МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Программное обеспечение информационных

технологий

Специализация 1 –40 01 01 Программное обеспечение информационных

технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных авиакомпании с использованием технологии миграции данных и объектов из одной СУБД в другую»

Выполнил студент Дикун Игорь Вячеславович

(Ф.И.О.)

Руководитель проекта асс. Комарова Е.И.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Комарова Е.И.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Комарова Е.И.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2022

Содержание

[Введение 3](#_Toc122781152)

[1. Аналитический обзор литературы 4](#_Toc122781153)

[2. Разработка программной системы 6](#_Toc122781154)

[2.1 Разработка модели базы данных 6](#_Toc122781155)

[2.2 Хранимые процедуры 7](#_Toc122781156)

[2.2.1 Выборка данных для таблиц 8](#_Toc122781157)

[2.2.2 Выборка данных по поисковому запросу 8](#_Toc122781158)

[2.2.3 Заполнение таблиц 100 000 строк 8](#_Toc122781159)

[2.2.4 Вход пользователя в аккаунт 8](#_Toc122781160)

[2.2.5 Удаление данных из таблиц 9](#_Toc122781161)

[2.2.6 Добавление данных в таблицы 9](#_Toc122781162)

[2.2.7 Изменение данных в таблицах 9](#_Toc122781163)

[2.2.8 Экспорт и импорт таблицы Users в формате XML 9](#_Toc122781164)

[2.3 Индексы 10](#_Toc122781165)

[3. Миграция данных 11](#_Toc122781166)

[3.1 Миграция данных в базу данных MySQL 11](#_Toc122781167)

[3.2 Миграция данных в базу данных Oracle 12](#_Toc122781168)

[4. Тестирование, проверка работоспособности и анализ данных 15](#_Toc122781169)

[4.1 Ошибка при регистрации существующего логина 15](#_Toc122781170)

[4.2 Ошибка при некорректных данных для входа 15](#_Toc122781171)

[4.3 Ошибка при поиске несуществующего рейса 15](#_Toc122781172)

[Заключение 17](#_Toc122781173)

[Список литературы 18](#_Toc122781174)

[Приложение A 19](#_Toc122781175)

[Приложение Б 20](#_Toc122781176)

[Приложение В 20](#_Toc122781177)

[Графические материалы 31](#_Toc122781178)

# **Введение**

Целью данной работы была разработка реляционной базы данных на тему «Авиакомпания».Эта база данных составлялась для обеспечения пользователем доступа к функциям заказа билетов от авиакомпании.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Реляционная база данных — база данных, основанная на реляционной модели данных.

В качестве СУБД для базы данных была выбрана Microsoft SQL Server, в связи с ее простотой, производительностью и надежностью.

# **1. Аналитический обзор литературы**

Сейчас используются базы данных с разработанным для них программным обеспечением, позволяя уменьшить объем бумажных работ и оптимизировать работу всей системы авиакомпании, попутно увеличивая прибыль компании, использующей современные технологии.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач, например, при автоматизации бухгалтерского учета. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). СУБД взаимодействуют не с локальными, а взаимосвязанными по информации массивами, называемыми базами данных. С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

В открытом доступе рассмотреть напрямую системы управления авиакомпаниями невозможно, так они пишутся на заказ. Но можно рассмотреть структуру организации авиакомпании среди онлайн-покупки авиабилетов. Одним из таких сервисов является «AirEuropa.com».

Рассмотрим в качестве примера главную страницу сайта на рисунке 1.1.

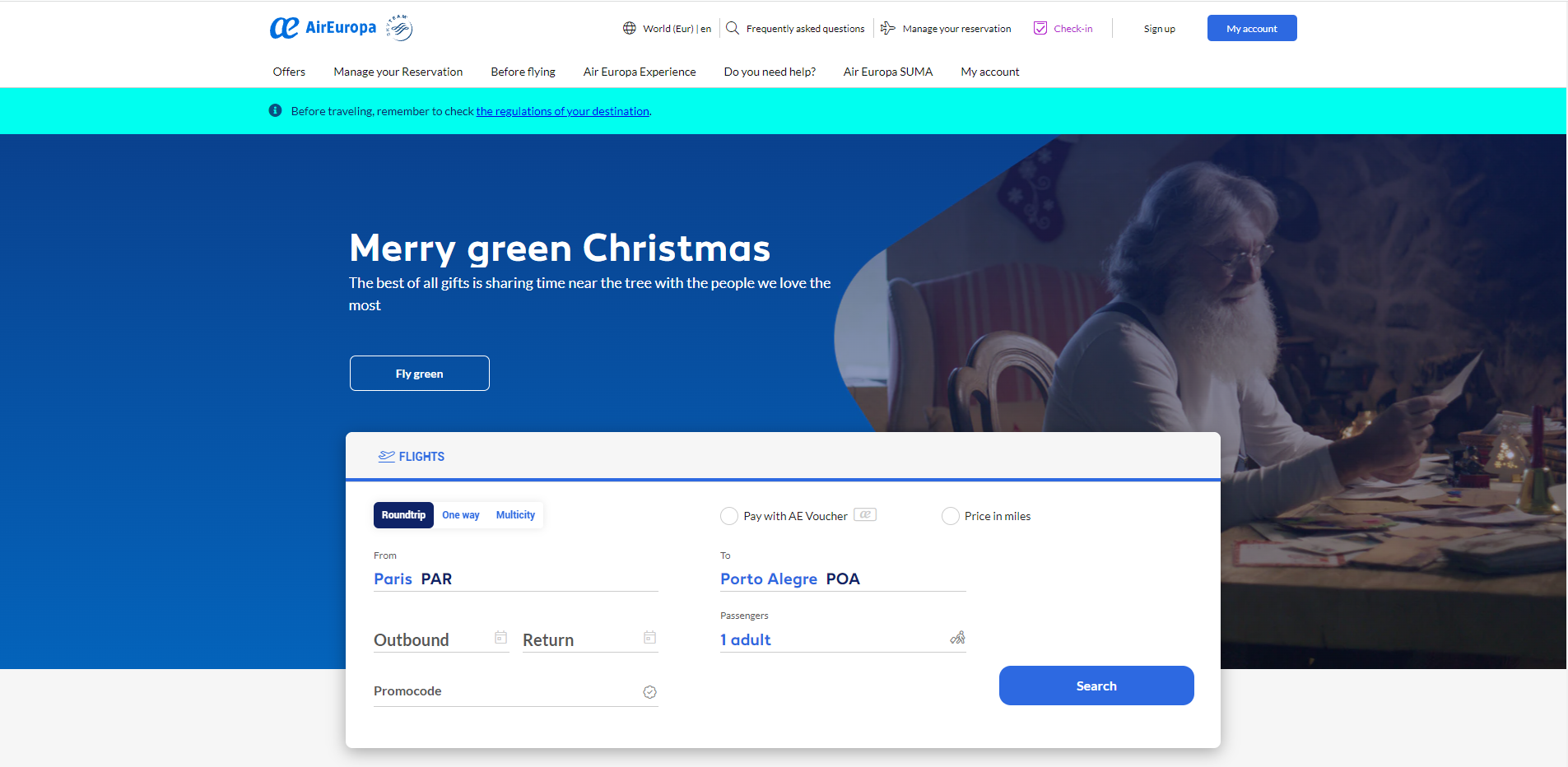


Рисунок 1.1 – Пример сайта «AirEuropa.com»

Тут структура сайта делится на поиск авиабилетов и дополнительную информацию. У поиска есть следующие особенности:

* Каждая точка имеет краткое имя;
* Выбор количества посадочных билетов;
* Дата вылета и дата возвращения назад;
* Промокод.

Так же рассмотрим сайт авиакомпании «AirFrance.com», одна из страниц из которого представлена на рисунке 1.2.

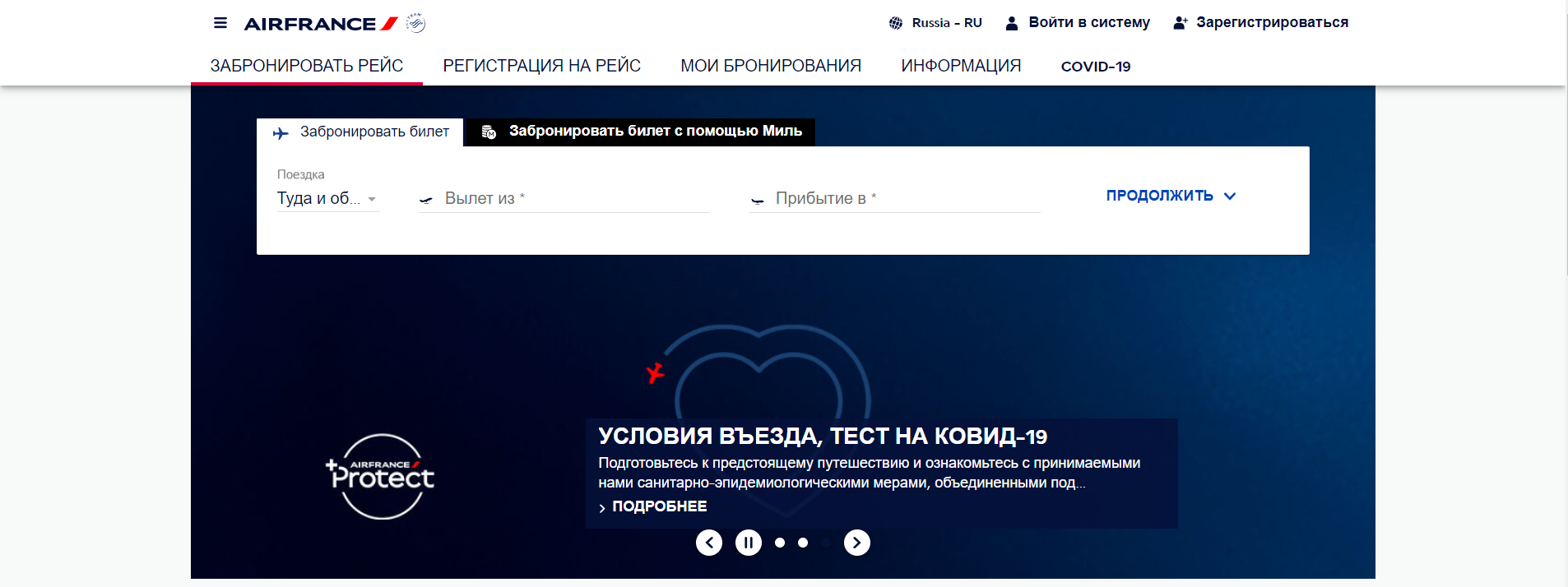


Рисунок 1.2 – Пример главной страницы «AirFrance.com»

Его основные особенности такие же, как и у «AirEuropa.сom», кроме того, что на главной странице нельзя выбрать количество пассажиров и ввести промокод.

Исходя из приведенных данных была разработана база данных, описанная в пункте 2.

# **2. Разработка программной системы**

При разработке курсового проекта понадобились следующие объекты:

1. Таблицы;
2. Хранимые процедуры;
3. Индексы;

## **2.1 Разработка модели базы данных**

Для реализации базы данных для сети магазинов было разработано 6 таблиц. Диаграмма базы со структурой связей представлена в приложении А.

Логически можно вывести 6 основных таблиц: Users, Bookings, Flights, Staff, Statistic и Airports.

ТаблицаAirportsпредставляет аэропорта авиакомпании из которых осуществляется вылет и посадка самолётов:

* airportCode – идентификатор аэропорта, тип varchar;
* name – название аэропорта, тип nvarchar,первичный ключ;
* city – город в котором расположен аэропорт, тип varchar,первичный ключ;
* airportType – вид аэропорта, тип varchar;

Таблица Staff представляет собой людей, работающих в магазинах, состоит из столбцов:

* personalId – идентификатор работника, тип int, первичный ключ;
* firstName – фамилия работника, тип nvarchar;
* secondName – имя работника, тип nvarcahr;
* lastName – отчество работника, тип nvarchar;
* birthday – день рождение работника, тип date;
* post – должность работника, тип nvarchar;
* experience – опыт работы, тип varchar;
* number – номер телефона работника, тип varchar;
* flightId – идентификатор перелёта к которому относится работник, тип int;

Таблица Bookings представляет собой заказы пользователей, состоит из столбцов:

* bookingId – идентификатор заказа, тип int, первичный ключ;
* passengerId – идентификатор пассажира, тип int,внешний ключ;
* bookingDateTime – время заказа, тип datetime;
* flightId – идентификатор перелёта, тип int, внешний ключ;
* arrivalAirport – аэропорт в который осуществляется перелёт, тип varchar;
* departureAirport – аэропорт из которого происходит перелёт, тип varchar;
* amount – количество заказанных билетов пользователем, тип int;
* price – стоимость заказанных билетов, тип int;
* status – статус заказа, тип varchar;

Таблица Flights представляет собой данные о рейсе, состоит из столбцов:

* flightid – идентификатор рейса, тип int, первичный ключ;
* flightnumber – идентификатор перелёта, тип int;
* departuredatetime – время вылета, тип date;
* departureairport – аэропорт из которого происходит вылет, тип varchar, внешний ключ;
* totalplaces – количество посадочных мест, тип int;
* arrivaldatetime – время посадки, тип date;
* arrivalairport – аэропорт в котором произошла посадка, тип varchar, внешний ключ;
* price – цена перелёта, тип int;
* status – статус в котором находится рейс, тип varchar;
* flightclass – класс рейса, тип int;

Таблица Users представляет пользователей платформы, состоит из столбцов:

* id – идентификатор пользователя, тип int, первичный ключ;
* login – логин пользователя, тип nvarchar;
* password – пароль пользователя, тип nvarchar;
* email – адрес электронной почты пользователя, тип nvarchar;
* firstName – имя пользователя, тип nvarchar;
* secondName – фамилия пользователя, тип nvarcahr;
* roles – идентификатор типа аккаунта, тип int;

Таблица Statistic представляет статистику авиакомпании, состоит из столбцов:

* confirmedflightscount – количество завершённых рейсов, тип int;
* canceledflightcount – количество отменённых рейсов, тип int;
* Profit – доход компании за определённый промежуток времени, тип int;
* Ratio – соотношение классов рейса, тип int;

Диаграмма базы данных, спроектированной в ходе разработки приведена на рисунке 2.1.

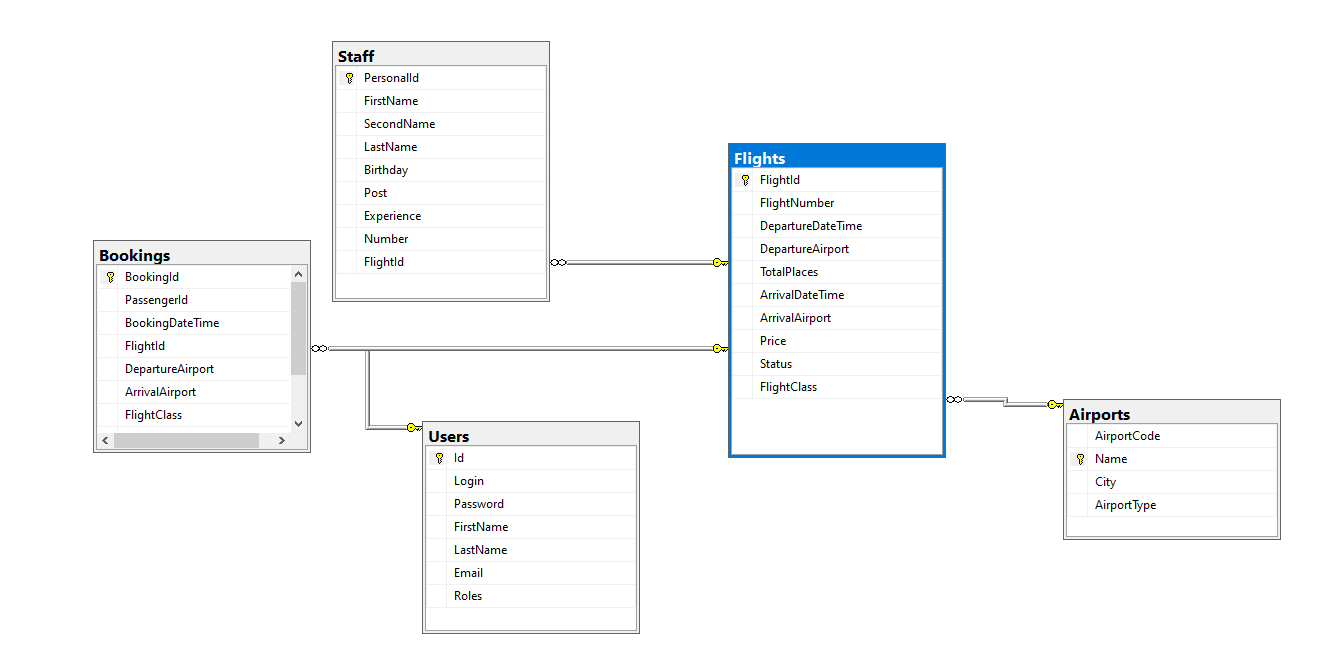


Рисунок 2.1 – Диаграмма базы данных

**2.2 Хранимые процедуры**

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

При разработке курсового проекта было создано 34 процедур для следующих целей:

1. Выборка данных для таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Вход администратора в аккаунт;
5. Вход пользователя в аккаунт;
6. Удаление данных из таблиц;
7. Добавление данных в таблицы;
8. Изменение данных в таблицах;
9. Экспорт и импорт таблицы Users в формате xml;

Листинг SQL-кода для создания процедур находится в приложении В.

### **2.2.1 Выборка данных для таблиц**

Для вывода данных из таблиц были написаны следующие процедуры: sp\_SelectUsersAdmin, sp\_ShowAllAirports, sp\_GetFlights, и sp\_ShowOrder, sp\_ShowStaff, sp\_ShowStatistic.

### **2.2.2 Выборка данных по поисковому запросу**

Для вывода данных из таблиц по поисковому запросу была написана процедура sp\_SearchFlight. sp\_SearchFlight выводит информацию о рейсе из таблицы Flights при вводе аэропорта вылета и посадки, а так же дате рейса.

### **2.2.3 Заполнение таблиц 100 000 строк**

Для заполнения таблиц было разработаны процедуры для генерации случайных строк и процедуры заполнения таблиц этими строками.

Процедуры заполнения таблиц: InsertThousands. Они были разработаны соответственно для заполнения таблицs:Users.

### **2.2.4 Вход пользователя в аккаунт**

Процедура sp\_GetUser ищет данные о пользователе по его логину и паролю. Процедура возвращает данные приложению, так как со стороны логики приложение проводится сверка пароля, а так же роли, которая присвоена пользователю. Данная процедура представлена ниже на рисунке 2.1.

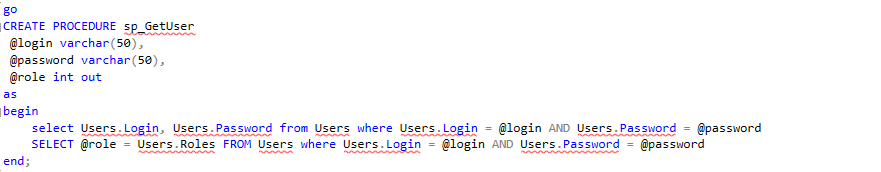


Рисунок 2.2 – Пример создания процедуры sp\_GetUser

### **2.2.5 Удаление данных из таблиц**

Для управления базой данных были созданы процедуры удаления: sp\_DeleteAirports, sp\_DeleteFlight, sp\_DeleteOrder, sp\_DeleteUsers и sp\_DeleteStaff. Они были разработаны соответственно для удаления строк из таблиц: Airports, Flights, Bookings, Users, Staff.

### **2.2.6 Добавление данных в таблицы**

Для управления базой данных были созданы процедуры добавления новых строк: sp\_InsertUsers, sp\_AddAirports, sp\_AddFlight, sp\_NewAddOrder, sp\_InsertStaff и sp\_CompletedCount. Они были разработаны соответственно для заполнения таблиц: Users, Airports, Flights, Bookings, Statistic и Staff.

### **2.2.7 Изменение данных в таблицах**

Для управления базой данных были созданы процедуры обновления: UpdateUsers, UpdateAirports, UpdateInfoFlight, OrderStatus, UpdateStaffInfo. Они были разработаны соответственно для обновления данных в таблицах: Users, Airports, Flights, Bookings, Staff.

### **2.2.8 Экспорт и импорт таблицы Users в формате XML**

Для экспорта таблицы Users в формате xml была разработана процедура ExportXML. Для формирования xml в select запросе используется конструкция FOR XML. Для вывода в файл была использована расширенная хранимая процедура xp\_cmdshell. Процедура ExportML представлена ниже на рисунке 2.2.

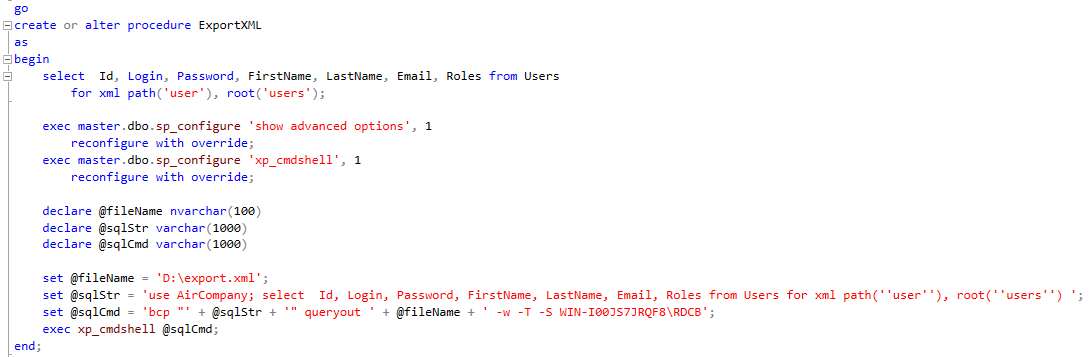


Рисунок 2.3 – Пример создания процедуры ExportXML

Для импорта данных в таблицу Users, из файла формата xml, была разработана процедура ImportFromXML. Для получения xml файла и последующего разбора со вставкой используется конструкция FROM OPENROWSET совместно с параметром BULK. Процедура ImportFromXML представлена ниже на рисунке 2.3.

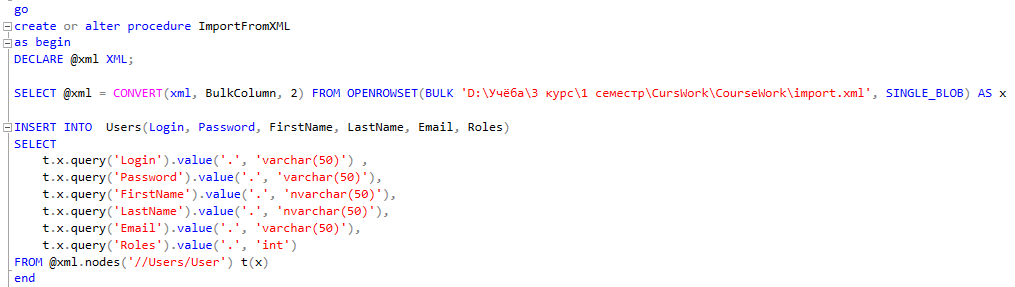


Рисунок 2.4 – Пример создания процедуры ImportFromXML

## **2.3 Индексы**

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. В связи с необходимостью выборки таблиц, заполненных большим количеством строк, было принято решение разработать 1 индекс для таблиц: Users.

**2.4 Вывод по разработке таблиц базы данных**

В результате разработки таблиц модели базы данных были выявлены необходимые данные, которые должны содержатся в таблицах. Также были определены типы данных и выделены необходимые связи для получения данных из других таблиц, актуализации информации и оптимизации ее получения.

База данных и таблицы были спроектированы на основе диаграммы вариантов использования, разработанной для клиента и администратора.

Диаграмма вариантов использования для пользователя представлена на рисунке 2.5.



Рисунок 2.5 – Диаграмма вариантов использования для клиента

# **3. Миграция данных**

В данном разделе будут описаны технологии, использованные в процессе разработки базы данных.

# **3.1 Миграция данных в базу данных MySQL**

Была реализована технология миграция данных и их объектов в такие базы данных как MySQL, Oracle. Для реализации миграции данных в базу данных MySQL потребовалось настроить программный интерфейс доступа к базам данных. Данный интерфейс представлен ниже на рисунке 3.1.

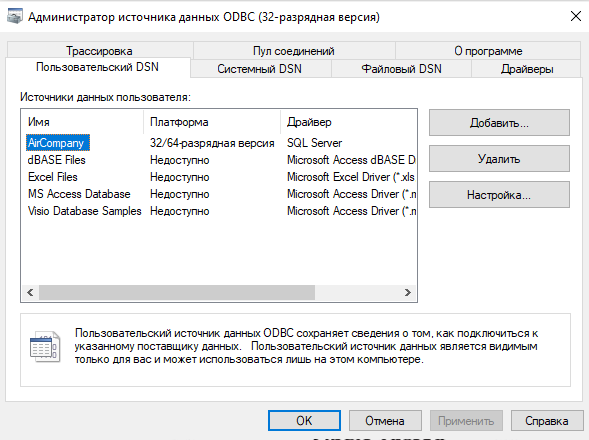


Рисунок 3.1 – Программный интерфейс доступа к базам данных.

После настройки программного интерфейса ODCB требовалось перейти в базу данных MySQL и настроить Migration Wizard с помощью которого мы осуществляем миграцию данных в данную базу данных, представлен ниже на рисунке 3.2.

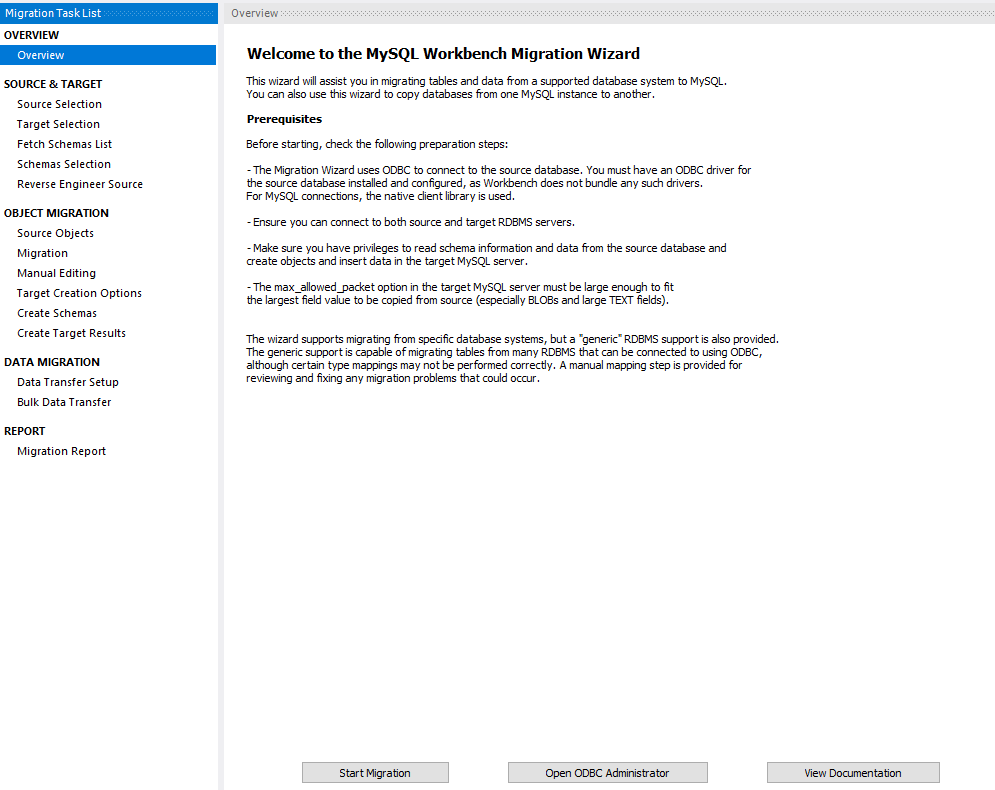


Рисунок 3.2 – Migration Wizard.

Как только прохождение всех этапов миграции в Migration Wizard завершившихся успешно, то на выходе мы получали полностью мигрированную базу данных AirCompany из SQL Server в MySQL показанную на рисунку 3.3.

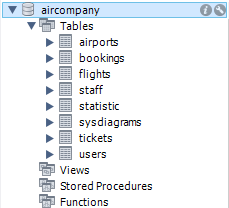


Рисунок 3.3 - Мигрированная база данных AirCompany.

Чтобы убедиться все ли данные были мигрированы мы можем проверить наши таблицы и убедиться в этом с помощью простого select запроса к ним, на пример к таблице Flights, представленном ниже на рисунке 3.4.

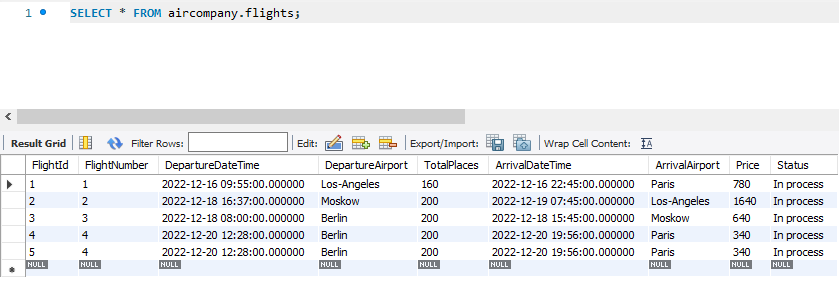


Рисунок 3.4 – Запрос к Microsoft SQL Server.

# **3.2 Миграция данных в базу данных Oracle**

Также была реализована технология миграции данных в базу данных Oracle SQL Developer. Для миграции данных нам потребовался SQL Developer в котором мы создаём подключение к SQL Server, в нашем случае это MSSQL\_Air представленное ниже на рисунке 3.5.

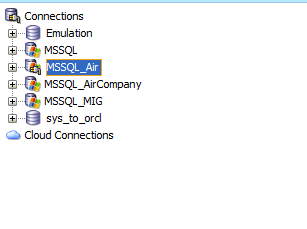


Рисунок 3.5 – Подключение MSSQL\_Air.

Неотъемлемой частью миграции является установка JTDS драйвера без которого не была бы возможна сам миграция данных. Данный драйвер позволяет создавать подключение к SQL Server и доставать оттуда данные. Интерфейс расширения драйвера представлен ниже на рисунке 3.6.

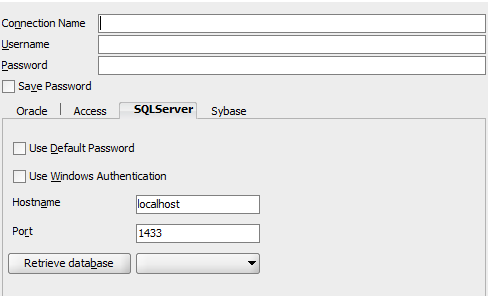


Рисунок 3.6– Расширение драйвера JTDS.

После успешной настройки драйвера мы можем получить разрешение на соединение с SQL Server и также мигрировать данные из базы данных. После успешной миграции данных таблица появится в списке наших таблиц подключения MSSQL\_Air. Представлена ниже на рисунке 3.7.

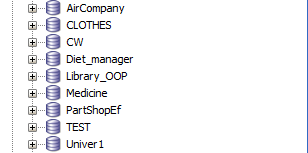


Рисунок 3.7 – Мигрированные базы данных в Oracle.

Чтобы проверить полностью ли мигрировались данные, то потребуется написать простой select запрос к нашей таблице Airports базы данных AirCompany и проверить. Представлено на рисунке 3.8.

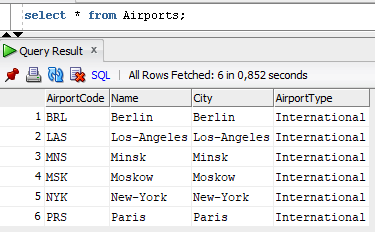


Рисунок 3.8 – Мигрированные данные таблицы Airports.

# **4. Тестирование, проверка работоспособности и анализ данных**

Тестирование отыгрывает важную роль при разработке любого программного продукта. Чем качественнее тестирование, тем лучше в итоге должен выйти конечный продукт. Часто можно столкнуться с ситуацией, когда тестирование программного кода проходит очень кропотливо, а на тестирование базы данных времени уже не остается либо оно делается по остаточному принципу.

Соответственно, для исключения такой ситуации в данном разделе будет описано тестирование базы данных.

# **4.1 Ошибка при регистрации существующего логина**

При регистрации пользователя может возникнуть ситуация, при которой данный логин или же электронная почта уже существует в базе данных. Ошибка отображена на рисунке 4.1.

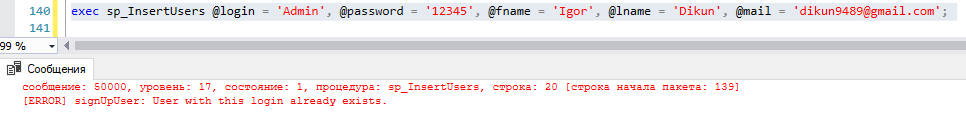


Рисунок 4.1 – Ошибка при регистрации существующего логина

# **4.2 Ошибка при некорректных данных для входа**

При предоставлении некорректных данных для входа (несуществующий логин или неверный пароль) также должна возвращаться ошибка и сообщение. Ошибка отображена на рисунке 4.2.

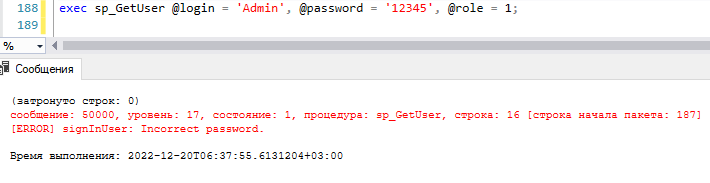
****

Рисунок 4.2 – Ошибка при некорректных данных для входа

# **4.3 Ошибка при поиске несуществующего рейса**

При поиске рейса, которого не существует, то должна возвращаться ошибка и выводится сообщение. Ошибка отображена на рисунке 4.3

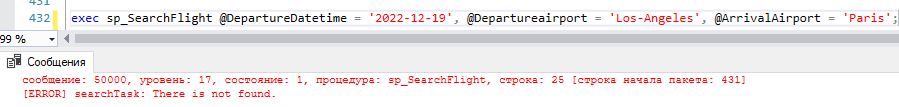


Рисунок 4.3 – ошибка при поиске несуществующего рейса.

1. **Тестирование производительности**

Производительность БД является решающим фактором эффективности управленческих и коммерческих приложений. Если поиск или запись данных выполняется медленно – способность к нормальной работе приложения падает. Существует единственный путь выяснить причину плохой производительности – выполнить количественные измерения и определить, что является причиной проблемы производительности.

Проблемы выявления узких мест производительности баз данных напрямую связаны с метриками, методами измерений производительности и технологией их выполнения.

Для того чтобы правильно организовать процесс тестирования БД, тестировщики должны обладать хорошими знаниями SQL и DML и иметь ясное представление о внутренней структуре БД. Это самый лучший и надежный способ тестирования БД особенно для приложений с низким и средним уровнем сложности. Данный метод не только дает уверенность, что тестирование выполнено качественно, но также повышает мастерство написания SQL-запросов.

В MS SQL Server оптимизация запросом в основном заключается в построение индексов над таблицами, и изменением плана запроса. Для тестирования производительности в таблицу Flight были добавлены 100 000 записей.

Ниже на рисунке 5.1 представлен результат статистического анализа и компиляции SQL Server до добавления индекса.

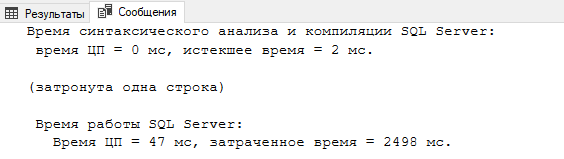


Рисунок 5.1– Статический анализ после создания индекса покрытия

# **Заключение**

Результатом выполнения курсового проекта стала база данных в базе данные SQL Server в СУБД SQL Server Management Studio 18.

Были реализованы основные требования, указанные в листе задания вместе со следующими пунктами:

* Регистрация и авторизация пользователя;
* Управление авиарейсами;
* Оформление заказа;
* Управление данными об аэропортах;
* Управление и хранение информации о рабочем персонале;
* Анализ деятельности авиакомпании;
* Импорт и экспорт данных таблицы в XML формат;
* Заполнение таблицы на 100 000 записей;
* Технология миграции данных.

Проект не может окончательно претендовать на полноценный коммерческий продукт, поскольку для демонстрации технологии и работы с базой данных были опущены некоторые атрибуты современных площадок, реализация которых не относится к основной теме курсового проекта.

Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

**Список литературы**

1. Mssql [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://www.npmjs.com/package/mssql. – Дата доступа: 28.10.2021.

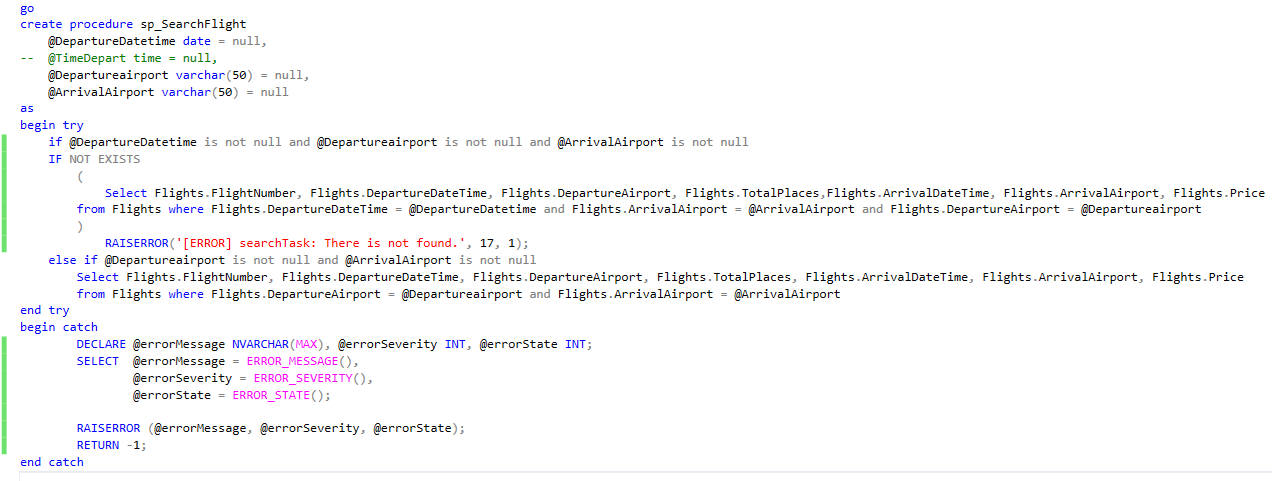
2. RESTORE Statements (Transact-SQL) [Электронный ресурс] / Microsoft. – Режим доступа: https://docs.microsoft.com/en-us/sql/t-sql/statements/restore-statements-transact-sql?view=sql-server-ver15. – Дата доступа: 04.11.2021.

3. BACKUP (Transact-SQL) [Электронный ресурс] / Microsoft. – Режим доступа: https://docs.microsoft.com/en-us/sql/t-sql/statements/backup-transact-sql?view=sql-server-ver15. – Дата доступа: 04.11.2021.

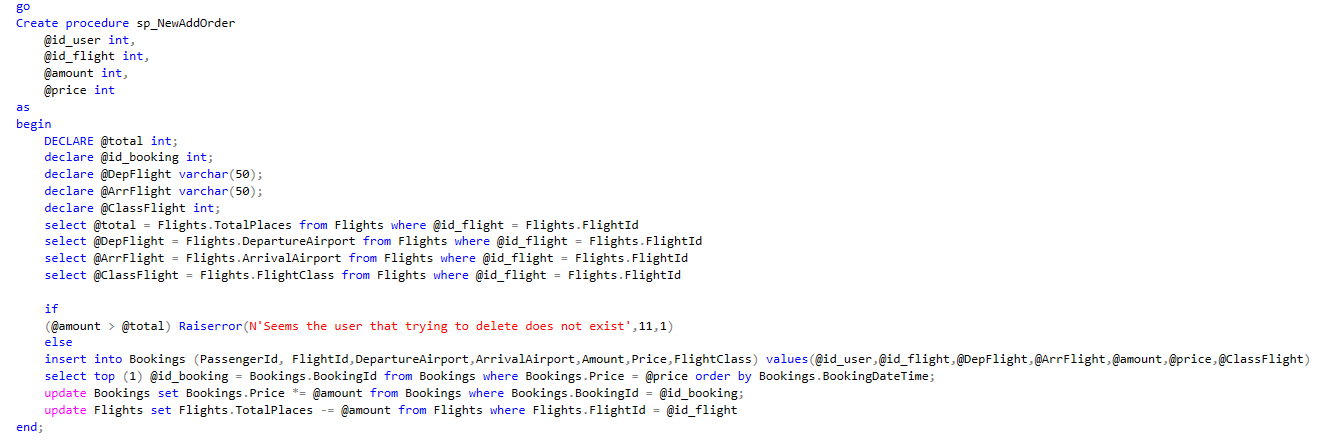
4. Restore a Database Backup Using SSMS [Электронный ресурс] / Microsoft. – Режим доступа: https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-ver15. – Дата доступа: 15.11.2021.

# **Приложение A**

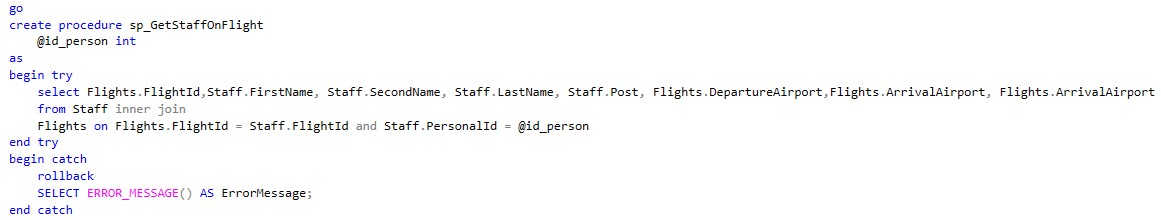
Процедура sp\_SearchFlight



Процедура sp\_NewAddOrder

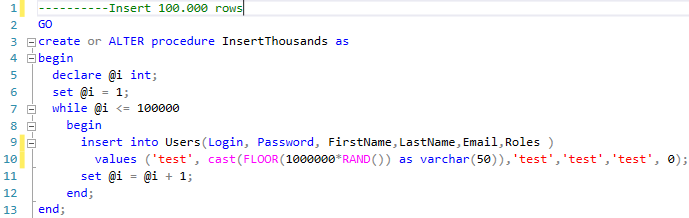


Процедура sp\_GetStaffOnFlight



# **Приложение Б**

Добавление 100 000 строк



# **Приложение В**

Создание процедур

go

CREATE PROCEDURE sp\_InsertUsers

@login varchar(50),

@password varchar(50),

@fname nvarchar(50),

@lname nvarchar(50),

@mail varchar(50)

AS

BEGIN try

IF EXISTS (SELECT \* FROM Users WHERE login = LTRIM(RTRIM(@login)))

RAISERROR ('[ERROR] signUpUser: User with this login already exists.', 17, 1);

INSERT INTO Users(Login,Password,FirstName,LastName,Email) values (@login,@password,@fname,@lname,@mail)

END try

begin catch

DECLARE @errorMessage NVARCHAR(MAX), @errorSeverity INT, @errorState INT;

SELECT @errorMessage = ERROR\_MESSAGE(),

@errorSeverity = ERROR\_SEVERITY(),

@errorState = ERROR\_STATE();

RAISERROR (@errorMessage, @errorSeverity, @errorState);

RETURN -1;

END CATCH;

go

CREATE PROCEDURE sp\_UpdateInfoInAdminPanel

@role int,

@id\_users int,

@login varchar(50),

@password varchar(50),

@fname nvarchar(50),

@lname nvarchar(50),

@mail varchar(50)

as

begin

update Users set Users.Roles = @role, Users.Login = @login, Users.Password = @password, Users.FirstName = @fname, Users.LastName = @lname, Users.Email = @mail

where Users.Id = @id\_users

end;

go

CREATE PROCEDURE sp\_GetUser

@login varchar(50),

@password varchar(50),

@role int out

as

begin

DECLARE @countWithThisEmailAndPassword INT;

SET @countWithThisEmailAndPassword = (

SELECT COUNT(\*)

FROM users

WHERE

password = @password);

IF (@countWithThisEmailAndPassword = 0)

RAISERROR('[ERROR] signInUser: Incorrect password.', 17, 1);

ELSE

PRINT 'You successfully signed in! Hello, ' + LTRIM(RTRIM(@login));

select Users.Login, Users.Password from Users where Users.Login = @login AND Users.Password = @password

SELECT @role = Users.Roles FROM Users where Users.Login = @login AND Users.Password = @password

RETURN 1;

end;

go

CREATE PROCEDURE sp\_UpdateUsers

@login varchar(50),

@password varchar(50),

@mail varchar(50)

as

begin

update Users set Users.Password = @password where Users.Login = @login

update Users set Users.Email = @mail where Users.Login = @login

end;

go

CREATE PROCEDURE sp\_DeleteUsers

@id\_user int

as

begin

delete Users where Users.Id = @id\_user

end;

go

CREATE PROCEDURE sp\_SelectUsersInAdmin

as

begin

Select \* from Users

end;

go

Create procedure sp\_SelectUsersInUser

@id\_users int out

as

begin

Select \* from Users where Users.Id = @id\_users

end;

go

Create procedure sp\_AddAirports

@AirportCode varchar(50),

@Name varchar(50),

@City varchar(50),

@AirportType varchar(50)

as

begin

INSERT INTO Airports(AirportCode,Name,City,AirportType) values (@AirportCode,@Name,@City,@AirportType)

end;

go

Create procedure sp\_DeleteAirports

@AirportCode varchar(50)

as

begin

delete Airports where Airports.AirportCode = @AirportCode

end;

go

Create procedure sp\_UpdateAirports

@AirportCode varchar(50),

@Name varchar(50),

@City varchar(50),

@AirportType varchar(50)

as

begin

update Airports set Airports.AirportCode = @AirportCode where Airports.AirportCode = @AirportCode

update Airports set Airports.Name = @Name where Airports.AirportCode = @AirportCode

update Airports set Airports.City = @City where Airports.AirportCode = @AirportCode

update Airports set Airports.AirportType = @AirportType where Airports.AirportCode = @AirportCode

end;

go

Create procedure sp\_ShowAllAirports

as

begin

select \* from Airports

end;

go

Create procedure sp\_AddFlight

@Flightnumber int,

@DepartureDateTime date,

@TimeDeparture time,

@Departureairport varchar(50),

@Totalplaces int, --ограничение

@ArrivalDateTime date,

@TimeArrival time,

@Arrivalairport varchar(50),

@price int,

@FlightClass int

as

begin

insert into Flights(FlightNumber, DepartureDateTime,TimeDepart, DepartureAirport, TotalPlaces, ArrivalDateTime,TimeArrival,ArrivalAirport,Price,FlightClass)

values (@Flightnumber, @DepartureDateTime,@TimeDeparture, @Departureairport, @Totalplaces,@ArrivalDateTime,@TimeArrival, @Arrivalairport, @price,@FlightClass)

end;

go

Create procedure sp\_UpdateInfoFlight

@Flights\_id int,

@Flightnumber int,

@DepartureDatetime datetime,

@Departureairport varchar(50),

@Totalplaces int,

@ArrivalDatetime datetime,

@Arrivalairport varchar(50),

@price int,

@class int

as

begin

update Flights set Flights.FlightNumber = @Flightnumber where Flights.FlightId = @Flights\_id

update Flights set Flights.DepartureDateTime = @DepartureDatetime where Flights.FlightId = @Flights\_id

update Flights set Flights.DepartureAirport = @Departureairport where Flights.FlightId = @Flights\_id

update Flights set Flights.TotalPlaces = @Totalplaces where Flights.FlightId = @Flights\_id

update Flights set Flights.ArrivalDateTime = @ArrivalDatetime where Flights.FlightId = @Flights\_id

update Flights set Flights.ArrivalAirport = @Arrivalairport where Flights.FlightId = @Flights\_id

update Flights set Flights.Price = @price where Flights.FlightId = @Flights\_id

update Flights set Flights.FlightClass = @class where Flights.FlightId = @Flights\_id

end;

go

create procedure sp\_FlightStatusCompleted

@status varchar(50) = 'Completed',

@Flight\_id int

as

begin

update Flights set Flights.Status = @status where Flights.FlightId = @Flight\_id

end;

go

create procedure sp\_FlightStatusCanceled

@status varchar(50) = 'Canceled',

@Flight\_id int

as

begin

update Flights set Flights.Status = @status where Flights.FlightId = @Flight\_id

update Bookings set Bookings.Status = @status where Bookings.FlightId = @Flight\_id

end;

go

Create procedure sp\_DeleteFlight

@Flights\_id int

as

begin

Delete Flights where Flights.FlightId = @Flights\_id

end;

go

Create procedure sp\_GetFlights

as

begin

select\*from Flights

end;

go

create procedure sp\_SearchFlight

@DepartureDatetime date = null,

@TimeDepart time = null,

@Departureairport varchar(50) = null,

@ArrivalAirport varchar(50) = null

as

begin try

if @DepartureDatetime is not null and @Departureairport is not null and @ArrivalAirport is not null and @TimeDepart is not null

Select Flights.FlightNumber, Flights.DepartureDateTime,Flights.TimeDepart, Flights.DepartureAirport, Flights.TotalPlaces,Flights.ArrivalDateTime,Flights.TimeArrival, Flights.ArrivalAirport, Flights.Price

from Flights where Flights.DepartureDateTime = @DepartureDatetime and Flights.ArrivalAirport = @ArrivalAirport and Flights.DepartureAirport = @Departureairport and Flights.TimeDepart = @TimeDepart

else if @Departureairport is not null and @ArrivalAirport is not null

Select Flights.FlightNumber, Flights.DepartureDateTime, Flights.DepartureAirport, Flights.TotalPlaces, Flights.ArrivalDateTime, Flights.ArrivalAirport, Flights.Price

from Flights where Flights.DepartureAirport = @Departureairport and Flights.ArrivalAirport = @ArrivalAirport

end try

begin catch

DECLARE @errorMessage NVARCHAR(MAX), @errorSeverity INT, @errorState INT;

SELECT @errorMessage = ERROR\_MESSAGE(),

@errorSeverity = ERROR\_SEVERITY(),

@errorState = ERROR\_STATE();

RAISERROR (@errorMessage, @errorSeverity, @errorState);

RETURN -1;

end catch

go

Create procedure sp\_NewAddOrder

@id\_user int,

@id\_flight int,

@amount int,

@price int

as

begin

DECLARE @total int;

declare @id\_booking int;

declare @DepFlight varchar(50);

declare @ArrFlight varchar(50);

declare @ClassFlight int;

select @total = Flights.TotalPlaces from Flights where @id\_flight = Flights.FlightId

select @DepFlight = Flights.DepartureAirport from Flights where @id\_flight = Flights.FlightId

select @ArrFlight = Flights.ArrivalAirport from Flights where @id\_flight = Flights.FlightId

select @ClassFlight = Flights.FlightClass from Flights where @id\_flight = Flights.FlightId

if

(@amount > @total) Raiserror(N'Seems the user that trying to delete does not exist',11,1)

else

insert into Bookings (PassengerId, FlightId,DepartureAirport,ArrivalAirport,Amount,Price,FlightClass) values(@id\_user,@id\_flight,@DepFlight,@ArrFlight,@amount,@price,@ClassFlight)

select top (1) @id\_booking = Bookings.BookingId from Bookings where Bookings.Price = @price order by Bookings.BookingDateTime;

update Bookings set Bookings.Price \*= @amount from Bookings where Bookings.BookingId = @id\_booking;

update Flights set Flights.TotalPlaces -= @amount from Flights where Flights.FlightId = @id\_flight

end;

go

Create procedure sp\_OrderStatus

@status varchar(50) = 'Confirmed',

@id\_booking int

as

begin

update Bookings set Bookings.Status = @status from Bookings where Bookings.BookingId = @id\_booking

end;

go

Create procedure sp\_DeleteOrder

@id\_booking int,

@amount int

as

begin

select @amount = Bookings.Amount from Bookings where Bookings.BookingId = @id\_booking

update Flights set Flights.TotalPlaces += @amount from Flights inner join Bookings on Flights.FlightId = Bookings.FlightId and Bookings.BookingId = @id\_booking

delete from Bookings where Bookings.BookingId = @id\_booking

end;

go

Create procedure sp\_GetOrderById

@id\_booking int

as

begin

Select \* from Bookings where Bookings.BookingId = @id\_booking

end;

go

Create procedure sp\_GetUserOrder

@id\_user int

as

begin try

SELECT Bookings.BookingId, Flights.DepartureDateTime, Flights.DepartureAirport,

Flights.ArrivalDateTime, Flights.ArrivalAirport, Bookings.Amount

FROM Bookings INNER JOIN

Flights on Flights.FlightId = Bookings.FlightId and Bookings.PassengerId = @id\_user

end try

begin catch

rollback

SELECT ERROR\_MESSAGE() AS ErrorMessage;

end catch

go

Create procedure sp\_ShowOrder

as

begin

Select \* from Bookings

end;

go

create procedure sp\_InsertStaff

@FirstName nvarchar(50),

@SecondName nvarchar(50),

@LastName nvarchar(50),

@Birthday date,

@Post nvarchar(50),

@Experience varchar(10),

@Number varchar(20),

@FlightId int

as

begin

INSERT INTO Staff(FirstName,SecondName,LastName,Birthday,Post,Experience,Number,FlightId)

values (@FirstName,@SecondName,@LastName,@Birthday,@Post,@Experience,@Number,@FlightId)

end;

go

create procedure sp\_DeleteStaff

@id\_person int

as

begin

delete Staff where Staff.PersonalId = @id\_person

end;

go

create procedure sp\_UpdateStaffInfo

@PersonId int,

@FirstName nvarchar(50),

@SecondName nvarchar(50),

@LastName nvarchar(50),

@Birthday date,

@Post nvarchar(50),

@Experience varchar(10),

@Number varchar(20),

@FlightId int

as

begin

update Staff set Staff.FirstName = @FirstName where Staff.PersonalId = @PersonId

update Staff set Staff.SecondName = @SecondName where Staff.PersonalId = @PersonId

update Staff set Staff.LastName = @LastName where Staff.PersonalId = @PersonId

update Staff set Staff.Birthday = @Birthday where Staff.PersonalId = @PersonId

update Staff set Staff.Post = @Post where Staff.PersonalId = @PersonId

update Staff set Staff.Experience = @Experience where Staff.PersonalId = @PersonId

update Staff set Staff.Number = @Number where Staff.PersonalId = @PersonId

update Staff set Staff.FlightId = @FlightId where Staff.PersonalId = @PersonId

end;

go

create procedure sp\_GetStaffOnFlight

@id\_person int

as

begin try

select Flights.FlightId,Staff.FirstName, Staff.SecondName, Staff.LastName, Staff.Post, Flights.DepartureAirport,Flights.ArrivalAirport, Flights.ArrivalAirport

from Staff inner join

Flights on Flights.FlightId = Staff.FlightId and Staff.PersonalId = @id\_person

end try

begin catch

rollback

SELECT ERROR\_MESSAGE() AS ErrorMessage;

end catch

go

create procedure sp\_ShowStaff

as

begin

select \* from Staff;

end;

go

create procedure sp\_CompletedCount

as

begin

DECLARE @statusConfirmed int;

DECLARE @statusCanceled int;

DECLARE @Profit int;

Declare @Oneclass decimal;

Declare @Twoclass decimal;

Declare @Delenie decimal;

select @statusConfirmed = (select COUNT(\*) from Flights where Flights.Status = 'Completed')

select @statusCanceled = (select COUNT(\*) from Flights where Flights.Status = 'Canceled')

select @Profit =SUM(Price) from Bookings where CAST(BookingDateTime AS time) BETWEEN CAST('2022-12-19 21:26:18.557' AS time) AND CAST('2022-12-19 21:36:45.180' AS time) and Status = 'Confirmed';

select @Oneclass = count(\*) from Bookings where FlightClass = '1';

select @Twoclass = count(\*) from Bookings where FlightClass = '2';

select @Delenie = @Oneclass/@Twoclass;

insert into Statistic(ConfirmedFlightsCount,CanceledFlightCount,Profit,Ratio)

values (@statusConfirmed,@statusCanceled,@Profit,@Delenie)

select\*from Statistic

end;

# **Графические материалы**

Диаграмма вариантов использования

