МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных для авиакомпании с применением технологии Oracle Advanced Analytics»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.И. Лешук

(подпись, дата)

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.И. Комарова

(должность, уч. звание) (подпись, дата)

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.И. Комарова

подпись дата инициалы и фамилия

**Содержание**

[Введение 5](#_Toc135306319)

[1. Постановка задачи 6](#_Toc135306320)

[2. Проектирование базы данных 7](#_Toc135306321)

[2.1 Схема базы данных 7](#_Toc135306322)

[2.2 Таблицы базы данных 8](#_Toc135306323)

[2.3 Вывод по разделу 12](#_Toc135306324)

[3. Разработка объектов базы данных 13](#_Toc135306325)

[3.1 Табличное пространство 13](#_Toc135306326)

[3.2 Профили безопасности, роли, пользователи 13](#_Toc135306327)

[3.3 Таблицы 13](#_Toc135306328)

[3.4 Триггеры 14](#_Toc135306329)

[3.5 Хранимые процедуры 15](#_Toc135306330)

[3.5.1 Процедуры пользователя 15](#_Toc135306331)

[3.5.2 Процедуры менеджера 16](#_Toc135306332)

[3.6 Вывод по разделу 18](#_Toc135306333)

[4. Описание процедур экспорта и импорта 19](#_Toc135306334)

[4.1 Описание процедур экспорта 19](#_Toc135306335)

[4.2 Описание процедур импорта 20](#_Toc135306336)

[4.3 Вывод по разделу 20](#_Toc135306337)

[5. Тестирование производительности 21](#_Toc135306338)

[5.1 Оптимизирование запросов 21](#_Toc135306339)

[5.2 Проверка выполнения хранимых процедур 23](#_Toc135306340)

[5.3 Вывод по разделу 24](#_Toc135306341)

[6. Описание и применение технологии Oracle Advanced Analytics в базе данных 25](#_Toc135306342)

[6.1 Описание взаимодействия с технологией 25](#_Toc135306343)

[6.2 Вывод по разделу 29](#_Toc135306344)

[7. Краткое описание приложения для демонстрации 30](#_Toc135306345)

[8. Руководство пользователя 33](#_Toc135306346)

[Заключение 35](#_Toc135306347)

[Список используемых источников 36](#_Toc135306348)

[Приложение А Создание таблиц 37](#_Toc135306349)

[Приложение Б Создание триггеров 39](#_Toc135306350)

[Приложение В Создание профилей безопасности, пользователей, ролей 41](#_Toc135306351)

[Приложение Г Создание процедур 42](#_Toc135306352)

# **Введение**

Базы данных являются ключевым компонентом современных информационных систем, используемых в различных областях жизни. Они позволяют эффективно хранить, организовывать и обрабатывать данные любого масштаба и сложности. Одной из самых мощных и широко используемых СУБД является Oracle Database, которая является промышленным стандартом для управления данными в крупных организациях и предприятиях. Она обладает высокой производительностью, масштабируемостью и надежностью, а также множеством функций и возможностей для обработки и анализа данных.

Использование дополнительных технологий, например таких, как Oracle Advanced Analytics, позволяет проводить анализ данных, выявлять скрытые закономерности, прогнозировать спрос на перелеты, оптимизировать маршруты и ценообразование.

В сфере авиаперевозок осуществляется непрерывное взаимодействие с большим количеством данных. Одним из ключевых элементов управления информацией в авиакомпании является база данных. Она хранит информацию о рейсах, пассажирах, билетах, бортовых журналах, персонале а также других важных аспектах авиаперевозок. Её корректные проектирование и внедрение упрощает взаимодействие с системой как управляющим лицам, так и пользователям. С другой же стороны, малейшие недочёты или ошибки могут привести к серьёзным сбоям, нарушению или медленной работе системы.  
 Учитывая вышесказанное, становится очевидно, что разработка базы данных авиакомпании является довольно тяжёлым и трудоёмким процессом, который требует детального рассмотрения и значительных знаний в данной сфере.

Целью данного курсового проекта является разработка реляционной базы данных для авиакомпании, которая предназначена для хранения информации о рейсах, клиентах и билетах, а также обеспечения быстрого доступа к хранимой информации. Для удобства использования будет разработано соответствующее приложение, которое позволит клиентам быстро и легко получать необходимую информацию о рейсах, а также осуществлять бронирование билетов.

# **Постановка задачи**

В рамках курсового проекта необходимо спроектировать инфраструктуру базы данных авиакомпании. Для этого необходимо проанализировать сферу, и определить, какими должны быть следующие объекты и их содержание в нашей базе данных – табличные пространства и их файлы, таблицы и связи между ними, а также ограничения целостности для них, профили безопасности, пользователи, триггеры, хранимые процедуры, функции, индексы. Далее необходимо разработать эти объекты в базе данных СУБД Oracle 12c посредством написания скриптов на языке SQL.  
 Необходимо заполнить таблицы данными посредством импорта из XML файлов.

Применить технологию Oracle Advanced Analytics для анализа данных в базе данных авиакомпании и проанализировать преимущества (полезные функции), которые предоставляет данная технология.

Также необходимо разработать приложение для удобной демонстрации функционала и возможностей, которые предоставляет проект.

Обязательно должна присутствовать реализация следующих функций: управление авиакомпанией ( управление рейсами, проверка наличия свободных мест на рейс, добавление новых узлов(пунктов назначения)), операции над билетами (бронирование, возврат), анализ работы предприятия (отчёты по прибыли, количеству перелетов и т.д.). Данные операции должны быть реализованы с помощью хранимых процедур.

Необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости.

# **Проектирование базы данных**

* 1. Схема базы данных

Для начала создадим отдельную PDB для этого курсового проекта c помощью утилиты Database Configuration Assistant.

Далее определим, какие нам нужны таблицы. Вообще, в базе данных авиакомпании могут насчитываться десятки таблиц, если это крупная компания и она оперирует большим количеством данных, предоставляет широкий спектр услуг(пассажирская перевозка, грузовая перевозка, перевозка багажа, предоставление бронирования через компаний-партнёров и т.д). Однако, мы ограничимся минимальным необходимым числом таблиц и включим туда какой-нибудь сервис (в данном случае, будет перевозка багажа). Предварительная диаграмма таблиц и их отношения представлены на рисунке 2.1.

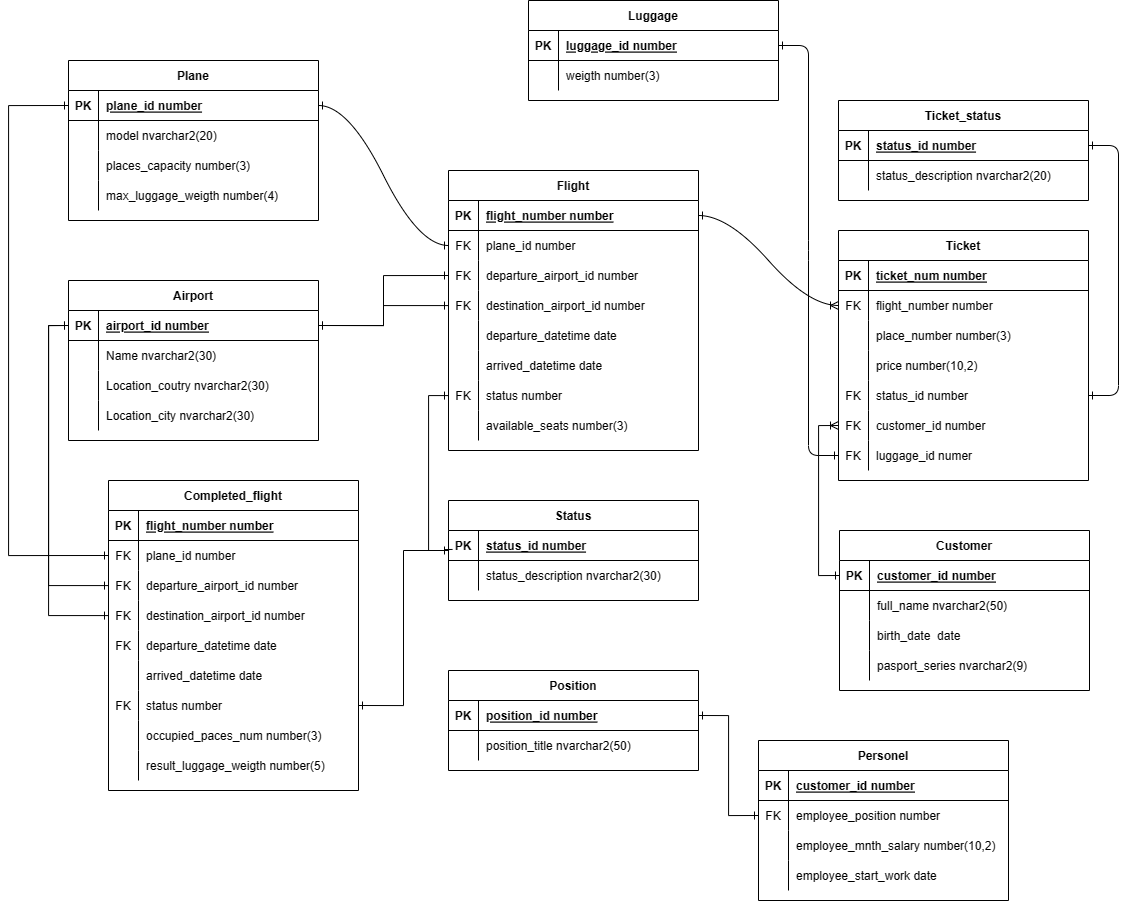


Рисунок 2.1 – Схема базы данных

Также в этих таблицах указаны предварительные типы данных для столбцов, которые могут измениться при необходимости.

Далее определим, как будет происходить взаимодействие с системой. Наглядно это продемонстрирует Use-case диаграмма – рисунок 2.2. Как мы видим из диаграммы, нам понадобится две группы пользователей – менеджеры (которые выполняют управление системой), а также обычные пользователи. Регулировать количество сессий будем с помощью настройки профилей безопасности. Все действия данных пользователей будут выполняться через хранимые процедуры.

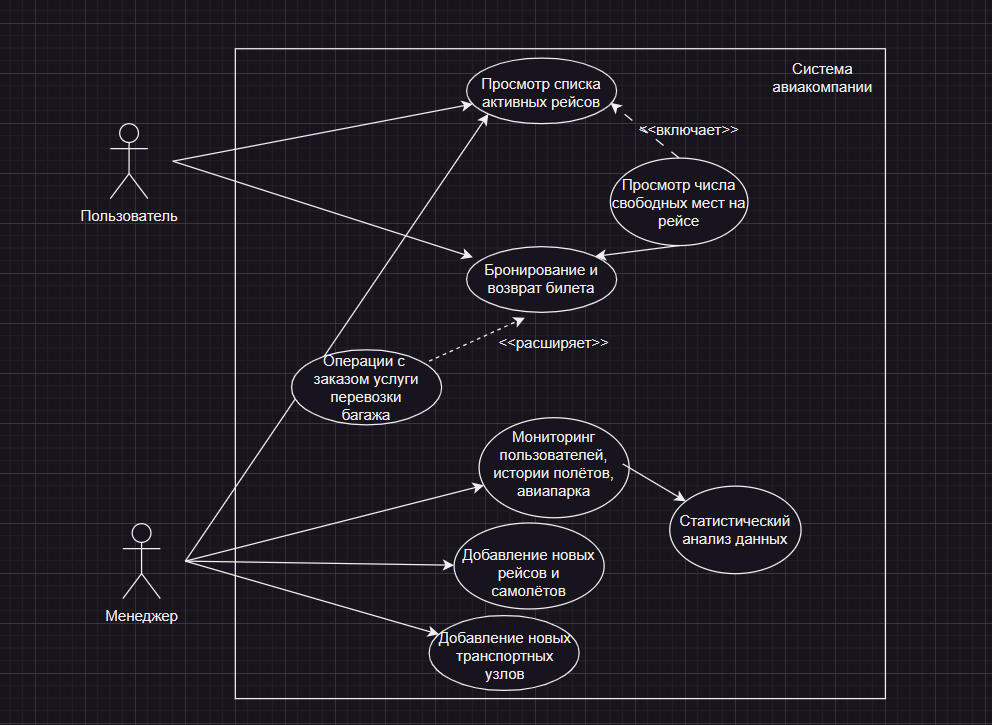


Рисунок 2.2 – Use Case диаграмма

* 1. Таблицы базы данных

Теперь опишем таблицы базы данных: названия, названия столбцов, типы данных столбцов, описание содержания столбцов. Описание представлено в таблицах 2.1 – 2.11.

Таблица 2.1 – Содержание таблицы Airport (Аэропорт)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| airport\_id | NUMBER | Уникальный идентификатор аэропорта |
| airport\_name | NVARCHAR2(60) | Название аэропорта |
| location\_country | VARCHAR2(40) | Название страны расположения |
| location\_city | VARCHAR2(40) | Название города расположения |

Эта таблица хранит в себе данные об аэропортах.

Таблица 2.2 – Содержание таблицы Plane (Самолёт)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| plane\_id | NUMBER | Уникальный идентификатор самолёта |
| plane\_model | NUMBER | Модель самолёта |
| places\_capacity | DATE | Количество посадочных мест |
| max\_luggage\_weigth | NUMBER | Максимальный вес багажа для перевозки |

В этой таблице содержатся данные о самолётах, используемых компанией для авиаперевозок.

Таблица 2.3 – Содержание таблицы Position (Должность)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| position\_id | NUMBER | Уникальный идентификатор должности |
| position\_title | NVARCHAR2(50) | Название должности |

Эта таблица хранит в себе данные о должностях, которые могут занимать сотрудники в компании.

Таблица 2.4 – Содержание таблицы Personel (Персонал)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| employee\_id | NUMBER | Уникальный идентификатор сотрудника |
| employee\_fname | NVARCHAR2(50) | ФИО сотрудника |
| employee\_position | NUMBER | Идентификатор должности |
| employee\_mnth\_salary | NUMBER (10,2) | Месячная зарплата сотрудника |
| employee\_start\_work | DATE | Дата начала работы сотрудника |

В данной таблице содержатся данные о персонале, работающих в данный момент в компании.

Таблица 2.5 – Содержание таблицы Status(Статус полёта)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| status\_id | NUMBER | Уникальный идентификатор статуса полёта |
| status\_description | NVARCHAR2(30) | Описание статуса полёта |

Эта таблица хранит в себе варианты статусов полёта, которые могут быть установлены для рейса.

Следующая таблица 2.6 содержит описание таблицы базы данных Flight, в которой будут храниться данные о рейсах.

Таблица 2.6 – Содержание таблицы Flight (Рейс)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| flight\_number | NUMBER | Уникальный идентификатор рейса |
| plane\_id | NUMBER | Уникальный идентификатор самолёта |
| departure\_airport\_id | NUMBER | Уникальный идентификатор самолёта отправления |
| destination\_airport\_id | NUMBER | Уникальный идентификатор самолёта прибытия |
| departure\_datetime | DATE | Время отправления |
| arrive\_datetime | DATE | Время прибытия |
| available\_seats | NUMBER(3) | Текущее количество свободных мест |
| status | NUMBER | Уникальный идентификатор статуса |

Таблица 2.7 – Содержание таблицы Completed\_flights (Завершённые рейс)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| flight\_number | NUMBER | Уникальный идентификатор рейса |
| plane\_id | NUMBER | Уникальный идентификатор самолёта |
| departure\_airport\_id | NUMBER | Уникальный идентификатор самолёта отправления |
| destination\_airport\_id | NUMBER | Уникальный идентификатор самолёта прибытия |
| departure\_datetime | DATE | Время отправления |
| arrive\_datetime | DATE | Время прибытия |
| occupied\_places\_num | NUMBER(3) | Количество занятых мест |
| result\_luggage\_weigth | NUMBER(4) | Результирующий вес багажа |
| status | NUMBER | Уникальный идентификатор статуса |

В данной таблице содержатся данные о завершённых рейсах.

Таблица 2.8 – Содержание таблицы Luggage (Багаж)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| luggage\_id | NUMBER | Уникальный идентификатор багажа |
| weigth | NUMBER(3) | Вес багажа |

Эта таблица хранит в себе строки с данными о весе багажа и соответствующего ему идентификатора. Данные из этой таблице найдут применение в таблице Ticket.

Таблица 2.9 – Содержание таблицы Ticket (Билет)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ticket\_num | NUMBER | Уникальный идентификатор билета |
| flight\_number | NUMBER | Уникальный идентификатор рейса |
| place\_number | NUMBER (3) | Номер места |
| price | NUMBER (7,2) | Стоимость билета |
| status\_num | NUMBER | Уникальный идентификатор статуса |
| customer\_id | NUMBER | Уникальный идентификатор клиента |
| luggage\_id | NUMBER | Уникальный идентификатор багажа |

Эта таблица хранит в себе данные о билетах, заказанных клиентами.

Таблица 2.10 – Содержание таблицы Ticket\_status(Статус билета)

|  |  |  |
| --- | --- | --- |
| Таблица PK | Таблица FK | Описание связи |
| Status\_num | Policies | Уникальный идентификатор статуса |
| status\_description | Policies | Описание статуса |

Предпоследняя таблица содержит идентификатор и соответствующий ему статус, который может применяться к статусу билета.

Таблица 2.11 – Содержание таблицы Customer (Клиенты)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| customer\_id | NUMBER | Уникальный идентификатор клиента |
| full\_name | NVARCHAR2(100) | ФИО клиента |
| date\_of\_birth | DATE | Дата рождения клиента |
| passport\_series | NVARCHAR2(10) | Серия паспорта клиента |

Последняя таблица содержит данные о клиентах.

Основные таблицы, которые обеспечат минимальную функциональность – Flight, Plane, Airport, Customer, Ticket, Personel. Таблицы Position, Status, Ticket\_status – для сохранения нормализации. Таблицы Luggage и Completed\_Flight– для дополнительного функционала. Первая – для реализации системы перевозки багажа. Вторая включает в себя уже завершённые рейсы (история рейсов, которые были исполнены компанией). Идея в следующем: когда рейс завершается (в таблице Flight ячейка Status изменяется на «Completed», «Cancelled»), то с помощью триггера, строка с данными этого рейса переносится в Completed\_Flight, и удаляется из таблицы запланированных рейсов. Пользователь при поиске нужных ему рейсов будет вызывать процедуру, в которой хранится select-запрос с условием where для таблицы Flight. Из неё же можно посмотреть, если ли свободные места на данный рейс. Столбец Available\_Seats будет автоматически калькулироваться: от общего числа мест будет отниматься кол-во проданных билетов на этот рейс (опять же, это обеспечит триггер).

Далее рассмотрим систему билетов и багажа. Когда пользователь покупает билет на рейс, соответствующая строка с данными добавляется в таблицу Ticket. Если пользователь выбирает перевозить багаж, то создаётся строка в таблице Luggage, а если нет, то в ячейке luggage\_id таблицы Ticket устанавливается значение NULL.Каждый самолёт имеет свой максимальный возможный вес багажа для перевозки. За подсчёт текущего веса багажа будет также отвечать триггер. Когда рейс завершается, статус соответствующих билетов изменяется на «Used», когда происходит возврат билета или отмена рейса – на «Refunded».

* 1. Вывод по разделу

В данном разделе мы начали работу над созданием базы данных авиакомпании: создана подключаемая база данных, спроектированы таблицы базы данных, столбцы и типы данных для них, определена модель взаимодействия с системой, а также каким образом будет реализован необходимый функционал при помощи объектов базы данных.

# **Разработка объектов базы данных**

* 1. **Табличное пространство**

После создания подключаемой базы данных для нашего проекта и до создания таблиц и заполнения её данными нам необходимо создать табличное пространство – листинг 3.1

create tablespace MININGG2

datafile 'C:\BD\mining2.dbf'

size 20 m

autoextend on next 10 m

maxsize 200M

extent management local;

commit;

Листинг 3.1 – Создание табличного пространства

После создания пользователей необходимо сделать это табличное пространство для них табличным пространством по умолчанию и выделить квоту.

* 1. **Профили безопасности, роли, пользователи**

Как уже описывалось выше, у нас будет 2 пользователя – клиент и менеджер. Отличаться у них будут профили безопасности и параметрами: число сессий для пользователя, «время жизни» пароля, количество неверных попыток ввода пароля, время сессии и простоя. У них будут также свои роли, отличающиеся привилегиями – пользователь сможет только выполнять хранимые процедуры, определенные для него и создавать подключение, менеджер же – просматривать все таблицы, создавать подключение, выполнять хранимые процедуры на добавление/удаление данных из таблиц. На данном этапе выдадим этим пользователям лишь привилегию на создание подключения. Код на создание объектов из данного пункта представлен в приложении В.

* 1. **Таблицы**

Перейдём к написанию скриптов для создания таблиц. Из общего можно отметить, что в каждой таблице есть поле, представляющее ID, которое является первичным ключом и заполняется с помощью Identity с шагом 1. Все числовые типы данных представлены типом number, в таблице Flight присутствует тип date для реализации времени отправления и прибытия, а также даты рождения клиентов и даты найма сотрудников. Символьные данные представлены типом nvarchar2. Почти на всех столбцах стоит ограничение целостности «not null» кроме тех, где это предусмотрено. На столбце с серий паспорта таблицы Customer стоит ограничение целостности «unique».

После создания таблиц они были заполнены небольшим количеством тестовых данных для проверки работоспособности и разработки объектов следующих пунктов. Скрипт для создания всех таблиц расположен в приложении А.

* 1. **Триггеры**

Для данной реализации у нас будет 4 триггера, полный скрипт для создания которых представлен в приложении Б. Первый – BEFORE INSERT на таблицу Ticket для обновления ячейки available\_seats для конкретных рейсов таблицы Flight, второй – AFTER UPDATE на случай отмены билета, который также будет регулировать количество свободных мест на рейс(при отмене билета повышает кол-во свободных мест). Третий – BEFORE INSERT на таблицу Ticket, он будет проверять, не превышает ли текущий вес багажа максимальное возможное значение. И последний AFTER UPDATE триггер на таблицу Flight, который при изменении статуса полёта в «Завершённый» перемещает эту строку в таблицу Completed\_flights. Рассмотрим детально первый триггер – Листинг 3.2.

CREATE OR REPLACE TRIGGER update\_available\_seats

BEFORE INSERT ON Ticket

FOR EACH ROW

DECLARE

v\_available\_seats number;

BEGIN

SELECT available\_seats

INTO v\_available\_seats

FROM Flight

WHERE flight\_number = :NEW.flight\_number;

IF v\_available\_seats <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'No available seats on this flight.');

END IF;

UPDATE Flight

SET available\_seats = (SELECT places\_capacity FROM Plane WHERE plane\_id =

(SELECT plane\_id FROM Flight WHERE flight\_number = :NEW.flight\_number)) -

(SELECT COUNT(\*) FROM Ticket WHERE flight\_number = :NEW.flight\_number and status\_num = 100) - 1

WHERE flight\_number = :NEW.flight\_number; END;

Листинг 3.2 – Update\_available\_seats триггер

Данный триггер будет срабатывать до операции INSERT в таблицу. Мы проверяем текущее количество свободных мест, если оно меньше либо равно 0 (страховка от выхода за пределы), то «выбрасываем» исключение. Если же количество свободных мест попадает в допустимые пределы, то производим обновление данной ячейки путём следующего вычисления: От общего числа мест в самолёте отнимаем число забронированных билетов (со статусом 100) на данный рейс. И так как это before триггер, нам нужно отнять дополнительную единицу.

* 1. Хранимые процедуры
     1. Процедуры пользователя

Для пользователя будет реализовано 4 процедуры: поиск авиабилетов, бронирование билета, возврат билета, просмотр всех билетов пользователя (билетная история). Рассмотрим детально процедуру поиска авиабилетов – Листинг 3.3.

CREATE OR REPLACE PROCEDURE SEARCH\_FLIGHTS(

departure\_city nvarchar2, destination\_city nvarchar2,

dep\_date nvarchar2,

p\_resflights OUT SYS\_REFCURSOR

)

AS

BEGIN

open p\_resflights for

select f.flight\_number, plane.plane\_model, a1.location\_city as "DEPARTURE\_CITY", a1.airport\_name AS "DEPARTURE\_AIRPORT" ,

a2.location\_city as "DESTINATION\_CITY", a2.airport\_name as "DESTINATION\_AIRPORT",

TO\_CHAR(f.departure\_datetime, 'DD.MM.YYYY HH24:MI:SS') AS departure\_datetime,

TO\_CHAR(f.arrive\_datetime, 'DD.MM.YYYY HH24:MI:SS') AS arrive\_datetime,

f.available\_seats, s.status\_description

from Flight f join Plane on plane.plane\_id = f.plane\_id

join Airport a1 on f.departure\_airport\_id = a1.airport\_id

join Airport a2 on f.destination\_airport\_id = a2.airport\_id

join Status s on f.status = s.status\_id

where a1.location\_city = departure\_city

and a2.location\_city = destination\_city

and TRUNC(f.departure\_datetime) = TO\_DATE(dep\_date, 'DD.MM.YYYY')

order by departure\_datetime asc; END;

Листинг 3.3 – Процедура для поиска авиабилетов

У нас есть 3 входных параметра: город отправления, город назначения, дата вылета. А также один выходной параметр типа SYS\_REFCUROSR. Необходимость в его использовании заключена в том, что в Oracle 12c нельзя просто обернуть SELECT-запрос в процедуру – данные обязательно должны куда-то записываться. И как раз те рейсы, что будут найдены с помощью запроса с условием в основной части процедуры будут возвращены как SYS\_REFCURSOR. Пример вызова процедуры представлен на рисунке 3.1.

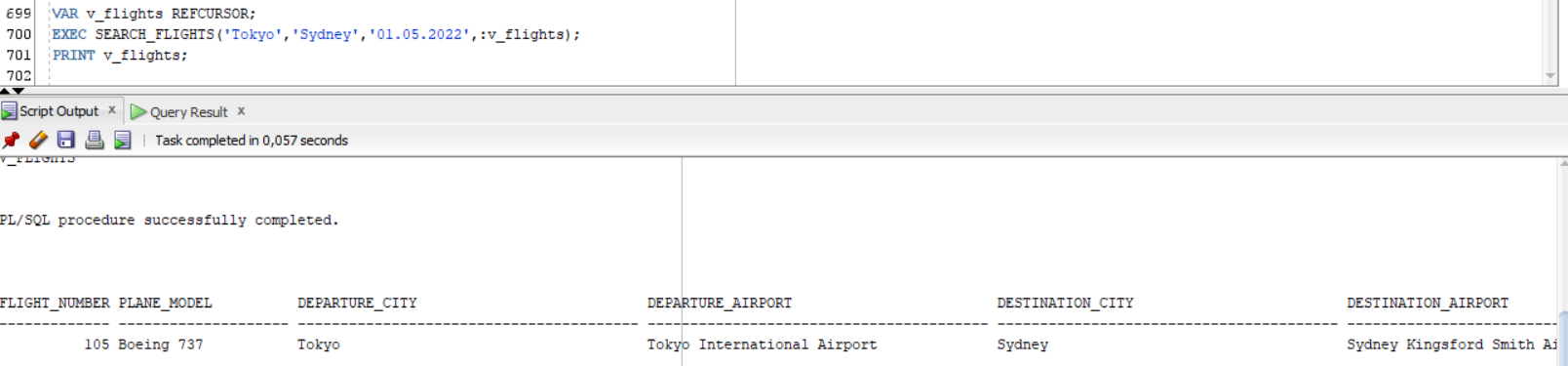


Рисунок 3.1 – Результат выполнения процедуры

* + 1. Процедуры менеджера

Для менеджера определим следующие процедуры: отмена рейса, проверка рейсов(осуществляется проход по всем строкам и те рейсы, у которых дата прибытия уже прошла, то есть меньше текущей, устанавливаются со статусом «Completed»), добавление нового самолёта, добавление и увольнение персонала, включение нового аэропорта, добавление новых рейсов в предстоящие рейсы. Отдельно рассмотрим процедуру добавления рейса – Листинг 3.4.

create or replace procedure ADD\_FLIGHT(

p\_plane\_id number, p\_dep\_airport\_id number,p\_dest\_airport\_id number,

p\_dep\_datetime date, p\_dest\_datetime date)

as

av\_seats NUMBER;

BEGIN

select places\_capacity into av\_seats from plane

where plane\_id = p\_plane\_id;

insert into Flight(plane\_id, departure\_airport\_id, destination\_airport\_id, departure\_datetime, arrive\_datetime, available\_seats,status)

values(p\_plane\_id, p\_dep\_airport\_id, p\_dest\_airport\_id, p\_dep\_datetime, p\_dest\_datetime, av\_seats, 1);

commit; END;

Листинг 3.4 – Процедура добавления рейса

В качестве входных параметров мы принимаем id самолёта, аэропортов вылета и назначения, дату и время вылета и прибытия. Также определяем дополнительную переменную для нахождения числа мест в данном самолёте. Вызов процедуры представлен на рисунке 3.2.

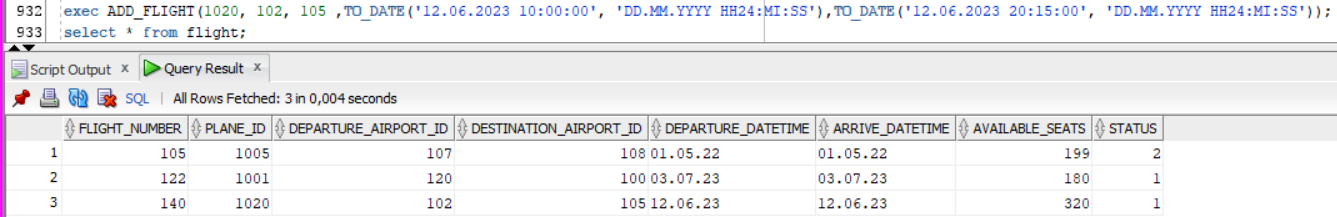


Рисунок 3.2 – Добавление нового рейса

Как мы видим, таблица Flight включает в себя большое количество внешних ключей. И чтобы производить операцию вставки в данную таблицу с помощью данной хранимой процедуры менеджер должен знать, каким строкам соответствуют ID, используемые им в данной хранимой процедуре. Можно сделать вывод, что появляется необходимость выдать привилегию на SELECT менеджеру из данных таблиц. Это же касается и таблиц, в которые менеджер будет проводить операцию вставки. Следовательно, надо разработать процедуры, которые будут возвращать результаты select-запросов.

Для тестирования производительности нам нужно заполнить какую-нибудь таблицу 100000 строками. Пусть это будет таблица Flight. Несмотря на то, что мы описали процедуру импорта данных, нам нужно сделать какой-то автоматический процесс. Создадим следующую хранимую процедуру: мы будем проходить по каждому аэропорту и создавать для него рейс с каждым другим аэропортом(за исключением тех, что находятся в одном городе), дату и время будем делать случайную в пределах года(можно выбрать интервал меньше), самолёт также выбираем случайно. Код представлен в листинге 3.5.

CREATE OR REPLACE PROCEDURE insert\_flights AS

v\_plane\_id NUMBER; v\_dep\_airport\_id NUMBER; v\_dest\_airport\_id NUMBER;

v\_dep\_datetime DATE; v\_arr\_datetime DATE; v\_available\_seats NUMBER;

v\_status\_id NUMBER := 1;

BEGIN

FOR dep\_airport IN (SELECT airport\_id, location\_city FROM airport) LOOP

FOR dest\_airport IN (SELECT airport\_id, location\_city FROM airport) LOOP

IF dep\_airport.location\_city != dest\_airport.location\_city THEN

SELECT plane\_id INTO v\_plane\_id FROM plane ORDER BY DBMS\_RANDOM.VALUE FETCH FIRST 1 ROWS ONLY;

v\_dep\_airport\_id := dep\_airport.airport\_id;

v\_dest\_airport\_id := dest\_airport.airport\_id;

v\_dep\_datetime := TO\_DATE('01.01.2025 00:00:00', 'DD.MM.YYYY HH24:MI:SS') + DBMS\_RANDOM.VALUE(0, 365);

v\_arr\_datetime := v\_dep\_datetime + NUMTODSINTERVAL(DBMS\_RANDOM.VALUE(180, 960), 'MINUTE');

SELECT places\_capacity into v\_available\_seats FROM plane WHERE plane\_id = v\_plane\_id;

INSERT INTO flight(plane\_id, departure\_airport\_id, destination\_airport\_id, departure\_datetime, arrive\_datetime, available\_seats, status)

VALUES(v\_plane\_id, v\_dep\_airport\_id, v\_dest\_airport\_id, TO\_DATE(v\_dep\_datetime, 'DD.MM.YYYY HH24:MI:SS'),

TO\_DATE(v\_arr\_datetime, 'DD.MM.YYYY HH24:MI:SS'), v\_available\_seats, v\_status\_id);

END IF; END LOOP; END LOOP; COMMIT; END;

Листинг 3.5 – Процедура для заполнения таблицы Flight

Стоит отметить ещё несколько особенностей: менеджер может просматривать билетную историю конкретного пользователя (выполнение соответствующей хранимой процедуры), просматривать предстоящие рейсы, как и пользователь. Также имеется как-бы «общая» хранимая процедура для проекта, которая будет добавлять пользователя в таблицу Customer.

Полный скрипт для создания вышеописанных хранимых процедур можно найти в приложении Г.

* 1. Вывод по разделу

В данном разделе было описано и выполнено создание объектов для базы данных авиакомпании, таких как табличное пространство, профили безопасности, роли и пользователи, таблицы, триггеры, хранимые процедуры. Также было определено взаимодействие всех этих элементов между собой.

# **Описание процедур экспорта и импорта**

* 1. Описание процедур экспорта

База данных авиакомпании может включать в себя большое количество информации. Здесь и возникает вопрос о том, каким образом можно эффективно экспортировать и импортировать данные. Для хранения данных был выбран XML-формат. Теперь предстоит разработать соответствующие хранимые процедуры для импорта/экспорта данных. Разберём структуру процедур на примере таблицы Airport – листинг 4.1. Предварительно создаём объект Directory, где прописываем путь к папке для хранения данных.

Создаём переменную типа CLOB (для хранения большого объёма данных) и с помощью DBMS\_XMLGEN.GETXML преобразуем все данные из таблицы к типу CLOB. Далее производим запись в предварительно определенную директорию, файл «Airport.xml» Процедуры экспорта для других таблиц также будут иметь аналогичную структуру.

CREATE OR REPLACE PROCEDURE EXPORT\_AIRPORT\_XML

AS

v\_clob CLOB;

v\_file UTL\_FILE.FILE\_TYPE;

BEGIN

SELECT DBMS\_XMLGEN.GETXML('SELECT \* FROM AIRPORT') INTO v\_clob FROM DUAL;

v\_file := UTL\_FILE.FOPEN('DATA\_DIR2', 'AIRPORT.xml', 'w');

UTL\_FILE.PUT(v\_file, v\_clob);

UTL\_FILE.FCLOSE(v\_file);

END;

Листинг 4.1 – Процедура для экспорта данных из таблицы Airport

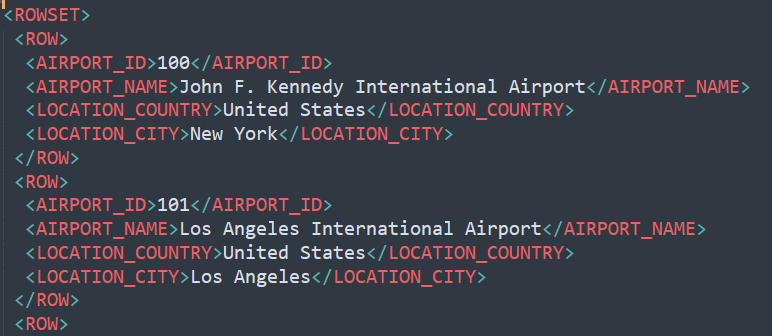


Рисунок 4.1 – Структура экспортированных данных

* 1. Описание процедур импорта

В процедуре импорта мы построчно считываем данные из xml-файла с помощью функции UTL\_FILE.GET\_LINE. При считывании тега «<Airport>»(одна строка данных) устанавливаем в NULL значение переменных, в которых будут хранится данные. Далее ищем соответствие тегов и берём данные из тега, которые записываем в переменную. При нахождении закрывающего тега «</Airport>» производим вставку в таблицу. Выполняем данные операции в цикле, пока не встретим конец файла. Полностью процедура для импорта описана в приложении Г под названием IMPORT\_AIRPORT\_XML.

* 1. Вывод по разделу

В этом разделе была рассмотрена проблема заполнения таблиц данными и экспорта данных из таблиц, а также предложено её решение с помощью хранимых процедур, реализующих экспорт и импорт данных в/из XML файл(-а).

# **Тестирование производительности**

* 1. Оптимизирование запросов

У нас уже созданы необходимые объекты базы данных, а также таблицы заполнены необходимым количеством данных – рисунок 5.1. Теперь необходимо протестировать, что всё работает, как нужно и как задумано.

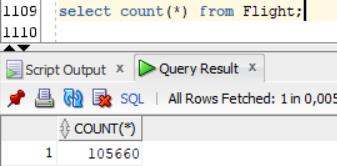


Рисунок 5.1 – Количество строк в таблицe Flight

Для начала проверим, какое количество времени занимает выполнение нашей хранимой процедуры для поиска билетов, в которой соде – Рисунок 5.2.

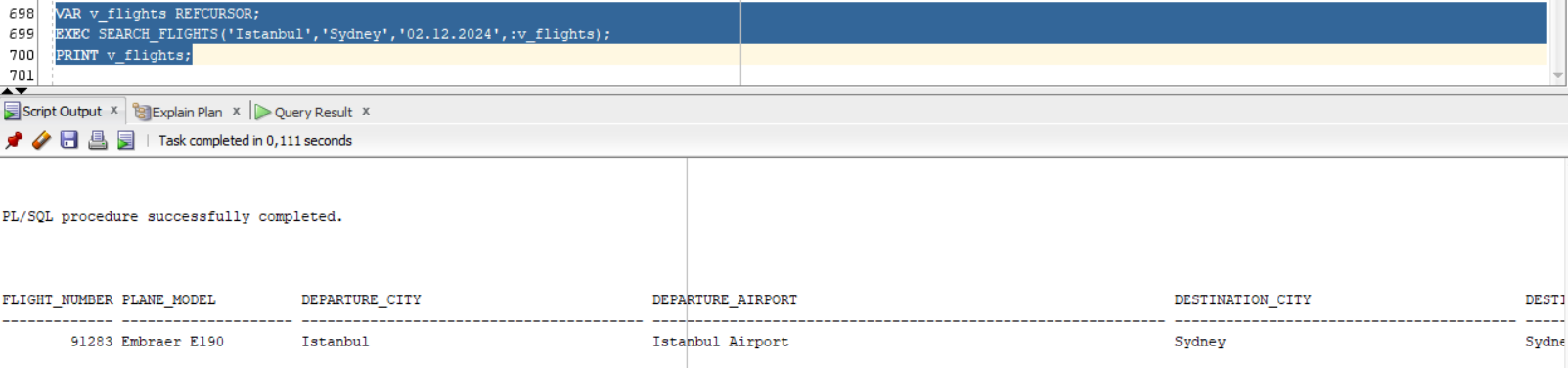


Рисунок 5.2 – Время поиска билета

Как мы можем видеть, время ожидания составило 0,111 секунды. Результат приемлемый. Естественно, в базах данных крупных авиакомпаний хранятся записи о миллионах рейсов, и поиск выполняется очень долго. Мы лишь смоделировали данную ситуацию. Теперь простым select-запросом найдём все рейсы из одного города в другой – рисунок 5.3. Теперь просмотрим план выполнения данного запроса – рисунок 5.4.

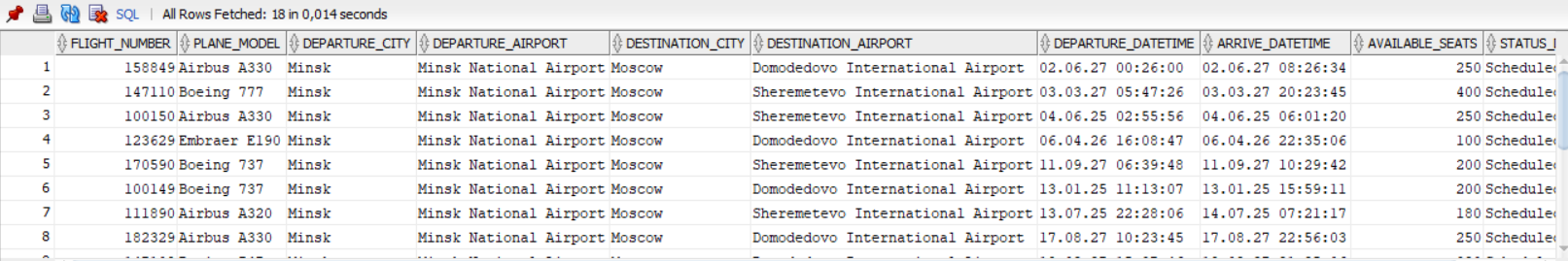


Рисунок 5.3 – Результат SELECT-запроса

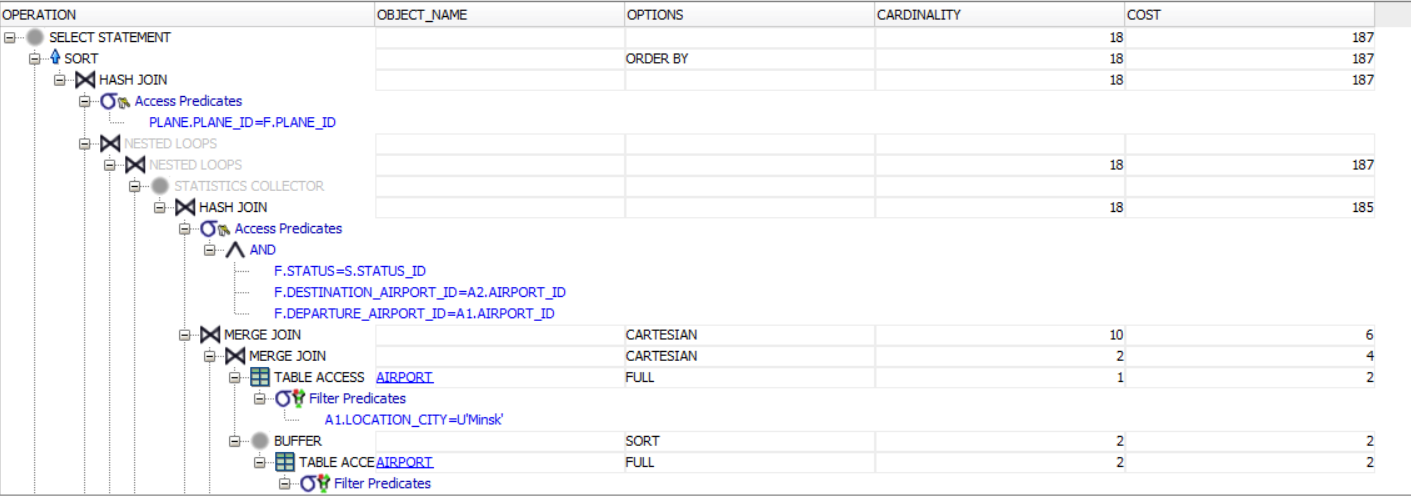


Рисунок 5.4 – План запроса

Теперь попробуем оптимизировать данный запрос с помощью индекса – Листинг 5.1.

Create index idx\_double\_date on Flight(departure\_airport\_id, destination\_airport\_id);

Листинг 5.1 – Создание индекса

Создание такого индекса обуславливаем тем, что при выборке у нас есть условие по двум этим столбцам, также в запросе присутствует секция order by.

Теперь проверим выполнение поиска рейсов – рисунок 5.5.

Как мы видим, ситуация изменилась: время запроса сократилось почти в 10 раз, стоимость выполнения уменьшилась. Следовательно делаем вывод, что создание данного индекса в текущих обстоятельствах является необходимым.

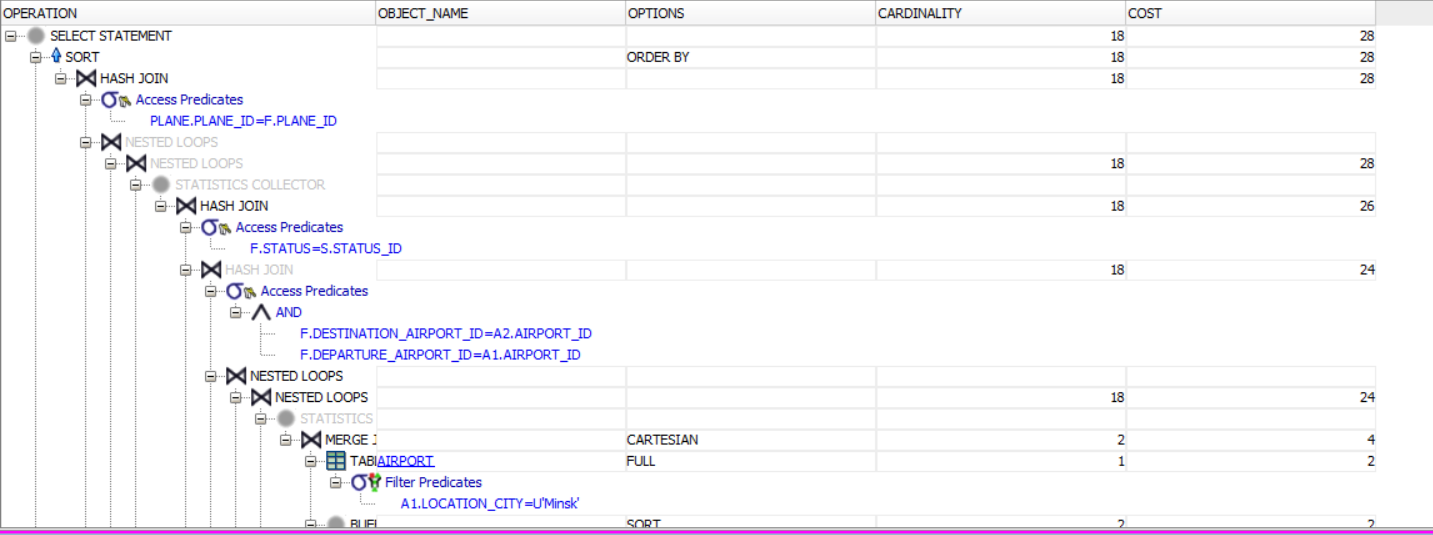


Рисунок 5.5 – Ситуация после создания индекса

* 1. Проверка выполнения хранимых процедур

Теперь протестируем основные хранимые процедуры заказа билета и багажа, отмены билета, отмены рейса.

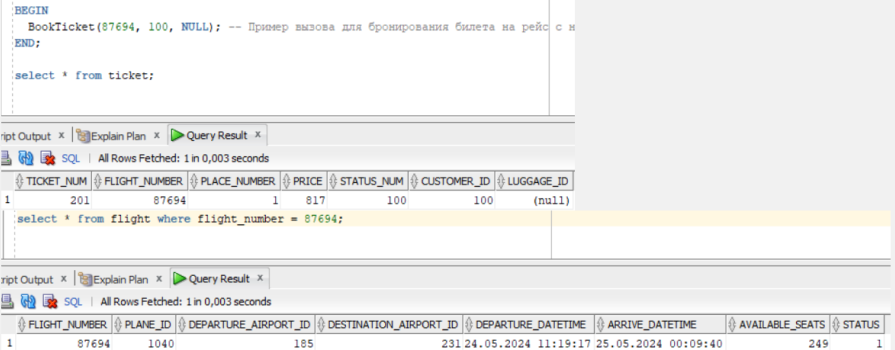


Рисунок 5.6 – Бронирование билета

После вставки ещё одного билета и отмены его с помощью процедуры CANCEL\_TICKET – результат на рисунке 5.7.

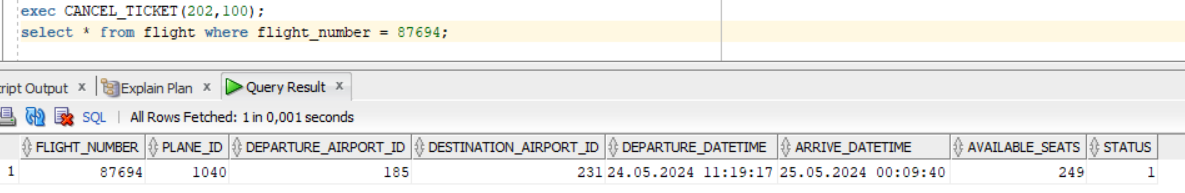


Рисунок 5.7 – Отмена билета

Теперь попробуем отменить рейс – рисунок 5.8.

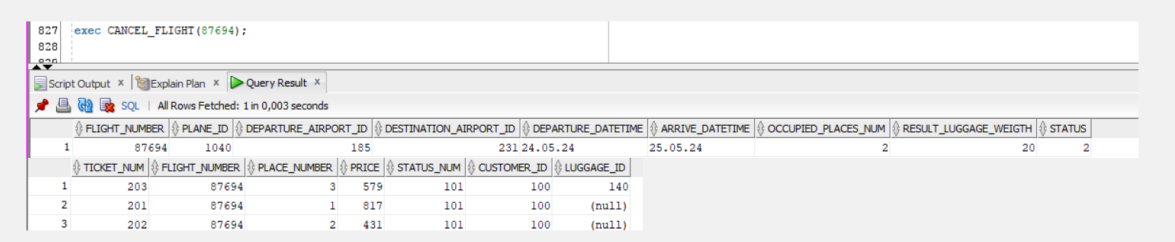


Рисунок 5.8 – Отмена рейса

Как мы видим, отменённый рейс перенёсся в таблицу Completed\_flight, все билеты на него отменились. Демонстрация билетной истории – на рисунке 5.9.

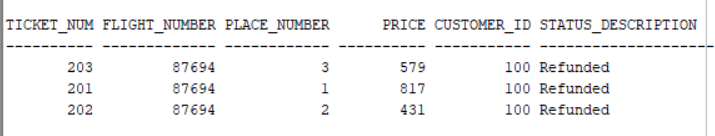


Рисунок 5.9 – Билетная история пользователя

Все операции вставки строк в таблицы за менеджера выполняются без проблем, процедуры выполняются быстро.

* 1. Вывод по разделу

В данном разделе был оптимизирован один из основных запросов, использующийся в процедуре поиска рейсов, с помощью индекса, а также проведено успешное тестирование выполнения хранимых процедур, реализующие основной функционал.

# **Описание и применение технологии Oracle Advanced Analytics в базе данных**

* 1. Описание взаимодействия с технологией

Oracle Analytics является дополнительным приложением для визуализации данных. Благодаря графическому дизайну мы можем выбирать только те данные, которые непосредственно пригодятся нам для визуализации без написания запросов и далее перейти к построению различных графиков и диаграмм, что помогает в анализировании отдельно взятых аспектах предприятия. Многие элементы реализованы по принципу Drag-and-drop. Диапазон реализации у данного программного продукта очень широкий, по этому рассмотрим лишь некоторые типовые задачи, которые эта технология поможет решить, описывая взаимодействие с данным приложением.

К счастью, все элементы локализированы под русский язык, а также со старта имеется тестовый проект, на котором можно потренироваться, но мы перейдём к созданию своего. Сразу после создания проекта необходимо создать набор данных – те таблицы и столбцы, по которые мы хотим визуализировать. Для этого нам необходимо выполнить подключение к своей базе данных через следующее окно – рисунок 6.1.

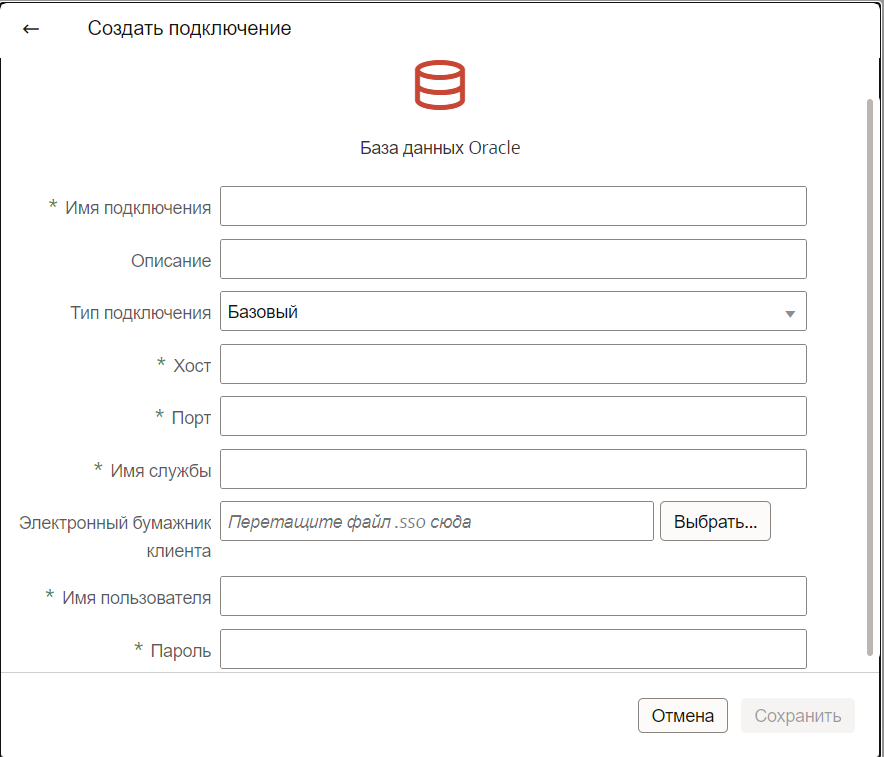


Рисунок 6.1 – Окно подключения

После подключения в оверлее слева выбираем необходимые нам таблицы из необходимой схемы – рисунок 6.2. После этого они переносятся в рабочую область. Если они связаны отношением первичного и внешнего ключа, эта связь графически отображается

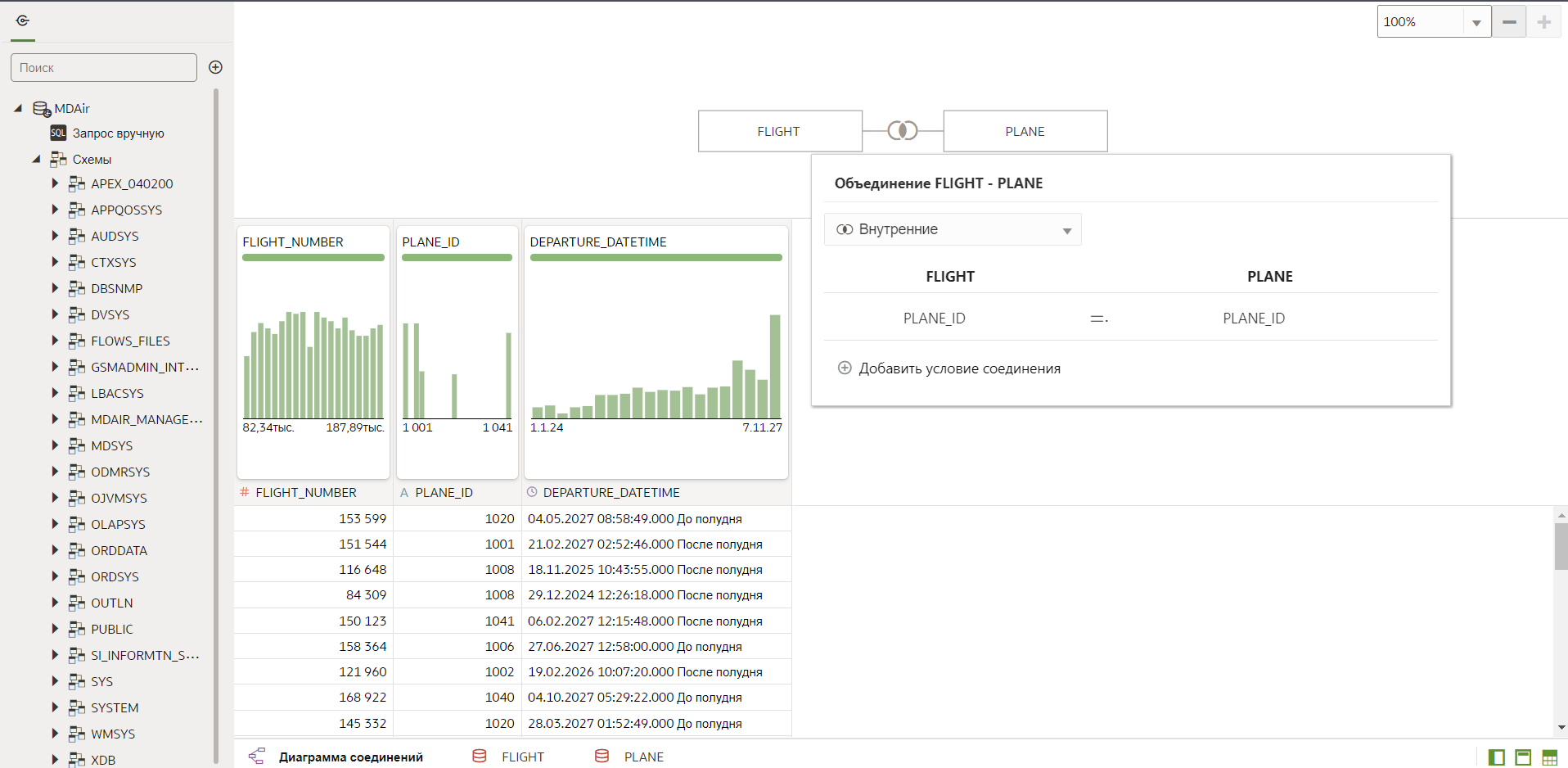


Рисунок 6.2 – Начало работы над набором данных

Как мы можем видеть, уже на данном этапе мы видим диаграммы значений каждого отдельного столбца.

Поставим себе задачу – найти количество рейсов, которые запланированы для каждой модели самолётов на определенный промежуток времени(например, на 2025 год). Ранее уже было описано, что мы выбираем необходимые для нас данные. Для нашей задачи не имеет значения вместительность самолёта или максимальный вес багажа, который может быть перевезён. Оставляем из таблицы Flight только 3 столбца, представленные на рисунке. Из таблицы Plane – Plane\_id и Model. В Oracle Analytics есть одна особенность: все столбцы можно поделить на две группы – атрибуты и показатели. Столбцы-атрибуты могут выступать только как атрибуты (значения по оси X), столбцы-показатели – значения по оси Y. Так как нам нужно как-то группировать количество рейсов, мы сделаем следующую настройку – рисунок 6.3. Устанавливаем данный столбец как показатель, включаем агрегирование уникальных значений. На этом создание набора данных для нашей задачи завершено. Перейдём к визуализации.

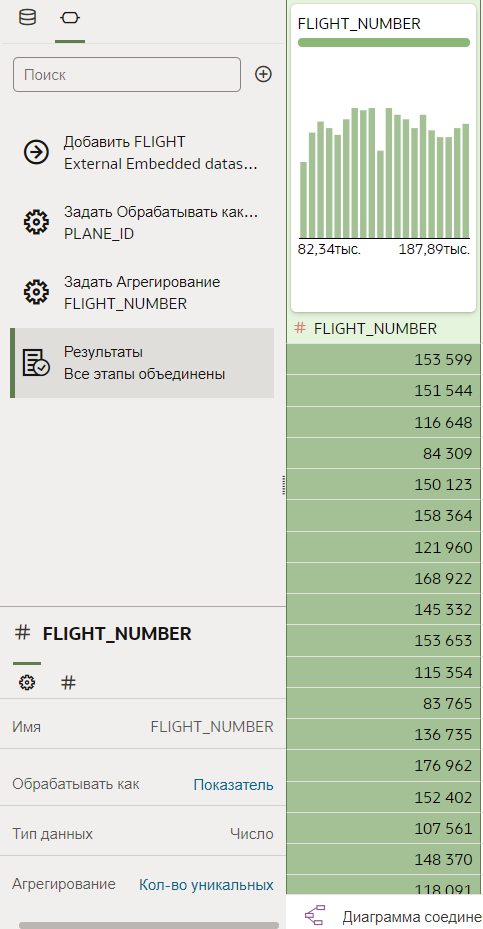


Рисунок 6.3 – Настройка столбца FLIGHT\_NUMBER

С помощью перетягивания необходимых столбцов в соответствующие области для настройки диаграммы, мы получаем следующий результат – рисунок 6.4. Здесь же мы можем и проанализировать нашу хранимую процедуру для заполнения таблицы Flight – несмотря на выборку случайной модели самолёта на рейс их общее число получилось практически схожее. Можем дополнить эту диаграмму дополнительными параметрами – установить параметр, который будет определять ширину столбца, цвет столбца для данного типа диаграммы.

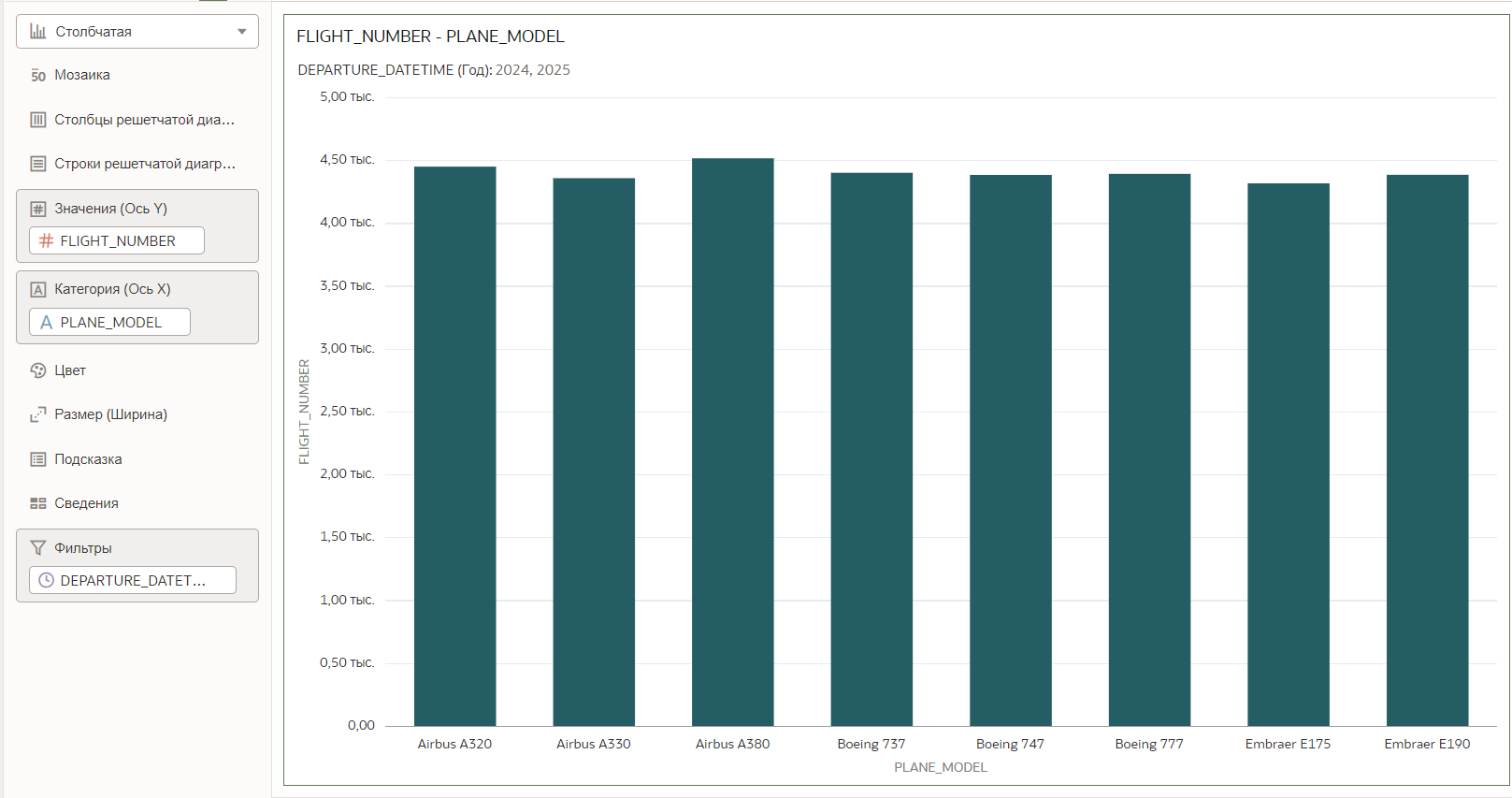


Рисунок 6.4 – Готовая диаграмма

Теперь поставим другую задачу – определим страны и города, в которые запланировано больше всего перелётов на лето 2025 года. Результат для стран – на рисунке 6.5. При наведении на сектор показывается детальная информация (кол-во рейсов, процент, страна).

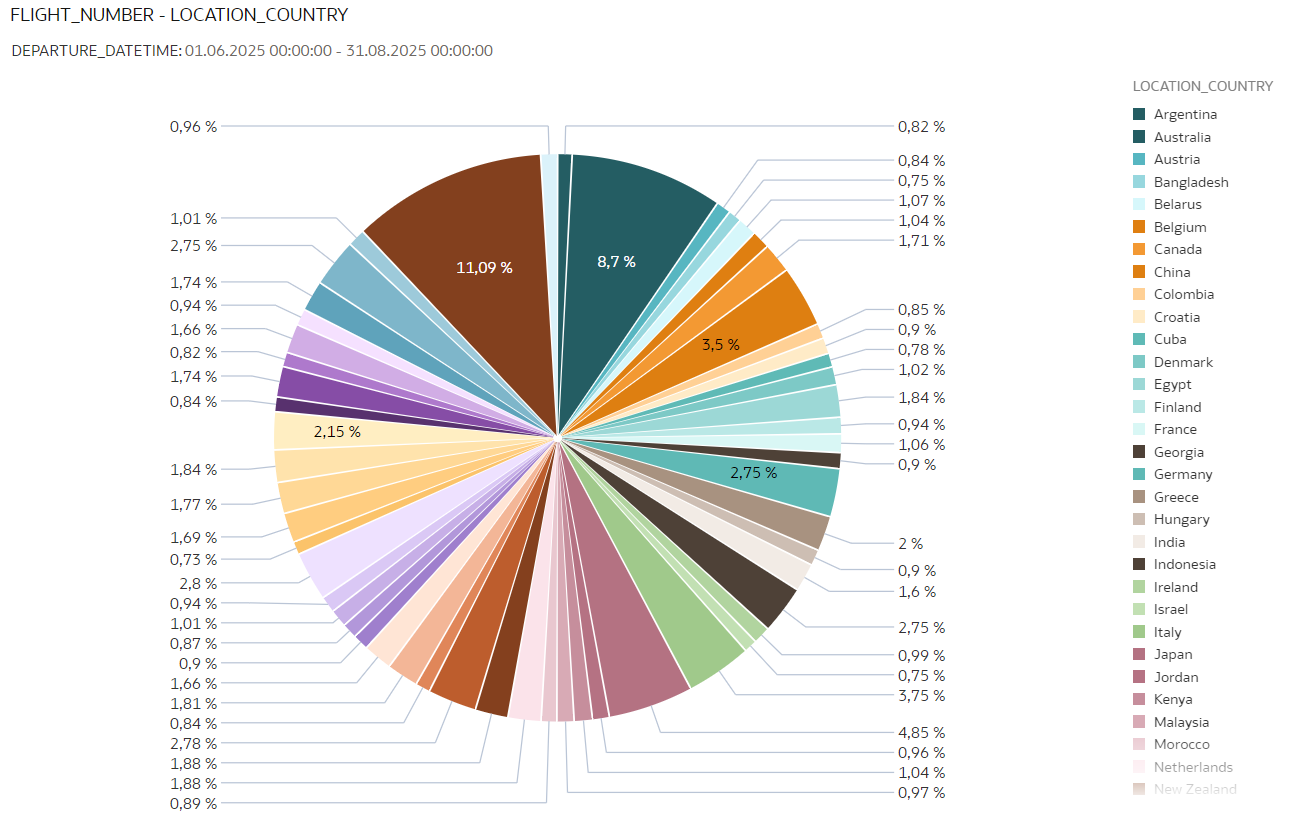


Рисунок 6.5 – Диаграмма запланированных рейсов по странам на лето 2025

А вот в таком соотношении расположились топ-10 городов, в которых запланировано больше всего рейсов на лето 2025 – рисунок 6.6.

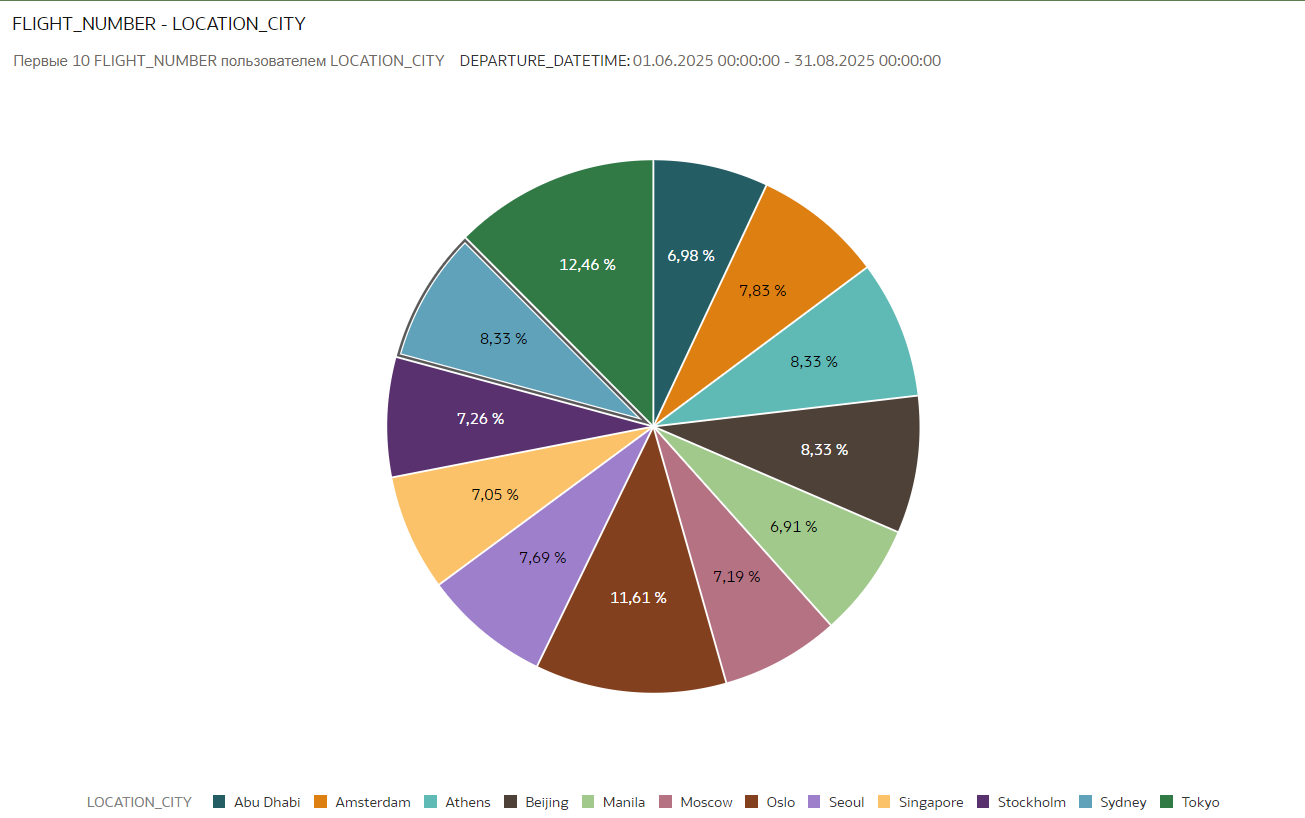
\*

Рисунок 6.6 – Диаграмма запланированных рейсов для городов

* 1. Вывод по разделу

В заключении по данной технологии хочется сказать, что Oracle Analytics – это мощный инструмент, который позволяет анализировать данные и создавать визуализации с помощью интуитивно понятного графического интерфейса. Данной технологии можно найти широкое применение в бизнес-анализе компаний любого масштаба.

# **Краткое описание приложения для демонстрации**

В приложении будет 2 окна. В первом, которое показано на рисунке 7.1. пользователь может провести регистрацию(добавление новой строки в таблицу Cutomer), авторизоваться(на данной странице нету разницы, происходят действия от пользователя, менеджера или гостя), а также выполнить поиск рейсов с интересующими его параметрами. После нажатия на кнопку поиск параметр «Место отправления», «Место назначения», а также   
«Дата» будут переданы второму окну, где будет выполняться процедура поиска необходимого рейса с данными параметрами – рисунок 7.2.

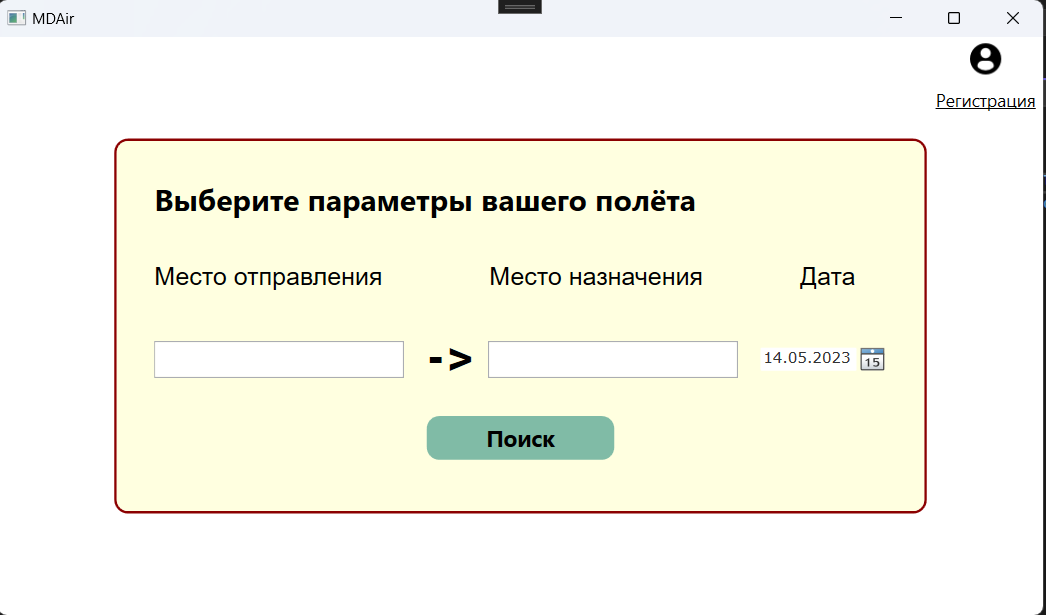


Рисунок 7.1 – Первое окно приложения

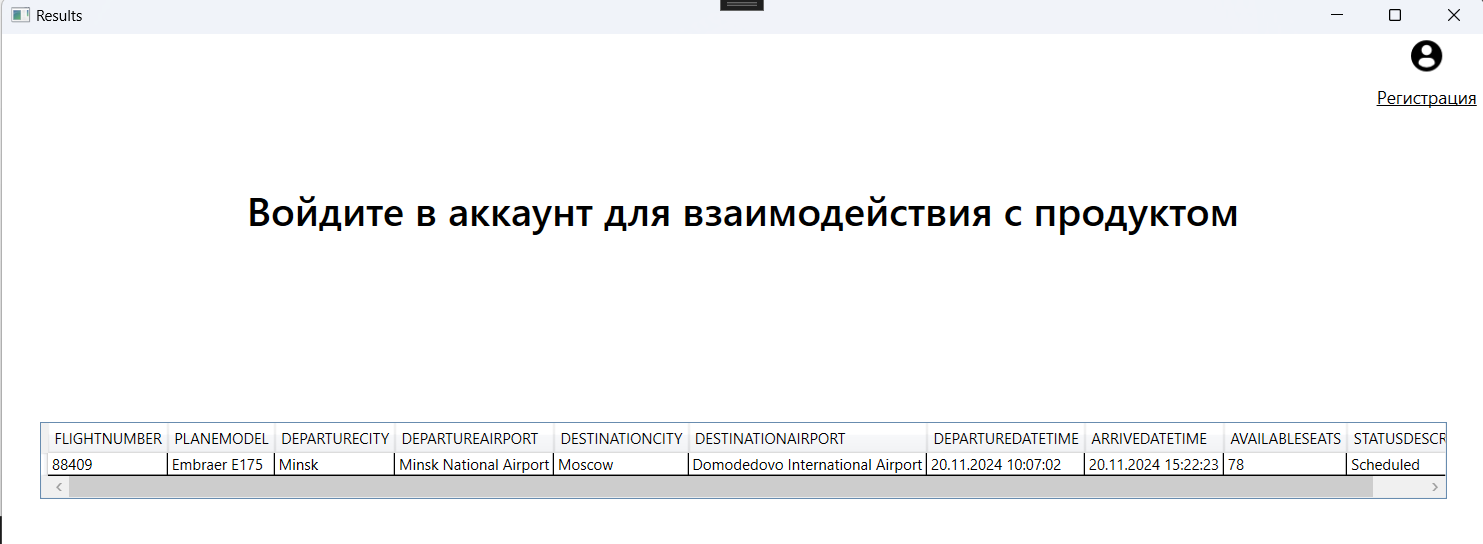


Рисунок 7.2– Поиск рейса без авторизации

Далее при нажатии на иконку пользователя в правом верхнем углу мы можем перейти к авторизации, При нажатии на надпись «Регистрация» - попадаем в панель авторизации – рисунок 7.3.

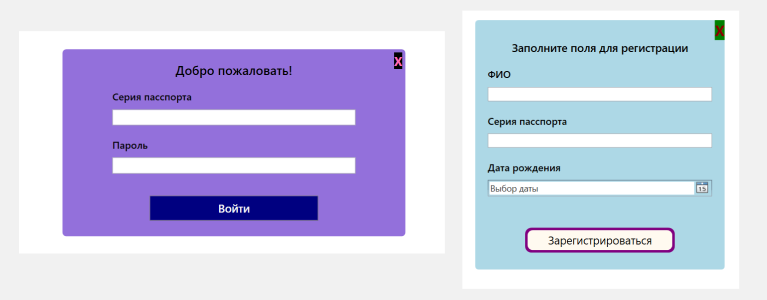


Рисунок 7.3 – Панели авторизации и регистрации

Для входа, как Менеджер мы должны в поле «Серия паспорта» ввести «Manager» и соответствующий пароль MDAIR\_MANAGER. Для обычных пользователей – пароль пользователя MDAIR\_CLIENT. От того, авторизированы мы под пользователем или менеджером появляются 2 элемента управления с различным функционалом – рисунок 7.4.

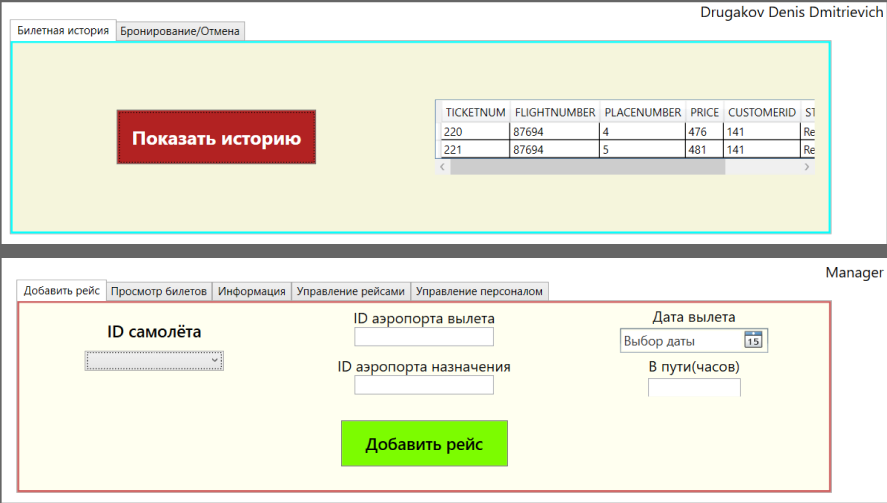


Рисунок 7.4 – Панели управления для менеджера и пользователя

Для пользователя функционал определён полностью – просмотр своей билетной истории, бронирование и отмена билета. Для менеджера же частично – просмотр билетной истории пользователей – рисунок 7.5, просмотр таблиц, управление рейсами и персоналом. Конечно, менеджер имеет гораздо больше возможностей, но оставшиеся возможности однотипны – добавление новых аэропортов, рейсов и т.д. Такой функционал уже представлен на вкладке «Управление персоналом», так что такой реализации будет достаточно для демонстрации возможностей менеджера.

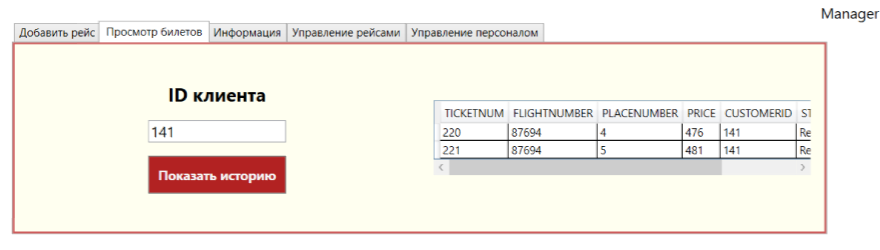


Рисунок 7.5 – Просмотр билетной истории менеджером

Возможностью расширения является добавление новых вкладок на панели управления менеджера, которые будут реализовывать выданные ему привилегии.

# **Руководство пользователя**

При входе в приложение пользователь попадает в первое окно . Здесь же пользователь может зарегистрироваться или войти с помощью интерактивных элементов справа сверху данного окна (что не является обязательным) – рисунок 8.1.

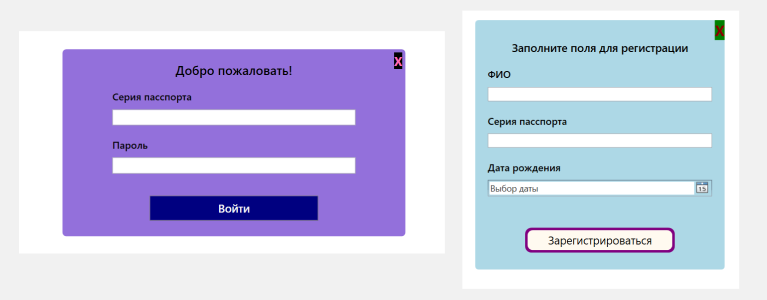


Рисунок 8.1 – Панели авторизации и регистрации.

Далее пользователю необходимо вписать в элементы TextBox необходимые ему города и выбрать дату в DatePicker – рисунок 8.2. Чтобы выполнить поиск рейсов – нажать на кнопку «Поиск».

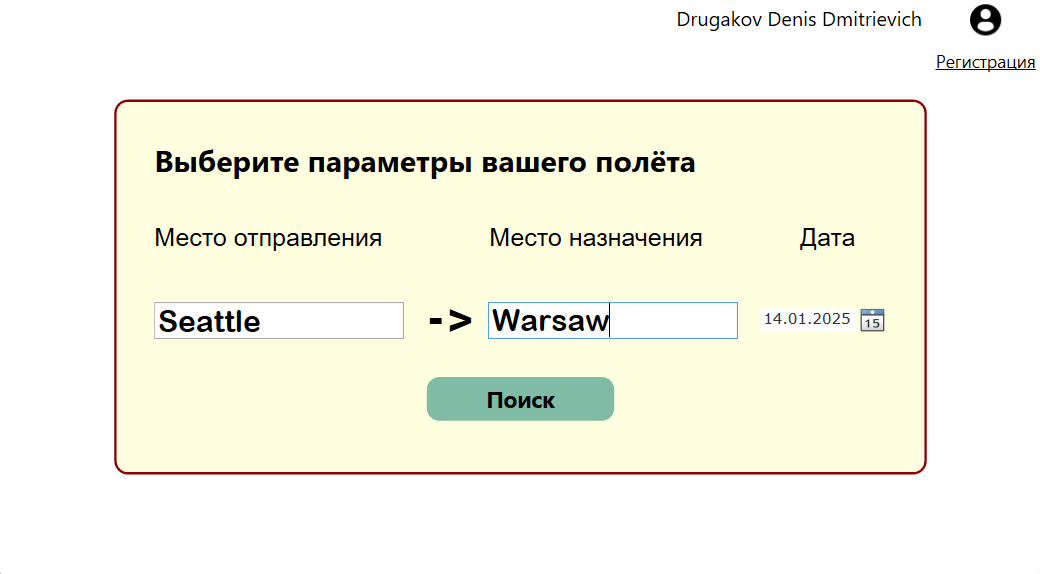


Рисунок 8.2 – Главное окно приложения

После этого у пользователя откроется второе окно с результатами поиска – рисунок 8.3.

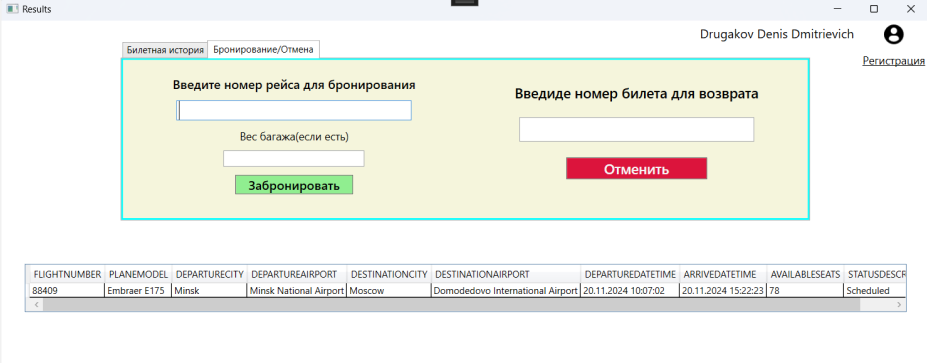


Рисунок 8.3 – Окно с результатами, вход за пользователя

Если на данном этапе пользователь ещё не авторизовался, то он не сможет выполнять бронирование билетов и другие характерные для него операции. Если же пользователь прошёл авторизацию, ему будет доступна панель управления, в которой с помощью интерактивных элементов (кнопки, текстовые области) он сможет выполнить бронирование и отмену билета, а также просмотреть свою билетную историю.

# **Заключение**

В ходе выполнения курсового проекта была спроектирована и реализована инфраструктура базы данных для авиакомпании с использованием СУБД Oracle 12c. Были определены и разработаны необходимые объекты базы данных, такие как табличные пространства, таблицы, ограничения целостности, профили безопасности, пользователи, триггеры, хранимые процедуры. Также был проведен импорт данных из XML файлов для заполнения таблиц.

Важным аспектом проекта было применение технологии Oracle Advanced Analytics для анализа данных в базе данных авиакомпании. Были изучены и применены полезные функции, предоставляемые данной технологией, для обеспечения анализа работы предприятия, количеству перелетов и другой инфографики. Это позволило получить ценные показатели и в перспективе улучшить принятие управленческих решений.

В рамках проекта было разработано приложение, которое демонстрирует функционал и возможности, предоставляемые базой данных авиакомпании. Реализованы функционал управления авиакомпанией, бронирования и возврата билетов, проверки наличия свободных мест, управление персоналом.

Также внимание было уделено производительности базы данных. Проведено тестирование на таблице, содержащей более 100 000 строк, и проведён анализ необходимых изменений в структуре базы данных для обеспечения оптимальной производительности.

В результате успешной реализации курсового проекта была создана база данных авиакомпании с широким функционалом, позволяющим эффективно управлять авиаперевозками, осуществлять операции над билетами и анализировать работу предприятия. Проект демонстрирует преимущества использования СУБД Oracle 12c и технологии Oracle Advanced Analytics для создания мощной и гибкой системы управления авиакомпанией.

# **Список используемых источников**

1. Блинова Е.А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 10.04.2023.
2. Работа с файлами в Oracle, [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/F49540\_01/DOC/server.815/a68001/utl\_file.htm. – Дата доступа: 01.05.2021.
3. Пользовательское руководство по Oracle Analytics. Создание подключения, [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/middleware/bi/analytics-desktop/bidvd/connect-data-sources.html – Дата доступа: 02.05.2023.
4. Набор разработчика Oracle для Visual Studio, [Электронный ресурс]. – Режим доступа:https://marketplace.visualstudio.com/items?itemName=OracleCorporation.OracleDeveloperToolsForVisualStudio2022&ssr=false#overview – Дата доступа: 25.04.2023.
5. Класс ConfigurationManager в C#, [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en‑us/dotnet/api/system.configuration.configurationmanager?view=windowsdesktop-7.0 – Дата доступа: 26.04.2023.

# **Приложение А Создание таблиц**

create table Airport(

airport\_id number GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 1) primary key,

airport\_name nvarchar2(60) not null,

location\_country nvarchar2(40) not null,

location\_city nvarchar2(40) not null);

create table Plane(

plane\_id number GENERATED BY DEFAULT AS IDENTITY (START WITH 1000 INCREMENT BY 1) primary key,

plane\_model nvarchar2(20) not null,

places\_capacity number(3) not null,

max\_luggage\_weigth number(4) not null);

create table Position(

position\_id number generated by default as identity(start with 1 maxvalue 100 increment by 1) primary key,

position\_title nvarchar2(50) not null);

create table Personel(

employee\_id number generated by default as identity(start with 1 increment by 1) primary key,

employee\_fname nvarchar2(50) not null,

employee\_position number not null,

employee\_mnth\_salary number(10,2) not null,

employee\_start\_work date not null,

constraint fk\_personel\_position foreign key (employee\_position)

references Position (position\_id));

create table Status(

status\_id number generated by default as identity(start with 1 increment by 1) primary key,

status\_description nvarchar2(30));

create table Flight(

flight\_number number GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 1) primary key,

plane\_id number references Plane(plane\_id),

departure\_airport\_id number references Airport(airport\_id) not null,

destination\_airport\_id number references Airport(airport\_id) not null,

departure\_datetime date not null,

arrive\_datetime date not null,

available\_seats number(3) not null,

status references Status(status\_id) not null );

create table Completed\_flights(

flight\_number number primary key,

plane\_id number references Plane(plane\_id),

departure\_airport\_id number references Airport(airport\_id) not null,

destination\_airport\_id number references Airport(airport\_id) not null,

departure\_datetime date not null,

arrive\_datetime date not null,

occupied\_places\_num number(3) not null,

result\_luggage\_weigth number(4) not null,

status references Status(status\_id) not null );

create table Luggage(

luggage\_id number GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 1) primary key,

weigth number(3) not null);

create table Ticket(

ticket\_num number GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 1) primary key,

flight\_number references Flight(flight\_number) not null,

place\_number number(3) not null,

price number(7,2) not null,

status\_num references Ticket\_status(status\_num) not null,

customer\_id number references Customer(customer\_id) not null,

luggage\_id number references Luggage(luggage\_id));

create table Ticket\_status(

status\_num number GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 1) primary key,

status\_description nvarchar2(20) not null);

CREATE TABLE Customer (

customer\_id NUMBER GENERATED BY DEFAULT AS IDENTITY(START WITH 100 INCREMENT BY 1) PRIMARY KEY,

full\_name NVARCHAR2(100) NOT NULL,

date\_of\_birth DATE NOT NULL,

passport\_series NVARCHAR2(10) not null unique);

# **Приложение Б Создание триггеров**

CREATE OR REPLACE TRIGGER update\_available\_seats

BEFORE INSERT ON Ticket

FOR EACH ROW

DECLARE

v\_available\_seats number;

BEGIN

SELECT available\_seats

INTO v\_available\_seats

FROM Flight

WHERE flight\_number = :NEW.flight\_number;

IF v\_available\_seats <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'No available seats on this flight.');

END IF;

UPDATE Flight

SET available\_seats = (SELECT places\_capacity FROM Plane WHERE plane\_id =

(SELECT plane\_id FROM Flight WHERE flight\_number = :NEW.flight\_number)) -

(SELECT COUNT(\*) FROM Ticket WHERE flight\_number = :NEW.flight\_number and status\_num = 100) - 1

WHERE flight\_number = :NEW.flight\_number; END;

CREATE OR REPLACE TRIGGER increase\_available\_seats

AFTER UPDATE OF status\_num ON Ticket

FOR EACH ROW

BEGIN

IF :NEW.status\_num = 101 AND :OLD.status\_num != 101 THEN -- добавляем проверку на изменение статуса

UPDATE Flight

SET available\_seats = LEAST(available\_seats + 1,

(SELECT places\_capacity FROM Plane WHERE plane\_id =

(SELECT plane\_id FROM Flight WHERE flight\_number = :NEW.flight\_number)))

WHERE flight\_number = :NEW.flight\_number;

END IF; END;

CREATE OR REPLACE TRIGGER lug\_weigth\_check

BEFORE INSERT ON Ticket

FOR EACH ROW

DECLARE

v\_total\_luggage\_weight NUMBER;

v\_max\_luggage\_weight NUMBER;

v\_plane\_id NUMBER;

BEGIN

SELECT NVL(SUM(l.weigth), 0)

INTO v\_total\_luggage\_weight

FROM Ticket t JOIN Luggage l ON t.luggage\_id = l.luggage\_id

WHERE t.flight\_number = :NEW.flight\_number and t.status\_num <> 101;

SELECT max\_luggage\_weigth, plane\_id

INTO v\_max\_luggage\_weight, v\_plane\_id

FROM Plane

WHERE plane\_id = (SELECT plane\_id FROM Flight WHERE flight\_number = :NEW.flight\_number);

IF v\_total\_luggage\_weight + NVL(:NEW.luggage\_id, 0) > v\_max\_luggage\_weight THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Total luggage weight is exceeding the limit.'); END IF; END;

create or replace trigger Flight\_status\_check

after update of status on Flight

for each row

declare

v\_total\_weight NUMBER;

oc\_seats NUMBER;

begin

SELECT COUNT(\*)

into oc\_seats

FROM Ticket

WHERE flight\_number = :new.flight\_number and status\_num = 100;

if :new.status in (2, 3) then

SELECT SUM(l.weigth)

into v\_total\_weight

FROM Luggage l

JOIN Ticket t ON l.luggage\_id = t.luggage\_id

WHERE t.flight\_number = :new.flight\_number;

insert into Completed\_flights(flight\_number, plane\_id, departure\_airport\_id, destination\_airport\_id,

departure\_datetime, arrive\_datetime, occupied\_places\_num, result\_luggage\_weigth, status)

values (:new.flight\_number, :new.plane\_id, :new.departure\_airport\_id, :new.destination\_airport\_id,

:new.departure\_datetime, :new.arrive\_datetime, oc\_seats, v\_total\_weight, :new.status); end if; end;

# **Приложение В Создание профилей безопасности, пользователей, ролей**

CREATE PROFILE CLIENT\_PROFILE limit

password\_life\_time unlimited

sessions\_per\_user 1000

FAILED\_LOGIN\_ATTEMPTS 5

PASSWORD\_LOCK\_TIME 1

PASSWORD\_REUSE\_TIME 10

PASSWORD\_GRACE\_TIME default

connect\_time 180

idle\_time 35;

create role CLIENT\_ROLE;

create user MDAIR\_CLIENT identified by 7777777

profile CLIENT\_PROFILE

account unlock;

CREATE PROFILE MANAGER\_PROFILE limit

password\_life\_time 90

sessions\_per\_user 3

FAILED\_LOGIN\_ATTEMPTS 3

PASSWORD\_LOCK\_TIME 2

PASSWORD\_REUSE\_TIME 10

PASSWORD\_GRACE\_TIME default

connect\_time 240

idle\_time 45;

create role MANAGER\_ROLE;

create user MDAIR\_MANAGER identified by qazwsx909

PROFILE MANAGER\_PROFILE

account unlock;

# **Приложение Г Создание процедур**

CREATE OR REPLACE PROCEDURE SEARCH\_FLIGHTS(

departure\_city nvarchar2,

destination\_city nvarchar2,

dep\_date date,

p\_resflights OUT SYS\_REFCURSOR

)

AS

BEGIN

open p\_resflights for

select f.flight\_number, plane.plane\_model, a1.location\_city as "DEPARTURE\_CITY", a1.airport\_name AS "DEPARTURE\_AIRPORT" ,

a2.location\_city as "DESTINATION\_CITY", a2.airport\_name as "DESTINATION\_AIRPORT",

TO\_CHAR(f.departure\_datetime, 'DD.MM.YYYY HH24:MI:SS') AS departure\_datetime,

TO\_CHAR(f.arrive\_datetime, 'DD.MM.YYYY HH24:MI:SS') AS arrive\_datetime,

f.available\_seats, s.status\_description

from Flight f join Plane on plane.plane\_id = f.plane\_id

join Airport a1 on f.departure\_airport\_id = a1.airport\_id

join Airport a2 on f.destination\_airport\_id = a2.airport\_id

join Status s on f.status = s.status\_id

where a1.location\_city = departure\_city

and a2.location\_city = destination\_city

and TRUNC(f.departure\_datetime) = dep\_date

order by departure\_datetime asc;

END;

CREATE OR REPLACE PROCEDURE BookTicket (

flight\_num IN NUMBER,

customer\_id IN NUMBER,

luggage\_weight IN NUMBER DEFAULT NULL

)

AS

luggage\_id NUMBER;

place\_num NUMBER;

price NUMBER;

BEGIN

IF luggage\_weight IS NOT NULL THEN

INSERT INTO Luggage(weigth)

VALUES (luggage\_weight);

SELECT MAX(luggage\_id) INTO luggage\_id FROM Luggage;

END IF;

SELECT COALESCE(MAX(place\_number), 0) + 1

INTO place\_num

FROM Ticket

WHERE flight\_number = flight\_num;

price := ROUND(DBMS\_RANDOM.VALUE(400, 1000), 0);

INSERT INTO Ticket(flight\_number, place\_number, price, status\_num, customer\_id, luggage\_id)

VALUES (flight\_num, place\_num, price, 100, customer\_id, luggage\_id);

END;

CREATE OR REPLACE PROCEDURE GET\_TICKETS\_BY\_CUSTOMER(

p\_customer\_id IN NUMBER,

p\_tickets OUT SYS\_REFCURSOR

) AS

BEGIN

OPEN p\_tickets FOR

SELECT t.ticket\_num, t.flight\_number, t.place\_number, t.price, t.customer\_id, ts.status\_description

FROM Ticket t inner join ticket\_status ts on t.status\_num = ts.status\_num

WHERE t.customer\_id = p\_customer\_id;

END;

CREATE OR REPLACE PROCEDURE CANCEL\_TICKET(

p\_ticket\_num IN NUMBER,

p\_customer\_id IN NUMBER

) AS

BEGIN

UPDATE Ticket

SET status\_num = 101

WHERE ticket\_num = p\_ticket\_num AND customer\_id = p\_customer\_id;

COMMIT;

END;

CREATE OR REPLACE PROCEDURE CANCEL\_FLIGHT (

p\_flight\_num NUMBER

) AS

v\_status NUMBER;

BEGIN

-- Получаем статус рейса

SELECT status INTO v\_status FROM flight WHERE flight\_number = p\_flight\_num;

-- Проверяем статус рейса и выбрасываем исключение при несоответствии

IF v\_status <> 1 THEN

RAISE\_APPLICATION\_ERROR(-20009, 'Flight is not on "scheduled" status, it cannot be cancelled');

END IF;

-- Обновляем статус рейса и билетов

UPDATE flight SET status = 2 WHERE flight\_number = p\_flight\_num;

UPDATE ticket SET status\_num = 101 WHERE flight\_number = p\_flight\_num AND status\_num <> 101;

COMMIT;

END;

CREATE OR REPLACE PROCEDURE CHECK\_FLIGHTS AS

BEGIN

UPDATE Flight

SET status =

CASE

WHEN departure\_datetime > SYSDATE AND SYSDATE < arrive\_datetime THEN 5 -- "Executing"

WHEN SYSDATE >= arrive\_datetime THEN 3 -- "Completed"

END

WHERE status = 1; -- check only for scheduled flights

END;

create or replace procedure ADD\_PLANE(

p\_plane\_model nvarchar2,

p\_pacles\_capacity number,

max\_luggage\_weigth number

)

as

begin

insert into Plane(PLANE\_MODEL, PLACES\_CAPACITY, MAX\_LUGGAGE\_WEIGTH)

values (p\_plane\_model,p\_pacles\_capacity,max\_luggage\_weigth);

commit;

end;

create or replace procedure ADD\_EMPLOYEE(

p\_empl\_fname nvarchar2,

p\_empl\_position number,

p\_salary number,

p\_start\_work\_date date

)

as

begin

insert into Personel(employee\_fname,employee\_position, employee\_mnth\_salary,employee\_start\_work)

values(p\_empl\_fname,p\_empl\_position,p\_salary,p\_start\_work\_date);

commit;

end;

CREATE OR REPLACE PROCEDURE REMOVE\_EMPLOYEE(

p\_empl\_id NUMBER

) AS

v\_exists NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_exists FROM Personel WHERE employee\_id = p\_empl\_id;

IF v\_exists > 0 THEN

DELETE FROM Personel WHERE employee\_id = p\_empl\_id;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'No such employee');

END IF;

END;

create or replace procedure ADD\_AIRPORT(

p\_airport\_name nvarchar2,

p\_location\_country nvarchar2,

p\_location\_city nvarchar2

)

as

begin

insert into AIRPORT(airport\_name, location\_country, location\_city)

values(p\_airport\_name,p\_location\_country,p\_location\_city);

commit;

end;

create or replace procedure ADD\_FLIGHT(

p\_plane\_id number,

p\_dep\_airport\_id number,

p\_dest\_airport\_id number,

p\_dep\_datetime date,

p\_dest\_datetime date

)

as

av\_seats NUMBER;

BEGIN

select places\_capacity

into av\_seats

from plane

where plane\_id = p\_plane\_id;

insert into Flight(plane\_id, departure\_airport\_id, destination\_airport\_id, departure\_datetime, arrive\_datetime, available\_seats,status)

values(p\_plane\_id, p\_dep\_airport\_id, p\_dest\_airport\_id, p\_dep\_datetime, p\_dest\_datetime, av\_seats, 1);

commit;

END;

create or replace procedure ADD\_CUSTOMER(

p\_cust\_fullname nvarchar2,

p\_birth\_date date,

p\_passport\_series nvarchar2

)

As begin

insert into customer(full\_name, date\_of\_birth, passport\_series)

values(p\_cust\_fullname, p\_birth\_date, p\_passport\_series);

commit; end;

create DIRECTORY DATA\_DIR2 as 'C:\BD\EXPIMP';

CREATE OR REPLACE PROCEDURE IMPORT\_AIRPORT\_XML AS

v\_file UTL\_FILE.FILE\_TYPE;

v\_line VARCHAR2(32766);

v\_airport\_name nvarchar2(40);

v\_location\_country nvarchar2(40);

v\_location\_city nvarchar2(40);

BEGIN

-- Открываем файл для чтения

v\_file := UTL\_FILE.FOPEN('DATA\_DIR2', 'exported\_data.xml', 'r', 32766);

-- Читаем строки из файла

LOOP

BEGIN

UTL\_FILE.GET\_LINE(v\_file, v\_line);

IF v\_line = '<Airport>' THEN

-- Начало записи аэропорта

v\_airport\_name := NULL;

v\_location\_country := NULL;

v\_location\_city := NULL;

ELSIF v\_line LIKE '<name>%' THEN

-- Чтение названия аэропорта

v\_airport\_name := SUBSTR(v\_line, 7, LENGTH(v\_line) - 13);

ELSIF v\_line LIKE '<country>%' THEN

-- Чтение страны аэропорта

v\_location\_country := SUBSTR(v\_line, 10, LENGTH(v\_line) - 19);

ELSIF v\_line LIKE '<city>%' THEN

-- Чтение города аэропорта

v\_location\_city := SUBSTR(v\_line, 7, LENGTH(v\_line) - 13);

ELSIF v\_line = '</Airport>' THEN

-- Конец записи аэропорта, сохраняем в базу данных

INSERT INTO Airport (airport\_name, location\_country, location\_city)

VALUES (v\_airport\_name, v\_location\_country, v\_location\_city);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Конец файла, выходим из цикла

EXIT; END; END LOOP;

UTL\_FILE.FCLOSE(v\_file); END;

CREATE OR REPLACE PROCEDURE EXPORT\_AIRPORT\_XML

AS

v\_clob CLOB;

v\_file UTL\_FILE.FILE\_TYPE;

BEGIN

SELECT DBMS\_XMLGEN.GETXML('SELECT \* FROM AIRPORT') INTO v\_clob FROM DUAL;

v\_file := UTL\_FILE.FOPEN('DATA\_DIR2', 'AIRPORT.xml', 'w');

UTL\_FILE.PUT(v\_file, v\_clob);

UTL\_FILE.FCLOSE(v\_file);

END;

CREATE OR REPLACE PROCEDURE CheckUserByPassportSeries(

pPassportSeries IN NVARCHAR2,

pResult OUT NUMBER

)

AS

vUserCount NUMBER;

BEGIN

-- Проверяем, есть ли пользователь с указанной серией паспорта

SELECT COUNT(\*) INTO vUserCount

FROM Customer

WHERE passport\_series = pPassportSeries;

IF vUserCount > 0 THEN

-- Пользователь с указанной серией паспорта существует

-- Можно продолжить проверку пароля или выполнить другие действия

pResult := 1;

ELSE

-- Пользователь с указанной серией паспорта не существует

pResult := 0;

END IF;

END;

CREATE OR REPLACE PROCEDURE GetUserInfoByPassportSeries(

pPassportSeries IN NVARCHAR2,

pCustomerId OUT NUMBER,

pFullName OUT NVARCHAR2

)

AS

BEGIN

SELECT customer\_id, full\_name

INTO pCustomerId, pFullName

FROM Customer

WHERE passport\_series = pPassportSeries;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

pCustomerId := NULL;

pFullName := NULL;

END;

DECLARE

vCustomerId NUMBER;

vFullName NVARCHAR2(100);

BEGIN

GetUserInfoByPassportSeries('AB7658412', vCustomerId, vFullName);

-- Используйте vCustomerId и vFullName для дальнейших действий

IF vCustomerId IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('User ID: ' || vCustomerId);

DBMS\_OUTPUT.PUT\_LINE('User full name: ' || vFullName);

ELSE

DBMS\_OUTPUT.PUT\_LINE('User not found');

END IF; END;