

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА» (СПбГУТ)
ФАКУЛЬТЕТ ИНСТИТУТ НЕПРЕРЫВНОГО ОБРАЗОВАНИЯ (ИНО)**

СБОРНИК ПРАКТИЧЕСКИХ РАБОТ

по теме:

**«Проектирование авиационной базы данных: от
ERD-моделирования до бизнес-аналитики»**

Рыжкова Дарья Анатольевна

IV курса группы ПИБ-113

(09.03.04 - Программная инженерия)

студенческий билет № **1905218**

Санкт-Петербург, 2025

ПРАКТИЧЕСКАЯ РАБОТА №1

Провести полный цикл работы с базой данных «airport», включая создание и настройку таблиц в соответствии с ERD-диаграммой, импорт данных из CSV-файлов в таблицы с проверкой корректности загрузки и решение практической задачи — определение доступных рейсов из Парижа с расчетом длительности перелетов.

Задание практической работы

1. Необходимо создать таблицы в соответствии со следующей ERD – диаграммой:

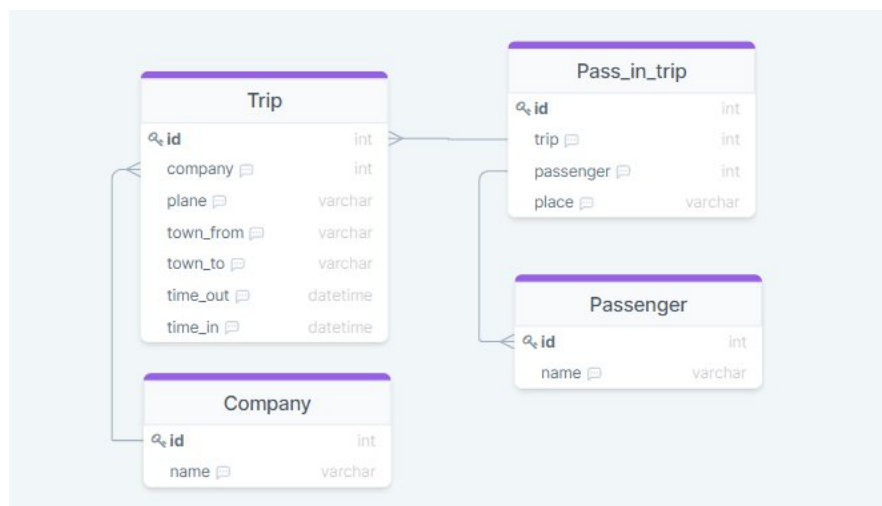


Рисунок 1. ERD-диаграмма БД задания практической работы

2. Далее следует заполнить содержимое таблиц. Необходимые данные для заполнения таблиц. Для таблиц, в которых содержится более 5 записей найти способ автоматического импорта в базу.

3. После завершения импорта таблиц, необходимо выполнить запрос, который выдаст информацию на вопрос: *В какие города можно улететь из Парижа (Paris) и сколько времени это займёт?*

Поля в результирующей таблице:

- ☐ town_to (город прибытия)
- ☐ flight_time (количество времени)

Используйте конструкцию "as flight_time" для вывода необходимого времени. Формат для вывода времени: HH:MM:SS.

Таблица Company

id	name
1	Don_avia
2	Aeroflot
3	Dale_avia
4	air_France
5	British_AW

Таблица Passenger

id	name
1	Bruce Willis
2	George Clooney
3	Kevin Costner
4	Donald Sutherland
5	Jennifer Lopez
6	Ray Liotta
7	Samuel L. Jackson
8	Nikole Kidman
9	Alan Rickman
10	Kurt Russell
11	Harrison Ford
12	Russell Crowe
13	Steve Martin
14	Michael Caine
15	Angelina Jolie
16	Mel Gibson
17	Michael Douglas
18	John Travolta
19	Sylvester Stallone
20	Tommy Lee Jones
21	Catherine Zeta-Jones
22	Antonio Banderas
23	Kim Basinger
24	Sam Neill
25	Gary Oldman
26	Clint Eastwood
27	Brad Pitt
28	Johnny Depp
29	Pierce Brosnan
30	Sean Connery
31	Bruce Willis
37	Mullah Omar

Таблица Pass in trip

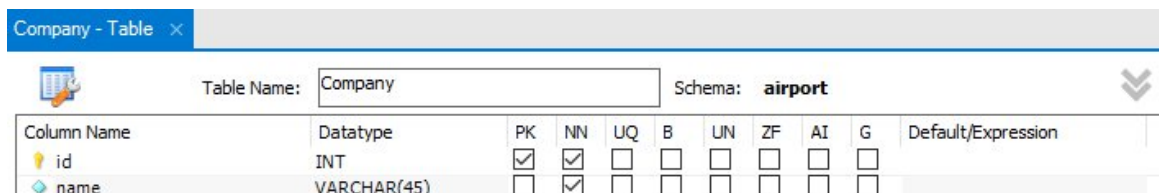
id	trip	passenger	place
1	1100	1	1a
2	1123	3	2a
3	1123	1	4c
4	1123	6	4b
5	1124	2	2d
6	1145	3	2c
7	1181	1	1a
8	1181	6	1b
9	1181	8	3c
10	1181	5	1b
11	1182	5	4b
12	1187	8	3a
13	1188	8	3a
14	1182	9	6d
15	1145	5	1d
16	1187	10	3d
17	8882	37	1a
18	7771	37	1c
19	7772	37	1a
20	8881	37	1d
21	7778	10	2a
22	7772	10	3a
23	7771	11	4a
24	7771	11	1b
25	7771	11	5a
26	7772	12	1d
27	7773	13	2d
28	7772	13	1b
29	8882	14	3d
30	7771	14	4d
31	7771	14	5d
32	7772	14	1c

Таблица Trip

id	company	plane	town_from	town_to	time out	time in
1100	4	Boeing	Rostov	Paris	1900-01-01T14:30:00.000Z	1900-01-01T17:50:00.000Z
1101	4	Boeing	Paris	Rostov	1900-01-01T08:12:00.000Z	1900-01-01T11:45:00.000Z
1123	3	TU-154	Rostov	Vladivostok	1900-01-01T16:20:00.000Z	1900-01-02T03:40:00.000Z
1124	3	TU-154	Vladivostok	Rostov	1900-01-01T09:00:00.000Z	1900-01-01T19:50:00.000Z
1145	2	IL-86	Moscow	Rostov	1900-01-01T09:35:00.000Z	1900-01-01T11:23:00.000Z
1146	2	IL-86	Rostov	Moscow	1900-01-01T17:55:00.000Z	1900-01-01T20:01:00.000Z
1181	1	TU-134	Rostov	Moscow	1900-01-01T06:12:00.000Z	1900-01-01T08:01:00.000Z
1182	1	TU-134	Moscow	Rostov	1900-01-01T12:35:00.000Z	1900-01-01T14:30:00.000Z
1187	1	TU-134	Rostov	Moscow	1900-01-01T15:42:00.000Z	1900-01-01T17:39:00.000Z
1188	1	TU-134	Moscow	Rostov	1900-01-01T22:50:00.000Z	1900-01-02T00:48:00.000Z
1195	1	TU-154	Rostov	Moscow	1900-01-01T23:30:00.000Z	1900-01-02T01:11:00.000Z
1196	1	TU-154	Moscow	Rostov	1900-01-01T04:00:00.000Z	1900-01-01T05:45:00.000Z
7771	5	Boeing	London	Singapore	1900-01-01T01:00:00.000Z	1900-01-01T11:00:00.000Z
7772	5	Boeing	Singapore	London	1900-01-01T12:00:00.000Z	1900-01-02T02:00:00.000Z
7773	5	Boeing	London	Singapore	1900-01-01T03:00:00.000Z	1900-01-01T13:00:00.000Z
7774	5	Boeing	Singapore	London	1900-01-01T14:00:00.000Z	1900-01-02T06:00:00.000Z
7775	5	Boeing	London	Singapore	1900-01-01T09:00:00.000Z	1900-01-01T20:00:00.000Z
7776	5	Boeing	Singapore	London	1900-01-01T18:00:00.000Z	1900-01-02T08:00:00.000Z
7777	5	Boeing	London	Singapore	1900-01-01T18:00:00.000Z	1900-01-02T06:00:00.000Z
7778	5	Boeing	Singapore	London	1900-01-01T22:00:00.000Z	1900-01-02T12:00:00.000Z
8881	5	Boeing	London	Paris	1900-01-01T03:00:00.000Z	1900-01-01T04:00:00.000Z
8882	5	Boeing	Paris	London	1900-01-01T22:00:00.000Z	1900-01-01T23:00:00.000Z

Ход выполнения практической работы

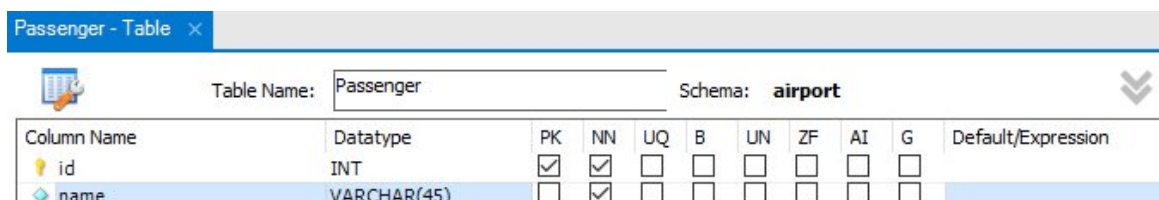
1. Создадим базу данных «airport» в программе-дизайнере MySQL Workbench.
2. В базе данных «airport» создадим таблицу «Company» с полями:
 - ☐ id тип INT – первичный ключ (PK);
 - ☐ name тип VARCHAR(45), не нулевое (NN);



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Рисунок 2. Таблица Company базы данных airport, вкладка Columns

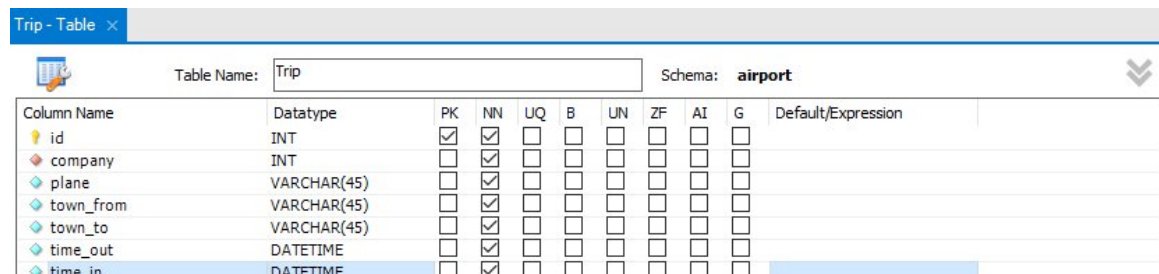
3. В базе данных «airport» создадим таблицу «Passenger» с полями:
 - ☐ id тип INT – первичный ключ (PK), не нулевое (NN);
 - ☐ name тип VARCHAR(45), не нулевое (NN).



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Рисунок 3. Таблица Passenger базы данных airport, вкладка Columns

4. В базе данных «airport» создадим таблицу «Trip» с полями:
 - ☐ id тип INT – первичный ключ (PK), не нулевое (NN);
 - ☐ company тип INT, не нулевое (NN);
 - ☐ plane тип VARCHAR(45), не нулевое (NN);
 - ☐ town_from тип VARCHAR(45), не нулевое (NN);
 - ☐ town_to тип VARCHAR(45), не нулевое (NN);
 - ☐ time_out тип DATETIME, не нулевое (NN);
 - ☐ time_in тип DATETIME, не нулевое (NN).



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
company	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
plane	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
town_from	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
town_to	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
time_out	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
time_in	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Рисунок 4. Таблица Trip базы данных airport, вкладка Columns

5. Для таблицы «Trip» создадим Foreign Key (вкладка Foreign Keys) с именем «company_id» на таблицу «airport.Company», колонку «id». В качестве события укажем «On Delete» со значением «CASCADE».

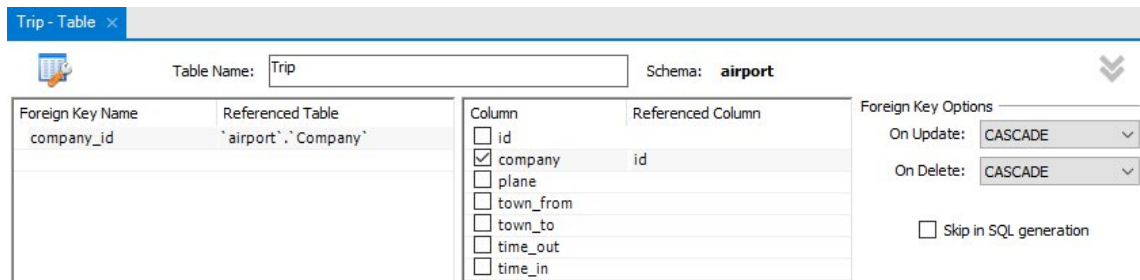


Рисунок 5. Создание внешнего ключа company_id, вкладка Foreign Keys

6. В базе данных «airport» создадим таблицу «Pass_in_trip» с полями:

- ☐ id тип INT – первичный ключ (PK), не нулевое (NN);
- ☐ trip тип INT, не нулевое (NN);
- ☐ passenger тип INT, не нулевое (NN);
- ☐ place тип VARCHAR(45).

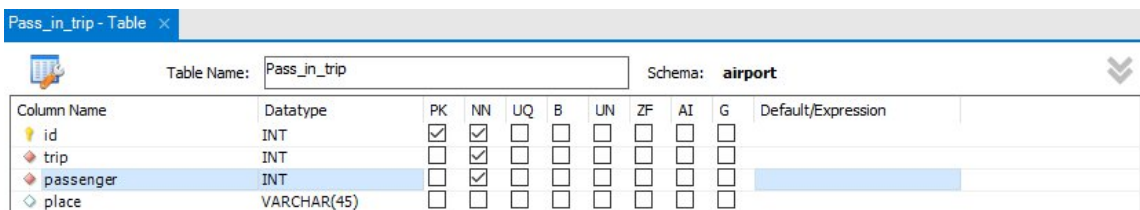


Рисунок 6. Таблица Pass_in_trip базы данных airport, вкладка Columns

7. Для таблицы «Pass_in_trip» создадим внешние ключи: Foreign Key с именем «trip_id» на таблицу «airport.Trip», колонку «id» и Foreign Key с именем «passenger_id» на таблицу «airport.Passenger», колонку «id».

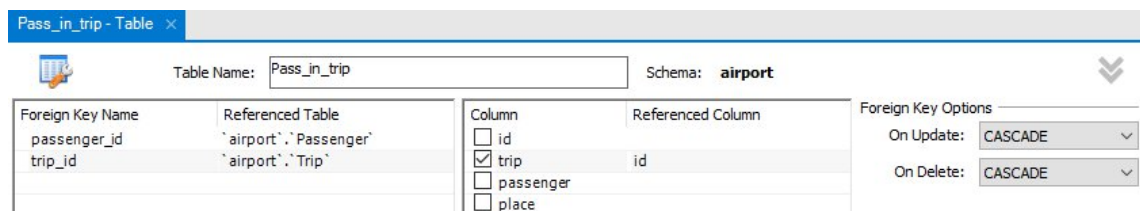


Рисунок 7. Создание внешних ключей trip_id и passenger_id, вкладка Foreign Keys

8. Сохраним созданную в программе-дизайнере схему базы данных на локальный компьютер. Подключимся к базе данных MySQL.

9. Активизируем базу данных «airport» (USE). В режиме выполнения SQL-запросов выполним следующие команды:

```

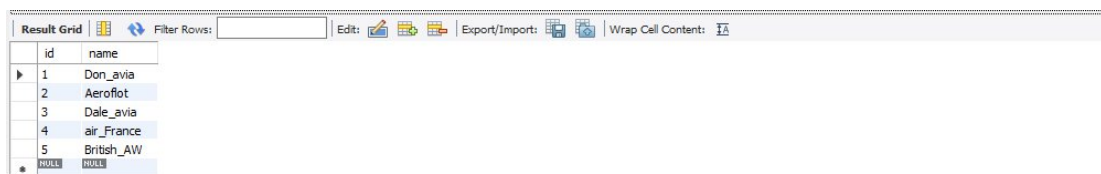
1 • USE airport;
2
3 -- Для Company
4 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/CompanyTable.csv'
5 INTO TABLE Company
6 FIELDS TERMINATED BY ','
7 LINES TERMINATED BY '\n'
8 IGNORE 1 ROWS;
9
10 -- Для Passenger
11 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/PassengerTable.csv'
12 INTO TABLE Passenger
13 FIELDS TERMINATED BY ','
14 LINES TERMINATED BY '\n'
15 IGNORE 1 ROWS;
16
17 -- Для Trip
18 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/TripTable.csv'
19 INTO TABLE Trip
20 FIELDS TERMINATED BY ','
21 LINES TERMINATED BY '\n'
22 IGNORE 1 ROWS
23 (id, company, plane, town_from, town_to, @time_out_var, @time_in_var) -- Временные переменные для хранения сырых значений
24 SET
25     time_out = STR_TO_DATE(@time_out_var, '%Y-%m-%dT%H:%i:%s.%fZ'), -- Преобразование формата из CSV
26     time_in = STR_TO_DATE(@time_in_var, '%Y-%m-%dT%H:%i:%s.%fZ');
27
28 -- Для Pass_in_trip
29 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/PassInTripTable.csv'
30 INTO TABLE Pass_in_trip
31 FIELDS TERMINATED BY ','
32 LINES TERMINATED BY '\n'
33 IGNORE 1 ROWS
34 (id, trip, passenger, place);

```

Рисунок 8. SQL-команды для импорта данных из CSV-файлов в БД. Редактор SQL

10. Проверим качество заполнения таблиц после импорта. Убедимся, что данные были корректно импортированы во все таблицы базы данных "airport". Выполним SQL команду: `SELECT * FROM table;`

1 • `SELECT * FROM airport.company;`



	id	name
1	1	Don_avia
2	2	Aeroflot
3	3	Dale_avia
4	4	air_France
5	5	British_AW

Рисунок 9. Проверка таблицы Company

2 • `SELECT * FROM Passenger;`



	id	name
1	1	Bruce Willis
2	2	George Clooney
3	3	Kevin Costner
4	4	Donald Sutherland
5	5	Jennifer Lopez

Рисунок 10. Проверка таблицы Passenger (на скриншоте только первые 5 записей)

3 • **SELECT * FROM Trip;**

	id	company	plane	town_from	town_to	time_out	time_in
▶	1100	4	Boeing	Rostov	Paris	1900-01-01 14:30:00	1900-01-01 17:50:00
	1101	4	Boeing	Paris	Rostov	1900-01-01 08:12:00	1900-01-01 11:45:00
	1123	3	TU-154	Rostov	Vladivostok	1900-01-01 16:20:00	1900-01-02 03:40:00
	1124	3	TU-154	Vladivostok	Rostov	1900-01-01 09:00:00	1900-01-01 19:50:00
	1145	2	IL-86	Moscow	Rostov	1900-01-01 09:35:00	1900-01-01 11:23:00
	1146	2	IL-86	Rostov	Moscow	1900-01-01 17:55:00	1900-01-01 20:01:00
	1181	1	TU-134	Rostov	Moscow	1900-01-01 06:12:00	1900-01-01 08:01:00

Рисунок 11. Проверка таблицы Trip (первые 7 записей)

4 • **SELECT * FROM Pass_in_trip;**

	id	trip	passenger	place
▶	1	1100	1	1a
	2	1123	3	2a
	3	1123	1	4c
	4	1123	6	4b
	5	1124	2	2d

Рисунок 12. Проверка таблицы Pass_in_trip (первые 5 записей)

11. Убедимся, что количество строк соответствует ожидаемому, для этого выполним запрос, который выведет количество записей из всех таблиц базы данных "airport" в одном результате:

```

1 • USE airport;
2 • SELECT
3     (SELECT COUNT(*) FROM Company) AS company_count,
4     (SELECT COUNT(*) FROM Passenger) AS passenger_count,
5     (SELECT COUNT(*) FROM Trip) AS trip_count,
6     (SELECT COUNT(*) FROM Pass_in_trip) AS pass_in_trip_count;

```

	company_count	passenger_count	trip_count	pass_in_trip_count
▶	5	32	22	32

Рисунок 13. Проверка количества записей в таблицах

12. Проверим целостность данных. Убедимся, что внешние ключи работают корректно. Для этого выполним следующий запрос:

```

1 • USE airport;
2 • SELECT p.name, c.name
3 FROM Pass_in_trip pit
4 JOIN Passenger p ON pit.passenger = p.id
5 JOIN Trip t ON pit.trip = t.id
6 JOIN Company c ON t.company = c.id;

```

Рисунок 14. SQL-запрос проверки связей между таблицами

Данный запрос отвечает на вопрос: Какие пассажиры летят рейсами каких авиакомпаний? Для этого запрос последовательно соединяет 4 таблицы:

❑ Pass_in_trip (посадки пассажиров, псевдоним pit) – центральная таблица

- ☐ Passenger (алиас p) – по полю passenger = id
- ☐ Trip (t) – по полю trip = id
- ☐ Company (c) – по полю company = id

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	name	name	
►	Bruce Willis	Don_avia	
	Ray Liotta	Don_avia	
	Nikole Kidman	Don_avia	
	Jennifer Lopez	Don_avia	
	Jennifer Lopez	Don_avia	
	Alan Rickman	Don_avia	
	Nikole Kidman	Don_avia	
	Kurt Russell	Don_avia	
	Nikole Kidman	Don_avia	
	Kevin Costner	Aeroflot	

Рисунок 15. Проверка целостности данных (первые 10 строк вывода)

13. Выполним SQL-запрос, который покажет все города, доступные для перелёта из Парижа, и длительность каждого перелёта (в формате HH:MM:SS):

В какие города можно улететь из Парижа (Paris) и сколько времени это займёт?

```

1 • USE airport;
2 • SELECT
3     town_to AS destination_city,
4     TIMEDIFF(time_in, time_out) AS flight_time
5 FROM Trip
6 WHERE town_from = 'Paris'
7 ORDER BY town_to; -- Сортировка результатов по алфавиту

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	destination_city	flight_time	
►	London	01:00:00	
	Rostov	03:33:00	

Рисунок 16. Результат выполнения запроса пункта 13

Выводы

В ходе выполнения практической работы была успешно проведена работа с базой данных «airport», включая создание таблиц, импорт данных и их анализ.

Таблицы созданы в соответствии с ERD-диаграммой, связи между ними настроены корректно. Данные импортированы из CSV-файлов без ошибок, что подтверждено проверкой количества записей (SELECT COUNT(*)) и проверкой целостности данных. Целостность данных проверена с помощью JOIN-запросов — все связи (пассажиры → рейсы → авиакомпании) работают корректно.

Практическая задача решена — определены города, в которые можно улететь из Парижа, и рассчитана длительность перелетов.

Навыки, полученные в ходе работы:

- ☐ Умение проектировать структуру БД на основе ERD-диаграммы.
- ☐ Навык загрузки данных из внешних источников (CSV) с обработкой форматов даты и времени.
- ☐ Опыт проверки качества данных после импорта.
- ☐ Составление SQL-запросов для анализа данных (фильтрация, соединение таблиц, расчет временных интервалов).

ПРАКТИЧЕСКАЯ РАБОТА №2

Освоение работы с временными таблицами в SQL. Применение сложных JOIN-соединений. Оптимизация подсчетов с использованием COUNT(DISTINCT). Визуализация связей между таблицами базы данных.

Задание практической работы

ERD – диаграмма используется, как и в предыдущей работе:

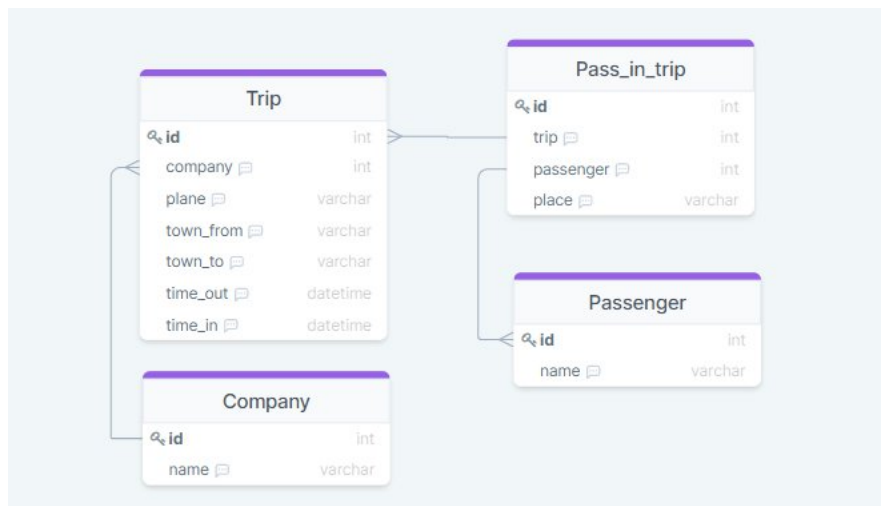


Рисунок 1. ERD-диаграмма БД задания практической работы

На основе данных, реализованных в практической работе №1, необходимо решить следующие задачи:

1. Выведите загруженность (число пассажиров) каждого рейса. Результат вывести в отсортированном виде по убыванию загруженности.
☐ Поля в результирующей таблице: trip, count.
2. Выведите имена всех пар пассажиров, летевших вместе на одном рейсе два или более раз, и количество таких совместных рейсов. В passengerName1 разместите имя пассажира с наименьшим идентификатором.
☐ Поля в результирующей таблице: passengerName1, passengerName2, count
3. Определить тех пассажиров из задания 2, которые были на рейсах из задания 1, если таковые имеются. Если есть вывести их имена.
☐ Поля в результирующей таблице: passengerNameOnTrip.

Ход выполнения практической работы

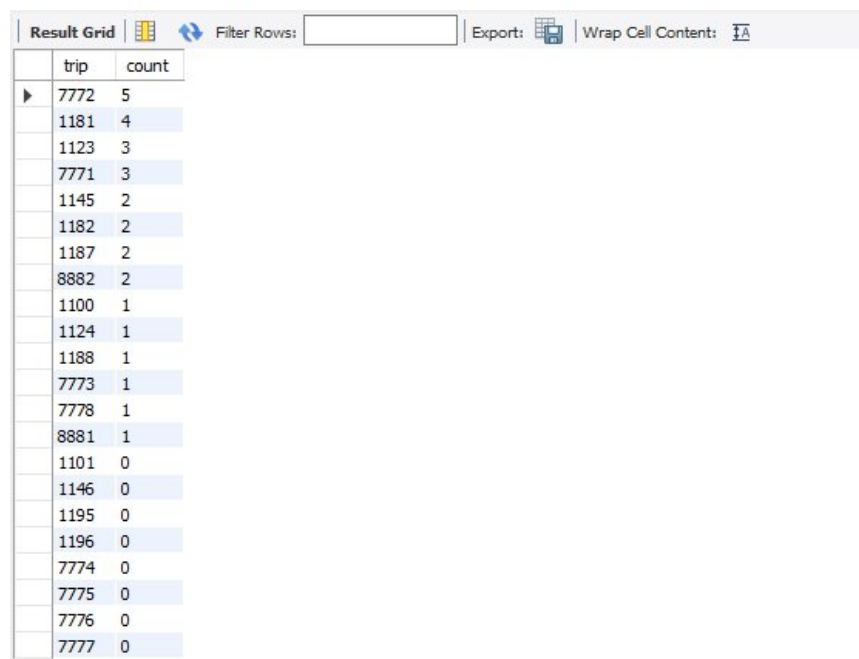
1. Для определения загруженности рейсов создадим SQL-запрос, который выведет все рейсы с указанием количества пассажиров на каждом, отсортированный по убыванию загруженности.

```
1  -- Выбираем номер рейса и количество пассажиров
2  • SELECT
3      t.id AS 'trip',
4      COUNT(DISTINCT pit.passenger) AS count -- Считаем пассажиров, а не их места
5      -- Используем LEFT JOIN, чтобы включить ВСЕ рейсы, даже без пассажиров
6  FROM Trip t LEFT JOIN Pass_in_trip pit
7  ON pit.trip = t.id -- Условие соединения: номер рейса в обеих таблицах
8  GROUP BY t.id      -- Группируем результаты по ID рейса
9  ORDER BY count DESC; -- Сортируем по убыванию
```

Рисунок 2. Запрос для анализа загруженности рейсов

В запросе используем LEFT JOIN для включения всех рейсов, даже без пассажиров. Это гарантирует, что все рейсы из таблицы Trip будут в результате, даже если для них нет записей в Pass_in_trip.

COUNT(DISTINCT pit.passenger) считает только реальных пассажиров, а не зарегистрированные на них места (на одного пассажира может быть куплено больше одного места), для рейсов без пассажиров автоматически выводится 0. ORDER BY ... DESC упорядочивает результаты по убыванию и позволяет сразу увидеть самые загруженные рейсы.



The screenshot shows a database interface with a 'Result Grid' tab. The grid displays the results of the SQL query, with columns 'trip' and 'count'. The results are sorted in descending order of passenger count. The first row shows trip 7772 with 5 passengers. Subsequent rows show trips with 4, 3, 2, and 1 passenger, followed by several trips with 0 passengers.

trip	count
7772	5
1181	4
1123	3
7771	3
1145	2
1182	2
1187	2
8882	2
1100	1
1124	1
1188	1
7773	1
7778	1
8881	1
1101	0
1146	0
1195	0
1196	0
7774	0
7775	0
7776	0
7777	0

Рисунок 3. Результат выполнения запроса о загруженности рейсов

2. Составим SQL-запрос на вывод имен всех пар пассажиров, летевших вместе на одном рейсе два и более раз, и количество таких совместных рейсов. В passengerName1 разместим имя пассажира с наименьшим идентификатором.

```

1  -- Пары пассажиров, летевших вместе 2+ раза
2  • SELECT
3      p1.name AS passengerName1,      -- Имя пассажира с меньшим ID
4      p2.name AS passengerName2,      -- Имя пассажира с большим ID
5      COUNT(DISTINCT pit1.trip) AS count -- Учитываем только уникальные рейсы!
6  FROM Pass_in_trip pit1             -- Таблица посадок на рейс
7  JOIN Pass_in_trip pit2             -- Соединяем с той же таблицей для нахождения пар
8      ON pit1.trip = pit2.trip        -- Условие: один и тот же рейс
9      AND pit1.passenger < pit2.passenger -- Исключаем дублирование (A+B и B+A или A=B) и сортируем по ID
10 JOIN Passenger p1 ON pit1.passenger = p1.id -- Получаем имя первого пассажира
11 JOIN Passenger p2 ON pit2.passenger = p2.id -- Получаем имя второго пассажира
12 GROUP BY passengerName1, passengerName2 -- Группируем по уникальным парам
13 HAVING count >= 2                   -- Фильтруем только пары с 2+ совместными рейсами
14 ORDER BY count DESC;               -- Сортируем по убыванию количества совместных рейсов

```

Рисунок 4. Запрос задания №2

Отметим, что запрос точно учитывает только уникальные совместные перелеты, игнорируя количество купленных мест на одного пассажира на каждом рейсе, так как таблица Pass_in_trip указывает и то, сколько мест закреплено за пассажиром, а он может купить больше одного места. Вот, например, пассажир с ID 11 купил три билета на рейс 7771, а пассажир 14 на тот же рейс купил два билета.

	id	trip	passenger	place
	23	7771	11	4a
	24	7771	11	1b
	25	7771	11	5a
	30	7771	14	4d
	31	7771	14	5d

Рисунок 5. Множественные места на рейс, таблица Pass_in_trip

Для решения проблемы используем функцию COUNT(DISTINCT pit1.trip), которая игнорирует множественные места одного пассажира на рейсе. Данная функция для каждой группы пассажиров (сформированной GROUP BY) берет все значения pit1.trip (номера рейсов) и удаляет из подсчета строки с повторяющимися номерами рейсов (DISTINCT) для одного и того же пассажира, затем подсчитывает количество оставшихся уникальных рейсов. Таким образом, в результате получаем:

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	passengerName1	passengerName2	count		
►	Michael Caine	Mullah Omar	3		
	Bruce Willis	Ray Liotta	2		

Рисунок 6. Результат выполнения запроса задания №2

3. Определить тех пассажиров из задания №2, которые были на рейсах из задания №1, если таковые имеются. Если есть вывести их имена.

Для решения данного задания будем использовать временные таблицы. Сначала создаем временные таблицы с промежуточными результатами, а затем выполняем финальный запрос, использующий обе таблицы. Также выведем содержимое временных таблиц для проверки на корректность выполнения запроса.

```
1 -- 1. Создаем временную таблицу для непустых рейсов (задание №1)
2 • CREATE TEMPORARY TABLE temp_flights AS
3   SELECT trip, COUNT(DISTINCT pit.passenger) AS count
4   FROM Pass_in_trip pit
5   GROUP BY trip
6   ORDER BY count DESC;
7
8 • SELECT * FROM temp_flights;
```

Рисунок 7. Создание временной таблицы задания №1



	trip	count
▶	7772	5
	1181	4
	1123	3
	7771	3
	1145	2
	1182	2
	1187	2
	8882	2
	1100	1
	1124	1
	1188	1
	7773	1
	7778	1
	8881	1

Рисунок 8. Проверка корректности временной таблицы задания №1

```
10 -- 2. Создаем временную таблицу для пар пассажиров (задание №2)
11 • CREATE TEMPORARY TABLE temp_pairs AS
12   SELECT
13     pit1.passenger AS id1, p1.name AS name1,
14     pit2.passenger AS id2, p2.name AS name2,
15     COUNT(DISTINCT pit1.trip) AS count
16   FROM Pass_in_trip pit1 JOIN Pass_in_trip pit2
17   ON pit1.trip = pit2.trip AND pit1.passenger < pit2.passenger
18   JOIN Passenger p1 ON pit1.passenger = p1.id
19   JOIN Passenger p2 ON pit2.passenger = p2.id
20   GROUP BY id1, name1, id2, name2
21   HAVING count >= 2
22   ORDER BY count DESC;
23
24 • SELECT * FROM temp_pairs;
```

Рисунок 9. Создание временной таблицы задания №2

Result Grid					
Filter Rows:					
Export: Wrap Cell Content:					
	id1	name1	id2	name2	count
▶	14	Michael Caine	37	Mullah Omar	3
	1	Bruce Willis	6	Ray Liotta	2

Рисунок 10. Проверка корректности временной таблицы задания №2

```

26 -- Основной запрос: находим имена пассажиров, удовлетворяющих обоим условиям
27 • SELECT DISTINCT p.name AS passengerNameOnTrip -- Удаляем дубликаты имен (если пассажир встречается в нескольких парах)
28 FROM Pass_in_trip pit
29 JOIN temp_flights tf ON pit.trip = tf.trip -- Соединяем с временной таблицей
30 JOIN temp_pairs tp ON (pit.passenger = tp.id1 OR pit.passenger = tp.id2)
31 JOIN Passenger p ON pit.passenger = p.id;
32
33 -- 4. Очищаем временные таблицы (необязательно, удалятся при закрытии сессии)
34 • DROP TEMPORARY TABLE temp_flights;
35 • DROP TEMPORARY TABLE temp_pairs;

```

Рисунок 11. Основной запрос для решения задания №3 и удаление временных таблиц (необязательно)

Result Grid	
Filter Rows:	
Export: Wrap Cell Content:	
	passengerNameOnTrip
▶	Michael Caine
	Mullah Omar
	Ray Liotta
	Bruce Willis

Рисунок 12. Результат выполнения задания №3

Выводы

В ходе выполнения практической работы были успешно решены все три задачи:

1. Создана временная таблица `temp_flights` с загруженными рейсами.
2. Построена таблица `temp_pairs` с часто летающими парами пассажиров.
3. Найдены пассажиры, удовлетворяющие обоим условиям, с выводом их имен.

Представленные решения корректно игнорируют дубликаты (множественные места на рейсе, повторение имен в парах пассажиров). Запросы оптимизированы через временные таблицы.

В результате практической работы были освоены следующие навыки. Работа с временными таблицами (`CREATE TEMPORARY TABLE`). Использование сложных `JOIN` с условиями `OR`. Подсчет уникальных значений через `COUNT(DISTINCT)`. Фильтрация групп с `HAVING`. Отладка промежуточных результатов (проверка временных таблиц).

Листинг SQL-запросов

```
-- Выбираем номер рейса и количество пассажиров
SELECT t.id AS 'trip',
       COUNT(DISTINCT pit.passenger) AS COUNT -- Считаем пассажиров, а не их места
-- Используем LEFT JOIN, чтобы включить ВСЕ рейсы, даже без пассажиров
FROM Trip t
LEFT JOIN Pass_in_trip pit ON pit.trip = t.id -- Условие соединения: номер рейса в
обеих таблицах
GROUP BY t.id -- Группируем результаты по ID рейса
ORDER BY COUNT DESC; -- Сортируем по убыванию

-- Пары пассажиров, летевших вместе 2+ раза
SELECT p1.name AS passengerName1, -- Имя пассажира с меньшим ID
       p2.name AS passengerName2, -- Имя пассажира с большим ID
       COUNT(DISTINCT pit1.trip) AS COUNT -- Учитываем только уникальные рейсы!
FROM Pass_in_trip pit1 -- Таблица посадок на рейс
JOIN Pass_in_trip pit2 -- Соединяем с той же таблицей для нахождения пар
ON pit1.trip = pit2.trip -- Условие: один и тот же рейс

AND pit1.passenger < pit2.passenger -- Исключаем дублирование (A+B и B+A или A=B) и
сортируем по ID
JOIN Passenger p1 ON pit1.passenger = p1.id -- Получаем имя первого пассажира
JOIN Passenger p2 ON pit2.passenger = p2.id -- Получаем имя второго пассажира
GROUP BY passengerName1, passengerName2 -- Группируем по уникальным парам
HAVING COUNT >= 2 -- Фильтруем только пары с 2+ совместными рейсами
ORDER BY COUNT DESC; -- Сортируем по убыванию количества совместных рейсов

-- 1. Создаем временную таблицу для непустых рейсов (задание №1)
CREATE
TEMPORARY TABLE temp_flights AS
SELECT trip,
       COUNT(DISTINCT pit.passenger) AS COUNT
FROM Pass_in_trip pit
GROUP BY trip
ORDER BY COUNT DESC;

SELECT * FROM temp_flights;

-- 2. Создаем временную таблицу для пар пассажиров (задание №2)
CREATE
TEMPORARY TABLE temp_pairs AS
SELECT pit1.passenger AS id1, p1.name AS name1,
       pit2.passenger AS id2, p2.name AS name2,
       COUNT(DISTINCT pit1.trip) AS COUNT
FROM Pass_in_trip pit1
JOIN Pass_in_trip pit2 ON pit1.trip = pit2.trip
AND pit1.passenger < pit2.passenger
JOIN Passenger p1 ON pit1.passenger = p1.id
JOIN Passenger p2 ON pit2.passenger = p2.id
GROUP BY id1, name1, id2, name2
HAVING COUNT >= 2
ORDER BY COUNT DESC;

SELECT * FROM temp_pairs;

-- Основной запрос: находим имена пассажиров, удовлетворяющих обоим условиям
SELECT DISTINCT p.name AS passengerNameOnTrip -- Удаляем дубликаты имен (если
пассажир встречается в нескольких парах)
FROM Pass_in_trip pit
JOIN temp_flights tf ON pit.trip = tf.trip -- Соединяем с временной таблицей
JOIN temp_pairs tp ON (pit.passenger = tp.id1 OR pit.passenger = tp.id2)
JOIN Passenger p ON pit.passenger = p.id;
-- 4. Очищаем временные таблицы (необязательно, удалятся при закрытии сессии)

DROPTEMPORARY TABLE temp_flights;
DROPTEMPORARY TABLE temp_pairs;
```

ПРАКТИЧЕСКАЯ РАБОТА №3

Создание процедуры для поиска авиакомпаний с минимальным количеством рейсов. Реализация механизма отката изменений через параметр `should_rollback`. Использование временных таблиц для сохранения промежуточных результатов.

Задание практической работы

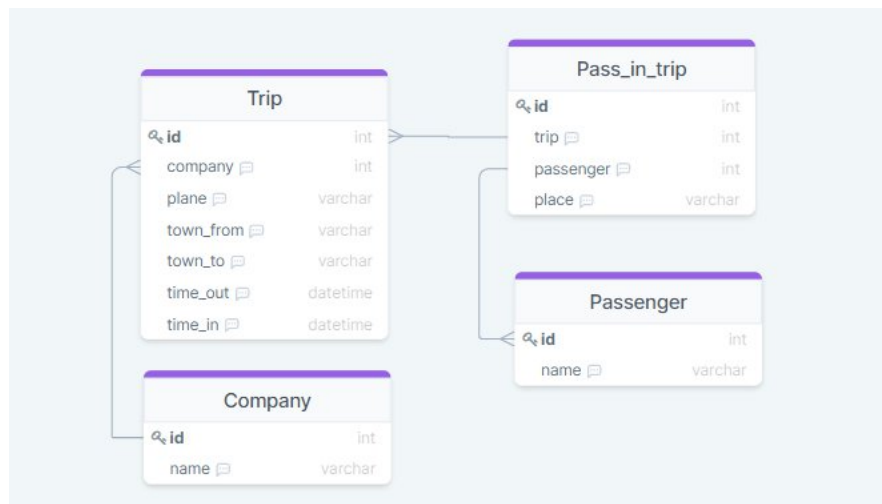


Рисунок 1. ERD-диаграмма БД задания практической работы

ERD-диаграмма используется, как и в предыдущей работе. На основе введенных ранее данных необходимо решить следующие задачи:

1. Создать хранимую процедуру, функционал которой должен возвращать массив компаний, совершивших наименьшее количество рейсов, при этом должна быть учтена возможность вернуть БД на состояние ДО запуска процедуры.
2. Вывести отсортированный по количеству перелетов (по убыванию) и имени (по возрастанию) список пассажиров, совершивших хотя бы 1 полет.
 - a. Поля в результирующей таблице: name, count
 - b. Используйте конструкцию "as count" для агрегатной функции подсчета количества перелетов. Это необходимо для корректной проверки.
3. Откорректировать запрос из задания №2, добавив в него условие: Вывести отсортированный по количеству перелетов (по убыванию) и имени (по возрастанию) список пассажиров, совершивших хотя бы 1 полет той компанией, полученной из задания №1.

Ход выполнения практической работы

1. Создадим хранимую процедуру `GetCompaniesWithMinFlights`, которая возвращает компании с наименьшим количеством рейсов и поддерживает откат изменений.

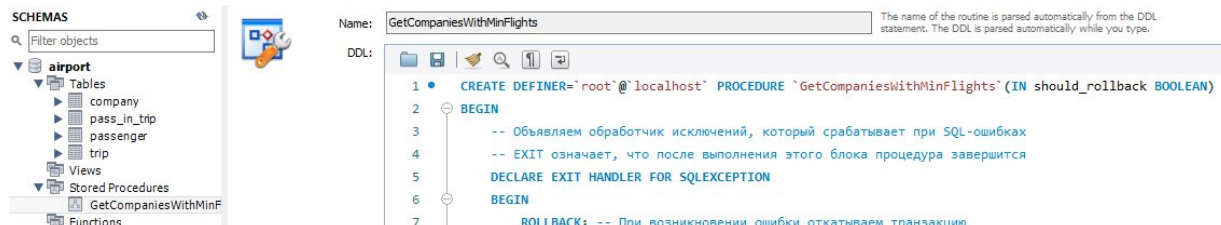


Рисунок 2. Окно редактирования хранимой процедуры (Stored Procedure) в MySQL Workbench

Процедура `GetCompaniesWithMinFlights` выполняет выборку авиакомпаний с минимальным количеством рейсов. Входной параметр `should_rollback` типа `BOOLEAN` управляет поведением транзакции (`TRUE` – откатываем транзакцию, `FALSE` – подтверждаем изменения). Начало работы процедуры сопровождается изменением разделителя `DELIMITER` на `//` для корректной обработки точек с запятой в теле процедуры.

Сначала объявляется обработчик исключений типа `SQLEXCEPTION` (Рисунок 3), который при возникновении ошибок выполняет откат транзакции через `ROLLBACK` и передает ошибку клиенту с помощью `RESIGNAL`.

```
1  -- Изменяем стандартный разделитель с ";" на "//" для корректной обработки ";" внутри процедуры
2  DELIMITER //
3
4  • -- Создаем новую хранимую процедуру с именем GetCompaniesWithMinFlights
5  -- Процедура принимает один входной параметр should_rollback типа BOOLEAN
6  CREATE PROCEDURE GetCompaniesWithMinFlights(IN should_rollback BOOLEAN)
7  BEGIN
8      -- Объявляем обработчик исключений, который срабатывает при SQL-ошибках
9      -- EXIT означает, что после выполнения обработчика процедура завершится
10     DECLARE EXIT HANDLER FOR SQLEXCEPTION
11     BEGIN ROLLBACK; -- При возникновении ошибки откатываем текущую транзакцию
12     RESIGNAL; -- Повторно генерируем ошибку для передачи клиенту
13     END;
```

Рисунок 3. Создание хранимой процедуры с обработкой транзакций и исключений в MySQL

Процедура начинает транзакцию командой `START TRANSACTION` (Рисунок 4), что позволяет при необходимости отменить все изменения. Далее создается временная таблица `temp_results` с полями `company_id` и `company_name` для хранения промежуточных результатов.

```

14
15      -- Начинаем новую транзакцию (все изменения можно будет откатить)
16      START TRANSACTION;
17      -- Удаляем временную таблицу, если она существует
18      -- IF EXISTS предотвращает ошибку, если таблица не существует
19      DROP TEMPORARY TABLE IF EXISTS temp_results;
20
21      -- Создаем новую временную таблицу для хранения результатов
22      -- Таблица будет автоматически удалена при завершении сессии
23      CREATE TEMPORARY TABLE temp_results (
24          company_id INT,          -- Колонка для ID компании (целое число)
25          company_name VARCHAR(45) -- Колонка для названия компании (строка до 45 символов)
26      );

```

Рисунок 4. Работа с временными таблицами (создание temp_results) в транзакции хранимой процедуры MySQL

Выполняем вставку данных во временную таблицу temp_results (Рисунок 5). Использование временной таблицы позволяет сохранить промежуточные данные, а LEFT JOIN гарантирует, что в выборку попадут все компании, даже те, у которых рейсы отсутствуют.

```

27
28      -- Вставляем данные во временную таблицу
29      INSERT INTO temp_results
30      -- Основной запрос для выбора компаний с минимальным количеством рейсов
31      SELECT c.id AS company_id, c.name AS company_name
32      FROM company c LEFT JOIN trip t ON c.id = t.company -- LEFT JOIN включает компании без рейсов
33      GROUP BY c.id, c.name -- Группируем по ID и названию компании
34      HAVING -- Фильтруем группы после агрегации
35          COUNT(t.id) = ( -- Оставляем только компании с минимальным числом рейсов
36              -- Подзапрос для нахождения минимального количества рейсов среди всех компаний
37              SELECT MIN(flight_count)
38              FROM (
39                  -- Подзапрос, считающий количество рейсов для каждой компании
40                  SELECT COUNT(t2.id) AS flight_count
41                  FROM company c2
42                  LEFT JOIN trip t2 ON c2.id = t2.company
43                  GROUP BY c2.id
44              ) AS counts
45          );
46
47      -- Возвращаем клиенту все записи из временной таблицы
48      SELECT * FROM temp_results;

```

Рисунок 5. Заполнение временной таблицы и выбор компаний с минимальным числом рейсов

Фильтрация с помощью HAVING оставляет только те компании, у которых количество рейсов (COUNT(t.id)) равно минимальному значению, найденному во вложенном подзапросе (SELECT MIN(flight_count) FROM...). Этот подзапрос сначала вычисляет количество рейсов для каждой компании (COUNT(t2.id)), а затем определяет минимальное значение (MIN(flight_count)).

После заполнения временной таблицы результаты возвращаются клиенту командой `SELECT * FROM temp_results` (Рисунок 5).

В завершении процедура возвращает клиенту все записи из временной таблицы (Рисунок 6).

В зависимости от параметра `should_rollback` выполняется либо подтверждение изменений (`COMMIT`), либо их откат (`ROLLBACK`). Временная таблица удаляется для освобождения ресурсов. Стандартный разделитель `;` восстанавливается командой `DELIMITER`.

```
46
47      -- Возвращаем клиенту все записи из временной таблицы
48      SELECT * FROM temp_results;
49
50      -- Проверяем параметр should_rollback
51      IF should_rollback THEN ROLLBACK; -- Если TRUE - откатываем транзакцию (все изменения отменяются)
52      ELSE COMMIT; -- Если FALSE - подтверждаем транзакцию (изменения сохраняются)
53      END IF;
54
55      -- Удаляем временную таблицу (освобождаем ресурсы)
56      DROP TEMPORARY TABLE IF EXISTS temp_results;
57  END //
58
59  DELIMITER ; -- Восстанавливаем стандартный разделитель команд
```

Рисунок 6. Возврат результатов из временной таблицы. Завершение транзакции и очистка ресурсов в хранимой процедуре MySQL

На Рисунке 7 показаны два режима работы процедуры `GetCompaniesWithMinFlights`, различающиеся только параметром управления транзакцией:

```
1 • CALL GetCompaniesWithMinFlights(FALSE); -- Без отката (сохраняем изменения)
2
3 • CALL GetCompaniesWithMinFlights(TRUE); -- С откатом (возвращаем БД в исходное состояние)
```

Рисунок 7. Варианты вызова хранимой процедуры `GetCompaniesWithMinFlights` с разными режимами транзакции

Хотя сейчас процедура не изменяет постоянные данные (работает в `read-only` режиме), откат сохраняет важную функциональность.

Режим `COMMIT (FALSE)` просто возвращает результат выборки.

Режим `ROLLBACK (TRUE)` потенциально обеспечивает: безопасное тестирование на реальных данных, отладку — выполнение процедуры без побочных эффектов, и готовность к интеграции процедуры как части более сложной транзакции. Такой подход обеспечивает гибкость при развитии функционала.

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	company_id	company_name		
▶	2	Aeroflot		
	3	Dale_avia		
	4	air_France		

Рисунок 8. Результат выполнения хранимой процедуры: список компаний с минимальным количеством рейсов

2. Вывести отсортированный по количеству перелетов (по убыванию) и имени (по возрастанию) список пассажиров, совершивших хотя бы 1 полет.

Отметим, что, как и в практической работе №2, запрос должен учитывать только уникальные перелеты, а не количество купленных мест на пассажира, так как таблица Pass_in_trip указывает и то, сколько мест закреплено за пассажиром, а он может купить больше одного места. Вот, например, пассажир с ID 11 купил три билета на рейс 7771, а пассажир 14 на тот же рейс купил два билета.

	id	trip	passenger	place
	23	7771	11	4a
	24	7771	11	1b
	25	7771	11	5a
	30	7771	14	4d
	31	7771	14	5d

Рисунок 9. Множественные места на рейс, таблица Pass_in_trip

Поэтому будем использовать COUNT(DISTINCT pit.trip) вместо COUNT(pit.trip), который указывал бы именно количество мест без учета рейсов. Теперь считаются только уникальные номера рейсов для каждого пассажира.

```

1 • SELECT
2     p.name AS name,
3     COUNT(DISTINCT pit.trip) AS count -- Учитываем только уникальные рейсы!
4 FROM passenger p JOIN pass_in_trip pit
5 ON p.id = pit.passenger
6 GROUP BY p.id, p.name
7 HAVING count >= 1 -- Проверяем уникальные рейсы
8 ORDER BY
9     count DESC, -- Сортировка по количеству перелетов (по убыванию)
10    name ASC;   -- Сортировка имен (алфавитный порядок)

```

Рисунок 10. SQL-запрос для подсчёта количества перелётов пассажиров

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	name	count			
►	Mullah Omar	4			
	Bruce Willis	3			
	Jennifer Lopez	3			
	Kurt Russell	3			
	Michael Caine	3			
	Nikole Kidman	3			
	Kevin Costner	2			
	Ray Liotta	2			
	Steve Martin	2			
	Alan Rickman	1			
	George Clooney	1			
	Harrison Ford	1			
	Russell Crowe	1			

Рисунок 11. Результат выполнения запроса для подсчёта количества перелётов пассажиров

3. Откорректировать запрос из задания №2, добавив в него условие: Вывести отсортированный по количеству перелетов (по убыванию) и имени (по возрастанию) список пассажиров, совершивших хотя бы 1 полет той компанией, полученной из задания №1.

Для начала откорректируем тело хранимой процедуры из задания №1.

```

1 • CREATE DEFINER='root'@'localhost' PROCEDURE `GetCompaniesWithMinFlights`(IN should_rollback BOOLEAN)
2 BEGIN
3     DECLARE EXIT HANDLER FOR SQLEXCEPTION
4     BEGIN
5
6
7
8
9     START TRANSACTION;
10
11     -- Создаем временную таблицу с уникальным именем, которая сохранится после процедуры
12     DROP TEMPORARY TABLE IF EXISTS global_min_companies;
13     CREATE TEMPORARY TABLE global_min_companies (
14
15
16
17
18     -- Заполняем таблицу данными
19     INSERT INTO global_min_companies
20     SELECT c.id, c.name
21     FROM company c
22     LEFT JOIN trip t ON c.id = t.company
23     GROUP BY c.id, c.name
24     HAVING COUNT(t.id) = (
25
26
27
28
29
30
31
32
33
34     -- Возвращаем результаты
35     SELECT * FROM global_min_companies;
36
37     -- Управление транзакцией
38     IF should_rollback THEN
39
40
41
42
43
44
45     -- ЯВНО НЕ УДАЛЯЕМ ТАБЛИЦУ, чтобы она была доступна после процедуры
46     -- Она автоматически удалится при завершении сессии
47 END

```

Рисунок 12. Изменение процедуры GetCompaniesWithMinFlights. Комментарии явно отражают суть изменений

Изменения были внесены для того, чтобы временная таблица temp_results (имя изменено на global_min_companies) была доступна и после завершения транзакции, для этого уберем инструкцию по удалению таблицы (DROP TEMPORARY TABLE IF EXISTS temp_results;) в конце процедуры, а также переименуем таблицу temp_results на global_min_companies, в нашем случае префикс «global_» становится маркером для таблиц, которые сознательно оставлены доступными и после завершения выполнения процедуры.

```

1 • CALL GetCompaniesWithMinFlights(FALSE); -- Вызываем процедуру
2
3   -- Основной запрос с использованием временной таблицы
4 • SELECT
5     p.name AS name,
6     COUNT(DISTINCT pit.trip) AS count
7   FROM passenger p
8   JOIN pass_in_trip pit ON p.id = pit.passenger
9   JOIN trip tr ON pit.trip = tr.id
10  JOIN global_min_companies gmin ON tr.company = gmin.company_id
11  GROUP BY p.id, p.name
12  HAVING COUNT(DISTINCT pit.trip) >= 1
13  ORDER BY count DESC, name ASC;

```

Рисунок 13. Откорректированный SQL-запрос для подсчёта количества перелётов пассажиров рейсами компаний из первого задания

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	name	count	
►	Bruce Willis	2	
	Kevin Costner	2	
	George Clooney	1	
	Jennifer Lopez	1	
	Ray Liotta	1	

Рисунок 14. Результат выполнения запроса задания №3

Убедимся, что пассажиры летали именно рейсами тех компаний, которые были получены в результате выполнения хранимой процедуры GetCompaniesWithMinFlights из задания №1. Для этого выполним проверочный запрос и выведем также названия компаний, рейсами которых летали пассажиры.

```

1 • CALL GetCompaniesWithMinFlights(FALSE); -- Вызываем процедуру
2
3 -- Основной запрос с использованием временной таблицы
4 • SELECT
5     p.name AS name,
6     COUNT(DISTINCT pit.trip) AS count,
7     gmin.company_name AS company_name
8 FROM passenger p
9 JOIN pass_in_trip pit ON p.id = pit.passenger
10 JOIN trip tr ON pit.trip = tr.id
11 JOIN global_min_companies gmin ON tr.company = gmin.company_id
12 GROUP BY p.id, p.name, gmin.company_name
13 HAVING COUNT(DISTINCT pit.trip) >= 1
14 ORDER BY count DESC, name ASC;

```

Рисунок 15. SQL-запрос – проверка корректности полученного результата (Рисунок 14)

Result Grid Filter Rows: Export: Wrap Cell Content:			
	name	count	company_name
►	Bruce Willis	1	air_France
	Bruce Willis	1	Dale_avia
	George Clooney	1	Dale_avia
	Jennifer Lopez	1	Aeroflot
	Kevin Costner	1	Aeroflot
	Kevin Costner	1	Dale_avia
	Ray Liotta	1	Dale_avia

Рисунок 16. Результат выполнения тестового запроса

Как видно на Рисунке 16, в результатах присутствуют только компании с минимальным количеством рейсов (air_France, Dale_avia, Aeroflot), что соответствует заданию №1.

Каждый пассажир имеет ровно 1 перелёт (count = 1) у указанных авиакомпаний, что подтверждает правильность использования COUNT(DISTINCT pit.trip).

Количество рейсов, совершенных одним и тем же пассажиром, совпадают. Задача решена верно.

Выводы

В результате выполнения практической работы была успешно создана хранимая процедура `GetCompaniesWithMinFlights`, возвращающая авиакомпании с минимальным числом рейсов, с поддержкой отката транзакций (принимаемый параметр `should_rollback`). Были использованы временные таблицы для сохранения промежуточных результатов (`temp_results/global_min_companies`).

Реализован запрос (Рисунок 13), выводящий пассажиров с сортировкой по количеству перелётов и именам пассажиров (`ORDER BY count DESC, name ASC`). Результат выполнения SQL-запроса задания №3 представлен на Рисунке 14.

Проверка данных полученного в результате выполнения SQL-запроса из задания №3 подтвердила корректность его работы (Рисунок 16). В выборку попали только целевые компании (`air_France`, `Dale_avia`, `Aeroflot`) из задания №1. Количество перелётов пассажиров подсчитано верно (без дублирования из-за нескольких мест на рейс), этому подтверждение столбец `count` (Рисунок 16).

В ходе выполнения данной практической работы были успешно закреплены и расширены ключевые навыки работы с базами данных, включая:

- ☐ Разработку сложных SQL-запросов с агрегатными функциями и условиями фильтрации (функции `COUNT()` и `MIN()` в сочетании с `GROUP BY` для групповой обработки данных)
- ☐ Создание и оптимизацию хранимых процедур в MySQL
- ☐ Управление транзакциями с реализацией механизма отката изменений (`ROLLBACK/COMMIT`)
- ☐ Использование временных таблиц для обработки промежуточных результатов
- ☐ Проверку выходных данных на соответствие требованиям задания

Все задачи выполнены в соответствии с требованиями, что подтверждается корректными результатами выполнения финальных запросов.

Листинг SQL-запросов

Хранимая процедура GetCompaniesWithMinFlights (задание №1)

```
CREATE DEFINER='root'@'localhost' PROCEDURE `GetCompaniesWithMinFlights`(IN should_rollback BOOLEAN)
BEGIN
    -- Объявляем обработчик исключений, который срабатывает при SQL-ошибках
    -- EXIT означает, что после выполнения обработчика процедура завершится
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN ROLLBACK; -- При возникновении ошибки откатываем текущую транзакцию
        RESIGNAL; -- Повторно генерируем ошибку для передачи клиенту
    END;

    -- Начинаем новую транзакцию (все изменения можно будет откатить)
    START TRANSACTION;

    -- Удаляем временную таблицу, если она существует
    -- IF EXISTS предотвращает ошибку, если таблица не существует
    DROP TEMPORARY TABLE IF EXISTS temp_results;

    -- Создаем новую временную таблицу для хранения результатов
    -- Таблица будет автоматически удалена при завершении сессии
    CREATE TEMPORARY TABLE temp_results (
        company_id INT, -- Колонка для ID компании (целое число)
        company_name VARCHAR(45) -- Колонка для названия компании (строка до 45 символов)
    );

    -- Вставляем данные во временную таблицу
    INSERT INTO temp_results
    -- Основной запрос для выбора компаний с минимальным количеством рейсов
    SELECT c.id AS company_id, c.name AS company_name
    FROM company c LEFT JOIN trip t ON c.id = t.company -- LEFT JOIN включает компании без рейсов
    GROUP BY c.id, c.name -- Группируем по ID и названию компании
    HAVING -- Фильтруем группы после агрегации
        COUNT(t.id) = ( -- Оставляем только компании с минимальным числом рейсов
            -- Подзапрос для нахождения минимального количества рейсов среди всех компаний
            SELECT MIN(flight_count)
            FROM (
                -- Подзапрос, считающий количество рейсов для каждой компании
                SELECT COUNT(t2.id) AS flight_count
                FROM company c2
                LEFT JOIN trip t2 ON c2.id = t2.company
                GROUP BY c2.id
            ) AS counts
        );

    -- Возвращаем клиенту все записи из временной таблицы
    SELECT * FROM temp_results;

    -- Проверяем параметр should_rollback
    IF should_rollback THEN ROLLBACK; -- Если TRUE - откатываем транзакцию (все изменения отменяются)
    ELSE COMMIT; -- Если FALSE - подтверждаем транзакцию (изменения сохраняются)
    END IF;

    -- Удаляем временную таблицу (освобождаем ресурсы)
    DROP TEMPORARY TABLE IF EXISTS temp_results;
END
```

SQL-запрос подсчёта количества перелётов пассажиров (задание №2)

```
SELECT
    p.name AS name,
    COUNT(DISTINCT pit.trip) AS count -- Учитываем только уникальные рейсы!
FROM passenger p JOIN pass_in_trip pit
ON p.id = pit.passenger
```

```
GROUP BY p.id, p.name
HAVING count >= 1 -- Проверяем уникальные рейсы
ORDER BY
    count DESC, -- Сортировка по количеству перелетов (по убыванию)
    name ASC; -- Сортировка имен (алфавитный порядок)
```

Отредактированная хранимая процедура (задание №3)

```
CREATE DEFINER='root'@'localhost' PROCEDURE `GetCompaniesWithMinFlights`(IN should_rollback BOOLEAN)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;

    -- Создаем временную таблицу с уникальным именем, которая сохранится после процедуры
    DROP TEMPORARY TABLE IF EXISTS global_min_companies;
    CREATE TEMPORARY TABLE global_min_companies (
        company_id INT,
        company_name VARCHAR(45)
    );

    -- Заполняем таблицу данными
    INSERT INTO global_min_companies
    SELECT c.id, c.name
    FROM company c LEFT JOIN trip t ON c.id = t.company
    GROUP BY c.id, c.name
    HAVING COUNT(t.id) = (
        SELECT MIN(flight_count)
        FROM (
            SELECT COUNT(t2.id) AS flight_count
            FROM company c2 LEFT JOIN trip t2
            ON c2.id = t2.company
            GROUP BY c2.id
        ) AS counts
    );

    -- Возвращаем результаты
    SELECT * FROM global_min_companies;

    -- Управление транзакцией
    IF should_rollback THEN
        ROLLBACK;
        -- Даже при откате оставляем таблицу, но очищаем её
        TRUNCATE TABLE global_min_companies;
    ELSE COMMIT;
    END IF;

    -- ЯВНО НЕ УДАЛЯЕМ ТАБЛИЦУ, чтобы она была доступна после процедуры
    -- Она автоматически удалится при завершении сессии
END
```

SQL-запрос подсчёта количества перелётов пассажиров (задание №3) с использованием данных временной таблицы global_min_companies (старое название temp_results)

```
CALL GetCompaniesWithMinFlights(FALSE); -- Вызываем процедуру

-- Основной запрос с использованием временной таблицы
SELECT
```

```
p.name AS name,  
COUNT(DISTINCT pit.trip) AS count  
FROM passenger p  
JOIN pass_in_trip pit ON p.id = pit.passenger  
JOIN trip tr ON pit.trip = tr.id  
JOIN global_min_companies gmin ON tr.company = gmin.company_id  
GROUP BY p.id, p.name  
HAVING COUNT(DISTINCT pit.trip) >= 1  
ORDER BY count DESC, name ASC;
```