevo International Airport | {1,2,3,4
 Chelyabinsk | CEK  | Chelyabinsk Balandino Airport
 Rostov      | ROV  | Rostov-on-Don Airpor
 Moscow      | VKO  | Vnukovo Inte
 Cheboksary  | CSY  | Cheboksary Airport
 Tomsk       | TOF  | Bogashevo Airport
(6 rows)
```

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

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

          now
------------------------
 2017-08-15 18:00:00+03
```

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

```
SELECT   s
         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   |
 Cancelled |   414
 Departed       58
 Delayed        41
 On Time       518
 Scheduled
(6 rows)
```

Let's find the next flight from Yekaterinburg 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 = 'Yekaterinburg'
AND      f.arrival_city = 'Moscow'
AND      f.scheduled_departure > bookings.now()
ORDER BY f.scheduled_departure
LIMIT    1;
```

```
-[ RECORD 1 ]-------------+---------------------------------
flight_id                 | 10927
flight_no                 | PG0226
scheduled_departure       | 2017-08-16 08:10:00+03
scheduled_departure_local | 2017-08-16 10:10:00
scheduled_arrival         | 2017-
scheduled_arrival_local
scheduled_duration
departure_airport
departure_airport_name
departure_city            |
arrival_airport
arrival_airport_name      | Sheremetyevo I
arrival_city
status                    | O
aircraft_code
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.

J.5.4. 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   | 2017-07-05 17:08:00+03 |   1204500.00
 3AC131   | 2017-07-31 01:06:00+03 |   1087100.00
 65A6EA   | 2017-07-03 06:28:00+03 |   1065600.00
 D7E9AA   | 2017-08-08 05:29:00+03 |   1062800.00
 EF479E   | 2017-08-02 15:58:00+03 |   1035100.00
 521C53   | 2017-07-08 09:25:00+03 |    985500.00
 514CA6   | 2017-07-27 05:07:00+03 |    955000.00
 D70BD9   | 2017-07-05 12:47:00+03 |    947500.00
 EC7EDA   | 2017-07-02 16:13:00+03 |    946800.00
 8E4370   | 2017-07-28 02:04:00+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_flight
         JOIN flights_v f ON tf.flight_id = f.flight_id
WHERE    tf.ticket_no = '000543266
ORDER BY f.scheduled_departure;
```

```
    when    |      departure       |       arrival        |  class   |
------------+----------------------+----------------------+----------+-----------
 29.07.2017 | Moscow (SVO)         | Anadyr (DYR)         | Business | 185300.00
 02.08.2017 | Anadyr (DYR)         | Khabarovsk (KHV)     | Business |
 03.08.2017 | Khabarovsk (KHV)     | Blagoveshchensk (BQS)| Business |
 08.08.2017 | Blagoveshchensk (BQS)| Khabarovsk (KHV)     | Business |
 12.08.2017 | Khabarovsk (KHV)     | Anadyr (DYR)         | Economy  |  30700.00
 17.08.2017 | Anadyr (DYR)         | Moscow (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
------------+----------------------+----------------------+-----------+---------
 29.07.2017 | Moscow (SVO)         | Anadyr (DYR)         | Arrived
 02.08.2017 | Anadyr (DYR)         | Khabarovsk (KHV)     | Arrived
 03.08.2017 | Khabarovsk (KHV)     | Bla                              | 2C
 08.08.2017 | Blagoveshchensk (BQS)| Khabarovsk (KHV)                 | 2D
 12.08.2017 | Khabarovsk (KHV)     | Anadyr (DYR)
 17.08.2017 | Anadyr (DYR)         | Moscow (SVO)
(6 rows)
```

J.5.5. 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', 8525, 'Business', 0),
            ('_000000000001', 4967, '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', 8525, 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 | 2017-08-16 09:45:00
scheduled_arrival_local
departure                 | St. Petersb
arrival                   | Mosco
status                    | O
seat_no
-[ RECORD 2 ]-------------+---------------------
book_ref
ticket_no                 | _000
passenger_id              |
passenger_name            | ALEKSA
fare_conditions
amount
scheduled_departure_local | 2017-08-23 10:20:00
scheduled_arrival_local
departure                 | Mos
arrival                   | St. Petersbur
status                    | Sch
seat_no
```

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