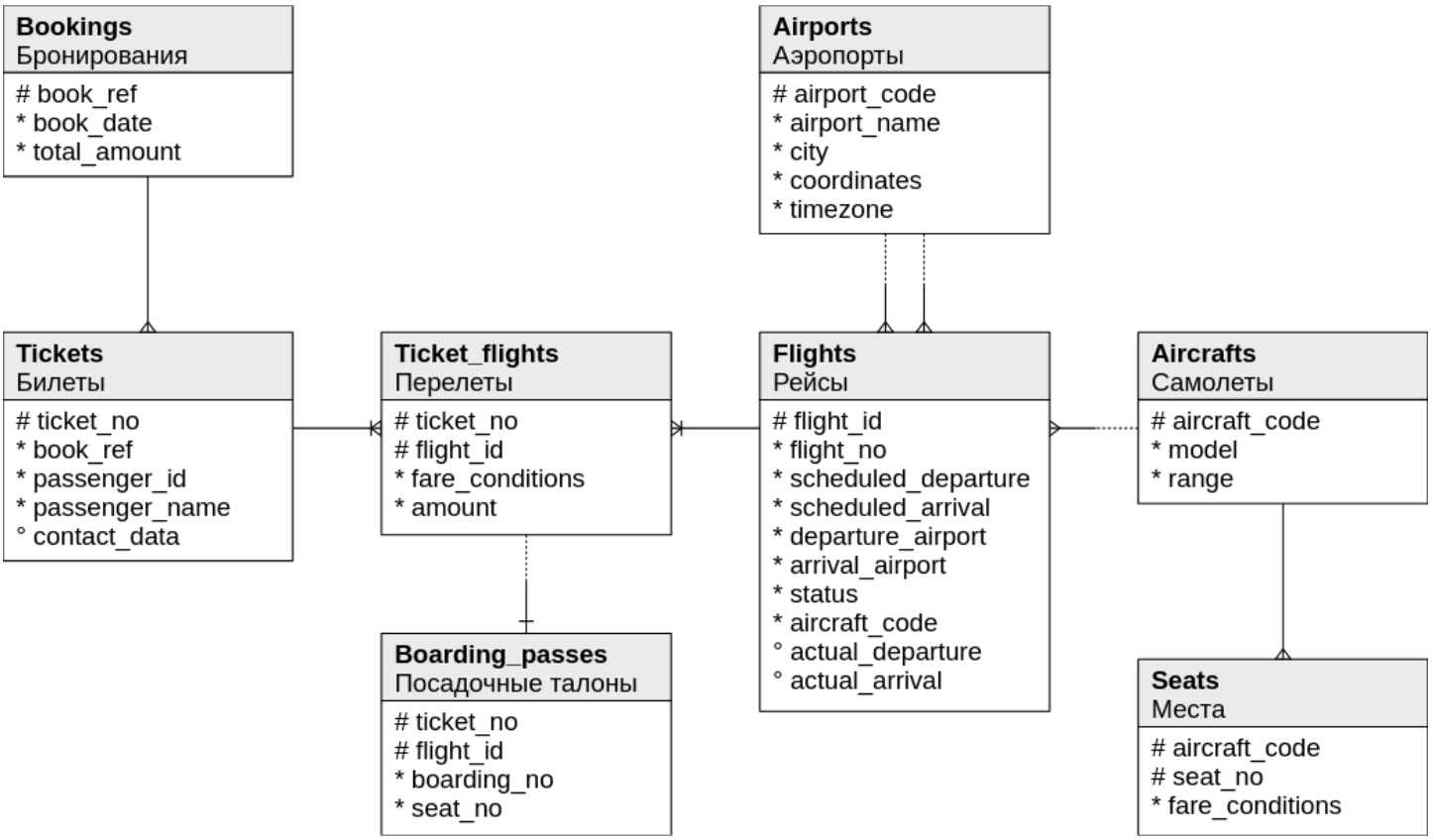
Домашнее задание №3. Базы данных

- Полезный пример 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. Например для таблицы “aircrafts” будет такая ссылка: https://storage.yandexcloud.net/airtrans-small/**aircrafts**.csv

Вся база одним архивом.

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:

<table>
<tr><td>1</td><td>Вывести города, в которых несколько аэропортов.

<table><tr><td>Город</td><td>Список аэропортов</td></tr><tr><td>Москва</td><td>SVO, DME...</td></tr></table></td><td></td></tr>
<tr><td>2</td><td>Вывести города, из которых чаще всего отменяли рейсы.

<table><tr><td>Город</td><td>Кол-во отмененных рейсов</td></tr><tr><td>Москва</td><td>5</td></tr><tr><td>Ульяновск</td><td>4</td></tr></table></td><td></td></tr>
<tr><td>3</td><td>В каждом городе вылета найти самый короткий маршрут. Отсортировать по продолжительности.

<table><tr><td>Город</td><td>Пункт прибытия</td><td>Средняя продолжительность полёта</td></tr><tr><td></td><td></td><td></td></tr></table>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.</td><td></td></tr>
<tr><td>4</td><td>Найти кол-во отмен рейсов по месяцам.

<table><tr><td>Месяц</td><td>Кол-во отмен</td></tr><tr><td></td><td></td></tr></table>Для получения месяца использовать поле scheduled_departure в таблице flights.</td><td>x</td></tr>
<tr><td>5</td><td>Выведите кол-во рейсов в Москву и из Москвы по дням недели за весь наблюдаемый период.
Для задачи C 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.</td><td>x</td></tr>
<tr><td>6</td><td>Отменить все рейсы самолета, заданной модели (модель - параметр). Все билеты, относящиеся к удаленным рейсам - удалить.</td><td></td></tr>
</table>

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

Построение графиков

Для запросов, помеченных “x” 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. Для передачи файлов (например., *.jar) между job’ами используйте cache. 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.
   a. В артефактах должен быть собранный Jar с вашим проектом.
2. Скачивание файла с данными из внешнего хранилища. Создание и наполнение БД.
   a. При необходимости сохранять БД между job’ами (см. сложный сценарий), кешируйте базу, а не исходные файлы.
3. Выполнение запросов.

[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.

a. Каждый запрос должен быть в отдельной job'е.
b. 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 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 и используйте dind более старой версии:

```
services:
  - docker:18.09-dind
```

Критерии оценивания

Задание	Задание A		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