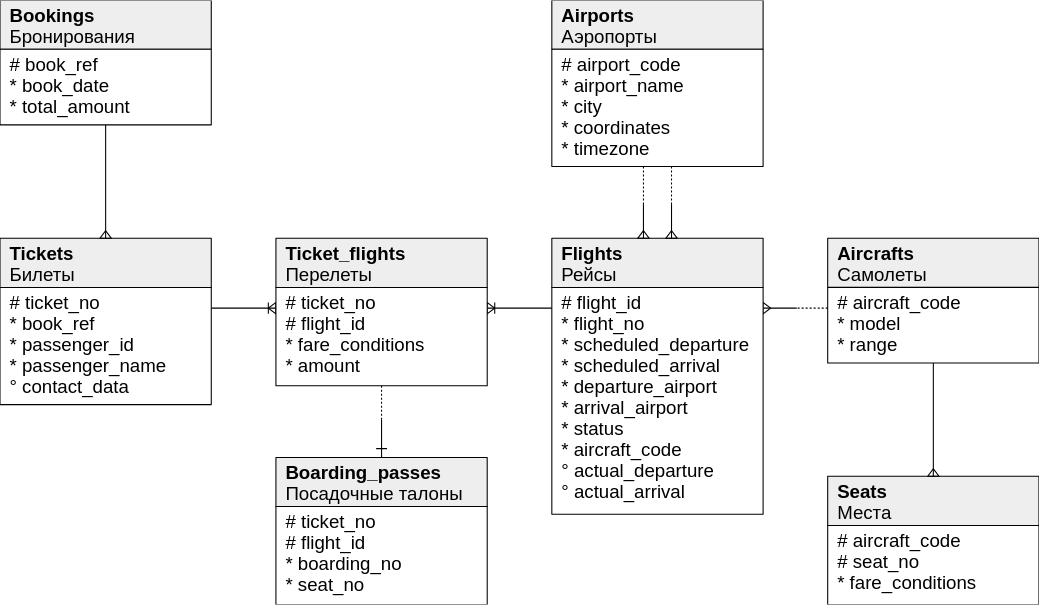
**Домашнее задание №3. Базы данных**

* Полезный [пример](https://github.com/inponomarev/java-mipt/tree/master/jdbc-example) по JDBC.
* Для сдачи задачу код, тесты и pom.xml нужно залить в репозиторий: <http://gitlab.atp-fivt.org/java2021/XXXX-hw3>.

# Исходные данные

## Описание базы

Имеется база данных авиаперевозок по России в 2017 г.



Основной сущностью является бронирование (bookings). В одно бронирование можно включить несколько пассажиров, каждому из которых выписывается отдельный билет (tickets). Билет имеет уникальный номер и содержит информацию о пассажире. Как имя, так и номер документа пассажира могут меняться с течением времени, так что невозможно однозначно найти все билеты одного человека; для простоты можно считать, что все пассажиры уникальны.

Билет включает один или несколько перелетов (ticket\_flights). Несколько перелетов могут включаться в билет в случаях, когда нет прямого рейса, соединяющего пункты отправления и назначения (полет с пересадками), либо когда билет взят «туда и обратно». В схеме данных нет жёсткого ограничения, но предполагается, что все билеты в одном бронировании имеют одинаковый набор перелетов.

При регистрации на рейс пассажиру выдаётся посадочный талон (boarding\_passes), в котором указано место в самолете. Пассажир может зарегистрироваться только на тот рейс, который есть у него в билете. Комбинация рейса и места в самолете должна быть уникальной, чтобы не допустить выдачу двух посадочных талонов на одно место.

Схема данных не контролирует, что места в посадочных талонах соответствуют имеющимся в самолете (такая проверка может быть сделана с использованием табличных триггеров или в приложении).

## Описание таблиц

### Таблица “aircrafts”

Каждая модель воздушного судна идентифицируется своим трёхзначным кодом (aircraft\_code). Указывается также название модели (model) и максимальная дальность полета в километрах (range). Поле model этой таблицы содержит переводы моделей самолётов на разные языки, в формате JSONB.

Столбец | Тип | Модификаторы | Описание

---------------+---------+--------------+-----------------------------------

aircraft\_code | char(3) | not null | Код самолета, IATA

model | jsonb | not null | Модель самолета

range | integer | not null | Максимальная дальность полета, км

### *Таблица “*airports”

Аэропорт идентифицируется трехбуквенным кодом (airport\_code) и имеет своё имя (airport\_name).

Для города не предусмотрено отдельной сущности, но введено поле с названием города (city), позволяющее найти аэропорты одного города. Это представление также включает координаты аэропорта (coordinates) и часовой пояс (timezone). Поля airport\_name и city содержат переводы значений на разные языки, в формате JSONB.

Столбец | Тип | Модификаторы | Описание

--------------+---------+--------------+--------------------------------------------

airport\_code | char(3) | not null | Код аэропорта

airport\_name | jsonb | not null | Название аэропорта

city | jsonb | not null | Город

coordinates | point | not null | Координаты аэропорта (долгота и широта)

timezone | text | not null | Часовой пояс аэропорта

### Таблица “boarding\_passes”

При регистрации на рейс, которая возможна за сутки до плановой даты отправления, пассажиру выдаётся посадочный талон. Он идентифицируется также, как и перелёт — номером билета и номером рейса. Посадочным талонам присваиваются последовательные номера (boarding\_no) в порядке регистрации пассажиров на рейс (этот номер будет уникальным только в пределах данного рейса). В посадочном талоне указывается номер места (seat\_no).

Столбец | Тип | Модификаторы | Описание

-------------+------------+--------------+--------------------------

ticket\_no | char(13) | not null | Номер билета

flight\_id | integer | not null | Идентификатор рейса

boarding\_no | integer | not null | Номер посадочного талона

seat\_no | varchar(4) | not null | Номер места

### Таблица “bookings”

Пассажир заранее (book\_date, максимум за месяц до рейса) бронирует билет себе и, возможно, нескольким другим пассажирам. Бронирование идентифицируется номером (book\_ref, шестизначная комбинация букв и цифр). Поле total\_amount хранит общую стоимость включённых в бронирование перелетов всех пассажиров.

Столбец | Тип | Модификаторы | Описание

--------------+---------------+--------------+---------------------------

book\_ref | char(6) | not null | Номер бронирования

book\_date | timestamptz | not null | Дата бронирования

total\_amount | numeric(10,2) | not null | Полная сумма бронирования

### Таблица “flights”

Естественный ключ таблицы рейсов состоит из двух полей — номера рейса (flight\_no) и даты отправления (scheduled\_departure). Чтобы сделать внешние ключи на эту таблицу компактнее, в качестве первичного используется суррогатный ключ (flight\_id).

Рейс всегда соединяет две точки — аэропорты вылета (departure\_airport) и прибытия (arrival\_airport). Такое понятие, как «рейс с пересадками» отсутствует: если из одного аэропорта до другого нет прямого рейса, в билет просто включаются несколько необходимых рейсов.

У каждого рейса есть запланированные дата и время вылета (scheduled\_departure) и прибытия (scheduled\_arrival). Реальные время вылета (actual\_departure) и прибытия (actual\_arrival) могут отличаться: обычно не сильно, но иногда и на несколько часов, если рейс задержан.

Статус рейса (status) может принимать одно из следующих значений:

* **Scheduled.** Рейс доступен для бронирования. Это происходит за месяц до плановой даты вылета; до этого запись о рейсе не существует в базе данных.
* **On Time.** Рейс доступен для регистрации (за сутки до плановой даты вылета) и не задержан.
* **Delayed.** Рейс доступен для регистрации (за сутки до плановой даты вылета), но задержан.
* **Departed.** Самолет уже вылетел и находится в воздухе.
* **Arrived.** Самолет прибыл в пункт назначения.
* **Cancelled.** Рейс отменён.

Столбец | Тип | Модификаторы | Описание

---------------------+-------------+--------------+-----------------------------

flight\_id | serial | not null | Идентификатор рейса

flight\_no | char(6) | not null | Номер рейса

scheduled\_departure | timestamptz | not null | Время вылета по расписанию

scheduled\_arrival | timestamptz | not null | Время прилёта по расписанию

departure\_airport | char(3) | not null | Аэропорт отправления

arrival\_airport | char(3) | not null | Аэропорт прибытия

status | varchar(20) | not null | Статус рейса

aircraft\_code | char(3) | not null | Код самолета, IATA

actual\_departure | timestamptz | | Фактическое время вылета

actual\_arrival | timestamptz | | Фактическое время прилёта

### Таблица “seats”

Места определяют схему салона каждой модели. Каждое место определяется своим номером (seat\_no) и имеет закреплённый за ним класс обслуживания (fare\_conditions) — **Economy**, **Comfort** или **Business**.

Столбец | Тип | Модификаторы | Описание

-----------------+-------------+--------------+--------------------

aircraft\_code | char(3) | not null | Код самолета, IATA

seat\_no | varchar(4) | not null | Номер места

fare\_conditions | varchar(10) | not null | Класс обслуживания

### Таблица “ticket\_flights”

Перелёт соединяет билет с рейсом и идентифицируется их номерами. Для каждого перелета указываются его стоимость (amount) и класс обслуживания (fare\_conditions).

Столбец | Тип | Модификаторы | Описание

-----------------+---------------+--------------+---------------------

ticket\_no | char(13) | not null | Номер билета

flight\_id | integer | not null | Идентификатор рейса

fare\_conditions | varchar(10) | not null | Класс обслуживания

amount | numeric(10,2) | not null | Стоимость перелета

### Таблица “tickets”

Билет имеет уникальный номер (ticket\_no), состоящий из 13 цифр. Билет содержит идентификатор пассажира (passenger\_id) — номер документа, удостоверяющего личность, — его фамилию и имя (passenger\_name) и контактную информацию (contact\_data). Ни идентификатор пассажира, ни имя не являются постоянными (можно поменять паспорт, можно сменить фамилию), поэтому однозначно найти все билеты одного и того же пассажира невозможно.

Столбец | Тип | Модификаторы | Описание

----------------+-------------+--------------+-----------------------------

ticket\_no | char(13) | not null | Номер билета

book\_ref | char(6) | not null | Номер бронирования

passenger\_id | varchar(20) | not null | Идентификатор пассажира

passenger\_name | text | not null | Имя пассажира

contact\_data | jsonb | | Контактные данные пассажира

## Ссылки для скачивания

Данные каждой таблицы хранятся в CSV-файле. Каждый файл доступен по ссылке: [https://storage.yandexcloud.net/airtrans-small/**{table\_name}**.csv](https://storage.yandexcloud.net/airtrans-small/aircrafts.csv). Например для таблицы “aircrafts” будет такая ссылка: [https://storage.yandexcloud.net/airtrans-small/**aircrafts**.csv](https://storage.yandexcloud.net/airtrans-small/aircrafts.csv)

[Вся база одним архивом](https://storage.yandexcloud.net/airtrans-small/airtrans.zip).

# Задачи

## Задача A. Загрузка данных

Реализовать код, который выкачивает данные, парсит и загружает в базу. На данном этапе код должен:

1. Создать базу данных и все необходимые таблицы.
2. Заполнить БД данными из скачанных файлов.

## Задача B. Работа с БД

Программа должна выполнять такие запросы:

| 1 | Вывести города, в которых несколько аэропортов.   | Город | Список аэропортов | | --- | --- | | Москва | SVO, DME... | |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | Вывести города, из которых чаще всего отменяли рейсы.   | Город | Кол-во отмененных рейсов | | --- | --- | | Москва | 5 | | Ульяновск | 4 | |  |
| 3 | В каждом городе вылета найти самый короткий маршрут. Отсортировать по продолжительности.   | Город | Пункт прибытия | Средняя продолжительность полёта | | --- | --- | --- | |  |  |  |   Среднюю продолжительность вычисляем по фактической продолжительности рейса. Обратите внимание, что некоторые рейсы могут быть не закончены в момент дампа базы, поэтому прибытие у них будет NULL. Такие рейсы не учитываем при подсчёте. |  |
| 4 | Найти кол-во отмен рейсов по месяцам.   | Месяц | Кол-во отмен | | --- | --- | |  |  |   Для получения месяца использовать поле scheduled\_departure в таблице flights. | x |
| 5 | Выведите кол-во рейсов в Москву и из Москвы по дням недели за весь наблюдаемый период.  Для задачи С построить 2 гистограммы:   * кол-во рейсов в Москву по дням недели * кол-во рейсов из Москвы по дням недели   Вывести обе гистограммы на 1 графике. | х |
| 6 | Отменить все рейсы самолета, заданной модели (модель - параметр). Все билеты, относящиеся к удаленным рейсам - удалить. |  |
| 7 | В связи с пандемией COVID-2017 все рейсы, прибывающие в Москву и отбывающие из неё, запланированные на даты в интервале, заданном параметром (например, [01.08.17, 15.08.17]), были отменены. Перевести соответствующие рейсы в Cancelled, а также посчитать убыток, который теряют компании-перевозчики по дням.  Построить гистограмму убытков по дням. | x |
| 8 | Написать запрос на добавление нового билета. При указании рейса и места в самолёте делать проверку, что соответствующие рейс и место существуют. |  |

Каждое из 8 действий должно быть реализовано в отдельном методе. Если действие реализуется несколькими запросами, оно должно работать в рамках 1 транзакции.

## Задача С. Представление результатов

Для представления результатов запросов в виде таблиц предлагается использовать библиотеку Apache POI, а для построения графиков JFreeChart.

### Генерация таблиц

1. Для каждого SELECT-запроса из задачи B реализовать метод, который формирует Excel-файл с результатами. Таблица должна содержать:

* заголовки (1-я строка),
* результат запроса[[1]](#footnote-0).

2. Заголовки должны иметь стиль ячеек, отличный от ячеек с результатом. Ячейки с заголовками должны быть заморожены.

### Построение графиков

Для запросов, помеченных “х” в правой колонке нужно реализовать метод для построения диаграмм. На выходе должна быть картинка с диаграммой.

## Задача D. GitLab-CI.

Для корректного запуска CI-job выбирайте Runner с тегом docker-atp. Подробнее про теги в GitLab-CI можно прочитать [здесь](https://docs.gitlab.com/ce/ci/yaml/README.html#tags). Для передачи файлов (например., \*.jar) между job’ами используйте [cache](https://docs.gitlab.com/ee/ci/caching/). Например:

| cache:  paths:  - ./.m2/repository  key: "$CI\_BUILD\_REF\_NAME" |
| --- |

Pipeline в GitLab-CI должен содержать такие stages:

1. Компиляция, тестирование (с помощью Unit-тестов) и сборку кода в JAR-файл с помощью Maven.
   1. В артефактах должен быть собранный Jar с вашим проектом.
2. Скачивание файла с данными из внешнего хранилища. Создание и наполнение БД.
   1. При необходимости сохранять БД между job’ами (см. сложный сценарий), кешируйте базу, а не исходные файлы.
3. Выполнение запросов.
   1. Каждый запрос должен быть в отдельной job’е.
   2. В артефактах job’ы должны быть (где необходимо) Excel-отчёты с результатами или картинки с диаграммами.

# Выбираем СУБД

Работа с базой в рамках CI возможна по двум сценариям.

### Обычный сценарий - база H2

H2 - это СУБД, которая позволяет работать с in-memory базами данных. Такая база создаётся при запуске проекта и умирает вместе с ним. Очень удобно для отладки и учебных задач поскольку не требует дополнительной инфраструктуры.

### Средний сценарий - база SQLite (или Postgres, MySQL)

SQLite - это очень легковесная СУБД (поддерживает не все типы данных, оч. бедная поддержка транзакций, ...). Основная особенность этой СУБД по отношению к ДЗ - вся база хранится в 1 файле! Т.е. чтоб хранить состояние между пунктами 2 и 3 достаточно закешировать файл с базой данных.

[Докер-контейнер](https://hub.docker.com/r/velkerr/oracle-hse) с установленной СУБД Oracle Express 11g.

### Ещё более сложный сценарий (+5 баллов)

Более сложный сценарий заключается в использовании более продвинутой СУБД, у которой состояние базы не хранится в 1 файле. Для передачи состояния такой базы между job’ами придется запаковать базу в контейнер и использовать dind. Если при использовании dind вы сталкиваетесь с такой ошибкой CI-job, обратите внимание на это [issue](https://gitlab.com/gitlab-org/gitlab-runner/-/issues/4501) и используйте dind более старой версии:

| services:  - docker:18.09-dind |
| --- |

## Критерии оценивания

| **Задание** | **Задание А** | | **Задание B** | | | | | | | | **Задание C** | | **Задание D** | | | **ВСЕГО** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Подзадача | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 1 | 2 | 3 |
| Балл (обычный сценарий) | 2 | 2 | 1 | 1 | 1,5 | 1,5 | 1,5 | 1,5 | 2,5 | 2 | 3,5 | 3,5 | 1 | 2,5 | 3 | 30 |
| Балл (Средний сценарий) | 3,5 | 3,5 | 1 | 1 | 1,5 | 1,5 | 1,5 | 1,5 | 2,5 | 2 | 3,5 | 3,5 | 1 | 4,5 | 3 | 35 |

1. Комментарий будет полезен когда вы более глубоко погрузитесь в ДЗ. В процессе генерации таблиц с помощью POI нужно будет писать много boiler-plate кода, вручную создавая ячейки для каждого поля в DTO-классе. Чтоб этого избежать удобно использовать Java Reflection API. С помощью Reflection можно получить все названия полей и их значения в текущем DTO-объекте и таким образом превратить заполнение ячеек в цикл. [↑](#footnote-ref-0)