

Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**ОТЧЕТ**  
**по лабораторной работе**

**«Язык SQL-DML»**

Базы данных

**Работу выполнил студент**

группа 43501/3      Дьячков В.В.

**Работу принял преподаватель**

\_\_\_\_\_ Мясов А.В.

Санкт-Петербург

2018

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Программа работы</b>	<b>3</b>
<b>3</b>	<b>Стандартные запросы</b>	<b>3</b>
3.1	Выборка с использованием логических операций . . . . .	3
3.2	Запрос с вычисляемым полем . . . . .	4
3.3	Выборка с использованием сортировки . . . . .	5
3.4	Запрос с вычислением совокупных характеристик таблиц . . . . .	5
3.5	Выборка из связанных таблиц . . . . .	6
3.6	Запрос с использованием группировки . . . . .	7
3.7	Вложенный запрос . . . . .	7
3.8	Вставка записей . . . . .	8
3.9	Изменение записей . . . . .	9
3.10	Удаление записей по условию . . . . .	9
3.11	Удаление с использованием вложенного запроса . . . . .	10
<b>4</b>	<b>Запросы в соответствии с заданием преподавателя</b>	<b>11</b>
4.1	Рейтинг городов по кварталам . . . . .	11
4.2	Клиенты, имеющие наибольший средний рост стоимости путевки	13
<b>5</b>	<b>Представления и хранимые процедуры</b>	<b>15</b>
<b>6</b>	<b>Выводы</b>	<b>19</b>

# 1. Цель работы

Познакомиться с языком создания запросов управления данными SQL-DML.

## 2. Программа работы

1. Изучение SQL-DML.
2. Выполнение всех запросов из списка стандартных запросов. Демонстрация результатов преподавателю.
3. Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием. Демонстрация результатов преподавателю.
4. Сохранение в БД выполненных запросов **SELECT** в виде представлений, запросов **INSERT**, **UPDATE** или **DELETE** – в виде ХП. Выкладывание скрипта в GitLab.

## 3. Стандартные запросы

### 3.1. Выборка с использованием логических операций

**Задание:** Сделайте выборку данных из одной таблицы при нескольких условиях, с использованием логических операций, **LIKE**, **BETWEEN**, **IN** (не менее 3-х разных примеров).

Выборка бронирований указанных после **IN** пользователей:

```
1 SELECT *
2 FROM reservation
3 WHERE user_id IN (1, 2, 3)
4 AND "from" < to_date('2017-01-31', 'YYYY-MM-DD');
```

Листинг 1: select-in.sql

	id	room_id	user_id	from	to	price	is_paid
3	1444	1526	1	2017-01-26	2017-02-06	\$5,315.00	t
4	1724	4742	3	2017-01-30	2017-02-14	\$12,607.00	f
5	7195	2238	1	2017-01-16	2017-01-30	\$10,545.00	t
6	(3 rows)						

Выборка бронирований между датами, указанными после **BETWEEN**:

```
1 SELECT *
2 FROM reservation
3 WHERE price < 100 :: money
4 AND "from" BETWEEN to_date('2017-01-01', 'YYYY-MM-DD')
5 AND to_date('2017-01-31', 'YYYY-MM-DD');
```

Листинг 2: select-between.sql

1	id	room_id	user_id	from	to	price	is_paid
2	-----	-----	-----	-----	-----	-----	-----
3	4292	3256	967	2017-01-26	2017-01-31	\$85.00	f
4	5008	1916	283	2017-01-26	2017-02-11	\$2.00	t
5	13670	720	914	2017-01-14	2017-01-17	\$73.00	t
6	16510	4480	382	2017-01-15	2017-01-31	\$42.00	t
7	(4 rows)						

Выборка пользователей, имя которых начинается с «вадим» и номер телефона начинается с единицы:

```

1 SELECT id, name, phone_number, city_id
2 FROM "user"
3 WHERE name LIKE '1%'
4 AND phone_number LIKE '1-666-%';

```

Листинг 3: select-like.sql

1	id	name	phone_number	city_id
2	-----	-----	-----	-----
3	177	вадим.фадеев	1-075-921-5362	110
4	320	вадим.богданова	1-773-542-8914	124
5	730	вадим.попова	1-196-683-9202	138
6	(3 rows)			

## 3.2. Запрос с вычисляемым полем

**Задание:** Создайте в запросе вычисляемое поле.

Вычисляемые значения: длительность бронирования и вычисление стоимости бронирования одного дня:

```

1 SELECT id, "to" - "from" AS duration,
2 price / NULLIF("to" - "from", 0) AS per_night
3 FROM reservation
4 LIMIT 10;

```

Листинг 4: select-eval.sql

1	id	duration	per_night
2	-----	-----	-----
3	1	1	\$1,999.00
4	2	5	\$1,560.00
5	3	7	\$1,421.14
6	4	19	\$599.57
7	5	12	\$488.08
8	6	17	\$449.29
9	7	16	\$235.06
10	8	8	\$749.12
11	9	19	\$278.21
12	10	14	\$815.28
13	(10 rows)		

### 3.3. Выборка с использованием сортировки

**Задание:** Сделайте выборку всех данных с сортировкой по нескольким полям.

Выборка бронирований с сортировкой по флагу оплаты заказа в порядке убывания (оплаченные заказы будут выше) и стоимости в порядке убывания.

```
1 SELECT *
2 FROM reservation
3 ORDER BY is_paid DESC, price DESC
4 LIMIT 10;
```

Листинг 5: select-order.sql

	id	room_id	user_id	from	to	price	is_paid
3	6663	948	282	2019-10-15	2019-10-28	\$14,999.00	t
4	17003	4774	490	2019-12-12	2019-12-14	\$14,997.00	t
5	4504	438	336	2017-11-28	2017-12-11	\$14,996.00	t
6	455	1405	837	2018-12-08	2018-12-08	\$14,996.00	t
7	18251	660	350	2018-11-13	2018-12-01	\$14,995.00	t
8	1972	2006	160	2019-07-27	2019-08-02	\$14,992.00	t
9	16859	3905	648	2017-06-15	2017-06-29	\$14,992.00	t
10	1947	71	63	2018-07-21	2018-08-02	\$14,991.00	t
11	7764	4147	735	2017-01-09	2017-01-18	\$14,980.00	t
12	929	1403	681	2018-11-28	2018-12-14	\$14,979.00	t
13	(10 rows)						

### 3.4. Запрос с вычислением совокупных характеристик таблиц

**Задание:** Создайте запрос, вычисляющий несколько совокупных характеристик таблиц.

Выборка с вычисление совокупных характеристик: количества бронирований, средней продолжительности бронирования, средней и максимальной стоимости заказа и суммы стоимости всех бронирований:

```
1 SELECT COUNT(*) AS count,
2        AVG("to" - "from") AS avg_duration,
3        AVG(price :: numeric) :: money AS avg_price,
4        MAX(price) AS max_price,
5        SUM(price) AS sum_price
6 FROM reservation;
```

Листинг 6: select-summary.sql

	count	avg_duration	avg_price	max_price	sum_price
3	20000	10.0916500000000000	\$7,500.42	\$14,999.00	\$150,008,493.00
4	(1 row)				

### 3.5. Выборка из связанных таблиц

**Задание:** Сделайте выборку данных из связанных таблиц (не менее двух примеров).

Соединение бронирований, комнат и тип комнат:

```
1 SELECT "from", "to", type, capacity
2 FROM reservation
3     JOIN room ON reservation.room_id = room.id
4     JOIN room_type ON room.room_type_id = room_type.id
5 LIMIT 10;
```

Листинг 7: select-join-1.sql

```
1      from |      to      | type | capacity
2  -----+-----+-----+-----
3  2019-06-17 | 2019-06-18 | Hestia | 1
4  2018-08-30 | 2018-09-04 | Zeus | 1
5  2018-07-17 | 2018-07-24 | Ares | 4
6  2018-01-18 | 2018-02-06 | Artemis | 3
7  2018-03-03 | 2018-03-15 | Aphrodite | 1
8  2019-10-03 | 2019-10-20 | Hermes | 2
9  2019-07-29 | 2019-08-14 | Hestia | 5
10 2017-08-18 | 2017-08-26 | Dionysus | 5
11 2017-07-19 | 2017-08-07 | Hades | 5
12 2018-08-05 | 2018-08-19 | Hestia | 5
13 (10 rows)
```

Соединение пользователей, городов и стран:

```
1 SELECT usr.name, city.name AS city, country.name AS country
2 FROM "user" AS usr
3     JOIN city ON usr.city_id = city.id
4     JOIN country ON city.country_id = country.id
5 LIMIT 10;
```

Листинг 8: select-join-2.sql

```
1      name |      city |      country
2  -----+-----+-----
3  светлана.авдеева | Владивосток | Никарагуа
4  валентин.панфилов | Красноярск | Габон
5  вадим.захарова | Рязань | Великобритания
6  владимир.беспалов | Иваново | Великобритания
7  анжела.некрасова | Воронеж | Фарерские Острова (не признана)
8  дмитрий.туров | Омск | Великобритания
9  андрей.щербакова | Сочи | Кирибати
10 фёдор.наумова | Иваново | Лесото
11 никита.щербаков | Брянск | Узбекистан
12 ангелина.кошелев | Кемерово | Лесото
13 (10 rows)
```

### 3.6. Запрос с использованием группировки

**Задание:** Создайте запрос, рассчитывающий совокупную характеристику с использованием группировки, наложите ограничение на результат группировки.

Группировка бронирований по пользователю с накладываемым условием, что число бронирований у пользователя больше 30:

```
1 SELECT user_id
2 FROM reservation
3 GROUP BY user_id
4 HAVING COUNT(1) > 30;
```

Листинг 9: select-group.sql

```
1 user_id
2 -----
3      663
4      745
5      593
6        8
7      846
8      133
9      337
10     163
11     851
12 (9 rows)
```

### 3.7. Вложенный запрос

**Задание:** Придумайте пример использования вложенного запроса.

Вложенный запрос: выборка пользователей, имеющих более 30 бронирований, и соединение с таблицей пользователей после этого:

```
1 SELECT usr.id, usr.name, q.cnt
2 FROM (SELECT user_id, COUNT(1) AS cnt
3       FROM reservation
4       GROUP BY user_id
5       HAVING COUNT(1) > 30
6     ) q
7 JOIN "user" AS usr ON q.user_id = usr.id;
```

Листинг 10: select-subquery.sql

```
1 id | name | cnt
2 ---+-----+---
3  8 | фёдор.наумова | 31
4 133 | евгения.елисеева | 31
5 163 | даниил.субботина | 32
6 337 | клавдия.трофимова | 33
7 593 | ирина.лазарев | 33
8 663 | георгий.степанова | 37
9 745 | юрий.анисимов | 32
10 846 | ольга.титова | 31
11 851 | георгий.виноградова | 31
12 (9 rows)
```

### 3.8. Вставка записей

**Задание:** С помощью оператора `INSERT` добавьте в каждую таблицу по одной записи.

```
1 INSERT INTO house_rules
2 VALUES (DEFAULT, '15:00:00', '12:00:00', 'Предоплата не возвращается при отмене бронирования
   менее чем за сутки'),
3         (DEFAULT, '16:00:00', '14:00:00', 'Предоплата возвращается');
4
5 INSERT INTO hotel
6 VALUES (DEFAULT, 'Гранд Будапешт', 1, 'Республика Зубровка', 5, 'Комфортабельный отель Гранд
   Будапешт');
7
8 INSERT INTO "user"
9 VALUES (DEFAULT, 'traveller', 'traveller@travel.com', '5
   E884898DA28047151D0E56F8DC6292773603D0D6AABDD62A11EF721D1542D8');
10
11 INSERT INTO room_type
12 VALUES (DEFAULT, 1, 'double king-size', 2, 'Номер для некурящих с 2 кроватями размера king-
   size'),
13         (DEFAULT, 1, 'single', 1, 'Номер для некурящих с 1 кроватью');
14
15 INSERT INTO room
16 VALUES (DEFAULT, 1, '№123'),
17         (DEFAULT, 2, '№456'),
18         (DEFAULT, 1, '№789');
19
20 INSERT INTO reservation
21 VALUES (DEFAULT, 1, 1, '2018-10-08', '2018-10-10', 1234, TRUE),
22         (DEFAULT, 2, 1, '2018-10-10', '2018-10-15', 4321, FALSE);
23
24 INSERT INTO guest
25 VALUES (DEFAULT, 1, 'M. Gustave', FALSE),
26         (DEFAULT, 1, 'Mr. Moustafa', FALSE),
27         (DEFAULT, 2, 'Serge X.', FALSE);
28
29 INSERT INTO review
30 VALUES (DEFAULT, 1, 'Все хорошо', 'Все плохо', 3),
31         (DEFAULT, 2, 'Все плохо', 'Все хорошо', 5);
32
33 INSERT INTO price
34 VALUES (DEFAULT, 1, '2018-10-01', '2018-10-31', 1234),
35         (DEFAULT, 1, '2018-09-01', '2018-09-30', 1000),
36         (DEFAULT, 2, '2018-10-01', '2018-10-31', 4321);
37
38 INSERT INTO facility
39 VALUES (DEFAULT, 'Wi-Fi'),
40         (DEFAULT, 'Холодильник'),
41         (DEFAULT, 'TV'),
42         (DEFAULT, 'Душ');
43
44 INSERT INTO room_facility
45 VALUES (1, 2),
46         (1, 3),
47         (1, 4),
48         (2, 1),
49         (2, 4);
```

Листинг 11: insert.sql



### 3.9. Изменение записей

**Задание:** С помощью оператора `UPDATE` измените значения нескольких полей у всех записей, отвечающих заданному условию.

Выставление цены бронирования, равной 1\$, в строках, где цена оказалась равна 0\$. Для наглядности до и после добавим запросы выборки таких строк:

```
1 SELECT *
2 FROM reservation
3 WHERE price = 0 :: money;
4
5 UPDATE reservation
6 SET price = 1 :: money
7 WHERE price = 0 :: money;
8
9 SELECT *
10 FROM reservation
11 WHERE price = 1 :: money;
```

Листинг 12: update.sql

```
1  id | room_id | user_id | from | to | price | is_paid
2  ----+-----+-----+-----+-----+-----+-----
3  9005 | 4836 | 163 | 2019-04-09 | 2019-04-12 | $0.00 | t
4  2679 | 2357 | 576 | 2017-11-15 | 2017-11-17 | $0.00 | f
5  16017 | 2548 | 566 | 2017-03-07 | 2017-03-11 | $0.00 | t
6  16811 | 2256 | 547 | 2019-12-30 | 2020-01-16 | $0.00 | f
7  (4 rows)
8
9 UPDATE 4
10 id | room_id | user_id | from | to | price | is_paid
11 ----+-----+-----+-----+-----+-----+-----
12 (0 rows)
```

### 3.10. Удаление записей по условию

**Задание:** С помощью оператора `DELETE` удалите запись, имеющую максимальное (минимальное) значение некоторой совокупной характеристики.

Удаление строки из таблицы цен, имеющую максимальную цену. Для наглядности до и после добавим запросы выборки таких строк:

```
1 SELECT *
2 FROM price
3 WHERE price.price = (SELECT MAX(price.price) FROM price);
4
5 DELETE
6 FROM price
7 WHERE price.price = (SELECT MAX(price.price) FROM price);
8
9 SELECT *
10 FROM price
11 WHERE price.price = (SELECT MAX(price.price) FROM price);
```

Листинг 13: delete-1.sql

```

1  id | room_type_id | from | to | price
2  -----+-----+-----+-----+-----
3  4904 | 207 | 2018-10-16 | 2018-10-17 | $14,992.00
4  (1 row)
5
6  DELETE 1
7  id | room_type_id | from | to | price
8  -----+-----+-----+-----+-----
9  9493 | 486 | 2017-09-05 | 2017-09-21 | $14,987.00
10 (1 row)

```

### 3.11. Удаление с использованием вложенного запроса

**Задание:** С помощью оператора **DELETE** удалите записи в главной таблице, на которые не ссылается подчиненная таблица (используя вложенный запрос).

Удаление пользователей, не имеющих бронирований. Для наглядности до и после добавим запросы выборки таких строк:

```

1  SELECT usr.id, usr.name
2  FROM "user" usr
3  WHERE usr.id NOT IN (SELECT user_id
4                        FROM reservation
5                        GROUP BY user_id
6                        );
7
8  DELETE
9  FROM "user" AS usr
10 WHERE usr.id NOT IN (SELECT user_id
11                      FROM reservation
12                      GROUP BY user_id
13                      );
14
15 SELECT usr.id, usr.name
16 FROM "user" usr
17 WHERE usr.id NOT IN (SELECT user_id
18                      FROM reservation
19                      GROUP BY user_id
20                      );

```

Листинг 14: delete-2.sql

```

1  id | name
2  -----+-----
3  1003 | test
4  1004 | test
5  (2 rows)
6
7  DELETE 2
8  id | name
9  -----+-----
10 (0 rows)

```

## 4. Запросы в соответствии с заданием преподавателя

### 4.1. Рейтинг городов по кварталам

**Задание:** Вывести рейтинг городов по кварталам. В рейтинге 5 городов в, которые больше всего ездят в каком-то квартале.

Будем формировать итоговый запрос поэтапно:

1. Формирование отчета посещаемости городов по кварталам. Для удобства упорядочим по городу и кварталу и возьмем записи для первых двух городов:

```
1 SELECT city.id AS city_id,  
2        EXTRACT(QUARTER FROM "from") AS quarter,  
3        COUNT(*) AS count  
4 FROM reservation  
5     JOIN room ON room.id = reservation.room_id  
6     JOIN hotel ON hotel.id = room.hotel_id  
7     JOIN city ON hotel.city_id = city.id  
8 GROUP BY city.id, quarter  
9 ORDER BY city.id, quarter  
10 LIMIT 8
```

Листинг 15: quarter-quarter-summary.sql

```
1 city_id | quarter | count  
2 -----+-----+-----  
3         1 |         1 |      9  
4         1 |         2 |     13  
5         1 |         3 |     13  
6         1 |         4 |     10  
7         2 |         1 |     33  
8         2 |         2 |     36  
9         2 |         3 |     25  
10        2 |         4 |     36  
11 (8 rows)
```

2. Формирование порядкового номера полученной строки посещаемости города относительно других городов в рамках квартала. Для этого используется оконная функция ROW\_NUMBER(), условие PARTITION BY разбивает строки на наборы меньшего набора (в нашем случае разбивает по кварталам), а ORDER BY указывает на порядок внутри этого набора (в нашем случае по уменьшению бронирований). Для удобства так же возьмем записи о первых двух городах:

```
1 WITH quarter_summary AS (  
2     SELECT city.id AS city_id,  
3            EXTRACT(QUARTER FROM "from") AS quarter,  
4            COUNT(*) AS count  
5     FROM reservation  
6         JOIN room ON room.id = reservation.room_id  
7         JOIN hotel ON hotel.id = room.hotel_id  
8         JOIN city ON hotel.city_id = city.id  
9     GROUP BY city.id, quarter  
10 )  
11 SELECT ROW_NUMBER() OVER (PARTITION BY quarter ORDER BY SUM(count) DESC) AS row_num,  
12        city_id AS city_id,
```

```

13         quarter                                AS quarter,
14         SUM(count)                             AS reservations
15 FROM quarter_summary
16 GROUP BY quarter, city_id
17 ORDER BY city_id, quarter
18 LIMIT 8

```

Листинг 16: quarter-summary.sql

row_num	city_id	quarter	reservations
148	1	1	9
130	1	2	13
134	1	3	13
141	1	4	10
57	2	1	33
58	2	2	36
89	2	3	25
53	2	4	36

(8 rows)

3. Формирование итогового рейтинга городов по кварталам. Для этого из предыдущего запроса берутся строки с порядковым номером 1, что соответствует ID городов, в которых забронировано больше всего номеров в одном из кварталов. После этого по ID соединяются города и страны:

```

1 WITH summary AS (
2     WITH quarter_summary AS (
3         SELECT city.id                                AS city_id,
4                EXTRACT(QUARTER FROM "from")           AS quarter,
5                COUNT(*)                             AS count
6         FROM reservation
7         JOIN room ON room.id = reservation.room_id
8         JOIN hotel ON hotel.id = room.hotel_id
9         JOIN city ON hotel.city_id = city.id
10        GROUP BY city.id, quarter
11    )
12    SELECT ROW_NUMBER() OVER (PARTITION BY quarter ORDER BY SUM(count) DESC) AS row_num,
13           city_id                                AS city_id,
14           quarter                                AS quarter,
15           SUM(count)                             AS reservations
16    FROM quarter_summary
17    GROUP BY city_id, quarter
18 )
19 SELECT summary.quarter    AS quarter,
20        city.name          AS city_name,
21        country.name       AS country_name,
22        summary.reservations AS reservations
23 FROM summary
24     JOIN city ON summary.city_id = city.id
25     JOIN country ON city.country_id = country.id
26 WHERE summary.row_num < 2
27 ORDER BY summary.quarter, summary.reservations DESC;

```

Листинг 17: quarter.sql

quarter	city_name	country_name	reservations
1	Ростов-на-Дону	Узбекистан	115

4	2		Ростов-на-Дону		Узбекистан		122
5	3		Ростов-на-Дону		Узбекистан		109
6	4		Воронеж		Парагвай		107
7	(4 rows)						

**Измерение времени:** увеличим количество бронирований до 100 тысяч с помощью генератора и измерим время исполнения итогового запроса:

1. execution time: 309 ms, fetching time: 54 ms
2. execution time: 196 ms, fetching time: 23 ms
3. execution time: 183 ms, fetching time: 11 ms

Видно, что после первого запуска время выполнения запроса сильно уменьшилось, что объясняется внутренними оптимизациями базы данных.

## 4.2. Клиенты, имеющие наибольший средний рост стоимости путевки

**Задание:** Вывести 5 клиентов, которые имеют наибольший средний рост стоимости путевки.

Будем формировать итоговый запрос поэтапно:

1. Формирование порядкового номера бронирования относительно бронирований отдельного пользователя. Для этого так же используется оконная функция ROW\_NUMBER() с ID пользователя как условием разбиения на наборы и датой начала бронирования для упорядочивания. Для анализа используются только пользователи, имеющие более 10 заказов. Для удобства возьмем только первые 10 строк:

```

1 SELECT ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY "from") AS row_num,
2        price AS price,
3        user_id AS user_id
4 FROM reservation
5 WHERE is_paid AND user_id IN (
6     SELECT user_id
7     FROM reservation
8     WHERE is_paid
9     GROUP BY user_id
10    HAVING COUNT(*) > 10
11 )
12 LIMIT 10

```

Листинг 18: travellers-increase-summary.sql

1	row_num		price		user_id
2	-----+-----+-----				
3	1		\$10,545.00		1
4	2		\$5,315.00		1
5	3		\$12,319.00		1
6	4		\$8,461.00		1
7	5		\$14,906.00		1
8	6		\$11,208.00		1
9	7		\$7,457.00		1

```

10      8 | $14,935.00 |      1
11      9 | $11,365.00 |      1
12     10 | $5,193.00 |      1
13 (10 rows)

```

2. Формирование для каждого пользователя средней разницы между стоимостью соседних (относительно даты) бронирований и выбор 5 клиентов с наибольшим этим показателем:

```

1 WITH increase_summary AS (
2     SELECT ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY "from") AS row_num,
3           price AS price,
4           user_id AS user_id
5 FROM reservation
6 WHERE is_paid AND user_id IN (
7     SELECT user_id
8     FROM reservation
9     WHERE is_paid
10    GROUP BY user_id
11    HAVING COUNT(*) > 10
12 )
13 )
14 SELECT curr.user_id AS user_id,
15        AVG((curr.price - prev.price) :: numeric) :: money AS avg_diff
16 FROM increase_summary curr
17 JOIN increase_summary prev
18 ON curr.user_id = prev.user_id AND curr.row_num = prev.row_num + 1
19 GROUP BY curr.user_id
20 ORDER BY avg_diff DESC
21 LIMIT 5

```

Листинг 19: travellers-summary.sql

```

1 user_id | avg_diff
2 -----+-----
3      183 | $1,294.90
4      578 | $1,279.00
5      536 | $1,207.30
6      281 | $1,199.18
7      778 | $1,165.42
8 (5 rows)

```

3. Формирование сводной таблицы 5 клиентов, имеющих наибольший средний рост стоимости путевки: ID и имя пользователя, количество бронирований, суммарная стоимость бронирований и средняя разница между бронированиями:

```

1 WITH summary AS (
2     WITH increase_summary AS (
3         SELECT ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY "from") AS row_num,
4               price AS price,
5               user_id AS user_id
6     FROM reservation
7     WHERE is_paid AND user_id IN (
8         SELECT user_id
9         FROM reservation
10        WHERE is_paid
11        GROUP BY user_id

```

```

12         HAVING COUNT(*) > 10
13     )
14 )
15 SELECT curr.user_id AS user_id,
16        AVG((curr.price - prev.price) :: numeric) :: money AS avg_diff
17 FROM increase_summary curr
18      JOIN increase_summary prev
19      ON curr.user_id = prev.user_id AND curr.row_num = prev.row_num + 1
20 GROUP BY curr.user_id
21 ORDER BY avg_diff DESC
22 LIMIT 5
23 )
24 SELECT usr.id, usr.name, reservations, total, avg_diff
25 FROM summary
26      JOIN "user" AS usr ON user_id = usr.id
27      JOIN (
28          SELECT user_id AS user_id,
29                 COUNT(*) AS reservations,
30                 SUM(price) AS total
31          FROM reservation
32          GROUP BY user_id
33      ) AS total ON total.user_id = usr.id;

```

Листинг 20: travellers.sql

1	id	name	reservations	total	avg_diff
2	-----+-----+-----+-----+-----				
3	578	лидия.соловьева	25	\$177,712.00	\$1,279.00
4	778	варвара.корнилова	20	\$171,200.00	\$1,165.42
5	281	анфиса.крылов	25	\$157,819.00	\$1,199.18
6	183	лидия.аксенов	24	\$184,655.00	\$1,294.90
7	536	дмитрий.воробьев	26	\$201,855.00	\$1,207.30
8	(5 rows)				

**Измерение времени:** увеличим количество бронирований до 100 тысяч с помощью генератора и измерим время исполнения итогового запроса:

1. execution time: 85 ms, fetching time: 33 ms
2. execution time: 75 ms, fetching time: 9 ms
3. execution time: 73 ms, fetching time: 9 ms

Видно, что после первого запуска время исполнения несильно уменьшилось.

## 5. Представления и хранимые процедуры

Оформим стандартные запросы и запросы, заданные преподавателем, в виде представлений. Представления – виртуальные именованные таблицы, создаваемые с помощью запроса **SELECT**.

```

1 CREATE VIEW select_in AS
2     SELECT *
3     FROM reservation
4     WHERE user_id IN (1, 2, 3)

```

```

5      AND "from" < to_date('2017-01-31', 'YYYY-MM-DD');
6
7  CREATE VIEW select_between AS
8      SELECT *
9      FROM reservation
10     WHERE price < 100 :: money
11           AND "from" BETWEEN to_date('2017-01-01', 'YYYY-MM-DD')
12                AND to_date('2017-01-31', 'YYYY-MM-DD');
13
14  CREATE VIEW select_like AS
15      SELECT id, name, phone_number, city_id
16      FROM "user"
17      WHERE name LIKE 'Вадим%'
18            AND phone_number LIKE '1-%';
19
20  CREATE VIEW select_eval AS
21      SELECT id, "to" - "from" AS duration,
22             price / NULLIF("to" - "from", 0) AS per_night
23      FROM reservation
24      LIMIT 10;
25
26  CREATE VIEW select_order AS
27      SELECT *
28      FROM reservation
29      ORDER BY is_paid DESC, price DESC
30      LIMIT 10;
31
32  CREATE VIEW select_summary AS
33      SELECT COUNT(*) AS count,
34             AVG("to" - "from") AS avg_duration,
35             AVG(price :: numeric) :: money AS avg_price,
36             MAX(price) AS max_price,
37             SUM(price) AS sum_price
38      FROM reservation;
39
40  CREATE VIEW select_join_1 AS
41      SELECT "from", "to", type, capacity
42      FROM reservation
43           JOIN room ON reservation.room_id = room.id
44           JOIN room_type ON room.room_type_id = room_type.id
45      LIMIT 10;
46
47  CREATE VIEW select_join_2 AS
48      SELECT usr.name, city.name AS city, country.name AS country
49      FROM "user" AS usr
50           JOIN city ON usr.city_id = city.id
51           JOIN country ON city.country_id = country.id
52      LIMIT 10;
53
54  CREATE VIEW select_group AS
55      SELECT user_id
56      FROM reservation
57      GROUP BY user_id
58      HAVING COUNT(1) > 30;
59
60  CREATE VIEW select_subquery AS
61      SELECT usr.id, usr.name, q.cnt
62      FROM (SELECT user_id, COUNT(1) AS cnt
63           FROM reservation
64           GROUP BY user_id

```



```

65         HAVING COUNT(1) > 30
66     ) q
67     JOIN "user" AS usr ON q.user_id = usr.id;
68
69 CREATE VIEW "quarter" AS
70     WITH summary AS (
71         WITH quarter_summary AS (
72             SELECT city.id AS city_id,
73                    EXTRACT(QUARTER FROM "from") AS quarter,
74                    COUNT(*) AS count
75             FROM reservation
76                 JOIN room ON room.id = reservation.room_id
77                 JOIN room_type ON room_type.id = room.room_type_id
78                 JOIN hotel ON hotel.id = room_type.hotel_id
79                 JOIN city ON hotel.city_id = city.id
80             GROUP BY city.id, quarter
81         )
82         SELECT ROW_NUMBER() OVER (PARTITION BY quarter ORDER BY SUM(count) DESC) AS row_num,
83                city_id AS city_id,
84                quarter AS quarter,
85                SUM(count) AS
86     reservations
87     FROM quarter_summary
88     GROUP BY city_id, quarter
89 )
90 SELECT summary.quarter AS quarter,
91        city.name AS city_name,
92        country.name AS country_name,
93        summary.reservations AS reservations
94 FROM summary
95     JOIN city ON summary.city_id = city.id
96     JOIN country ON city.country_id = country.id
97 WHERE summary.row_num < 2
98 ORDER BY summary.quarter, summary.reservations DESC;
99
100 CREATE VIEW travelling AS
101     WITH summary AS (
102         WITH increase_summary AS (
103             SELECT ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY "from") AS row_num,
104                    price AS price,
105                    user_id AS user_id
106             FROM reservation
107             WHERE is_paid AND user_id IN (
108                 SELECT user_id
109                 FROM reservation
110                 WHERE is_paid
111                 GROUP BY user_id
112                 HAVING COUNT(*) > 10
113             )
114         )
115         SELECT curr.user_id AS user_id,
116                AVG((curr.price - prev.price) :: numeric) :: money AS avg_diff
117     FROM increase_summary curr
118         JOIN increase_summary prev
119         ON curr.user_id = prev.user_id AND curr.row_num = prev.row_num + 1
120     GROUP BY curr.user_id
121     ORDER BY avg_diff DESC
122     LIMIT 5
123 )
124 SELECT usr.id, usr.name, reservations, total, avg_diff

```

```

124     FROM summary
125     JOIN "user" AS usr ON user_id = usr.id
126     JOIN (
127         SELECT user_id    AS user_id,
128                COUNT(*)   AS reservations,
129                SUM(price) AS total
130         FROM reservation
131         GROUP BY user_id
132     ) AS total ON total.user_id = usr.id;

```

Листинг 21: views.sql

Оформим запросы изменения и удаления данных как хранимые процедуры. Однако, хранимые процедуры появятся только в PostgreSQL11, поэтому оформим не как процедуры, а как функции, возвращающие void.

```

1 CREATE FUNCTION IncreaseZeroPrice()
2     RETURNS void AS
3 $$
4 BEGIN
5     UPDATE reservation
6     SET price = 1 :: money
7     WHERE price = 0 :: money;
8 END;
9 $$
10 LANGUAGE plpgsql;
11
12 CREATE FUNCTION DeleteMaxPrice()
13     RETURNS void AS
14 $$
15 BEGIN
16     DELETE
17     FROM price
18     WHERE price.price = (SELECT MAX(price.price)
19                          FROM price
20                          );
21 END;
22 $$
23 LANGUAGE plpgsql;
24
25 CREATE FUNCTION DeleteUsersWithoutReservations()
26     RETURNS void AS
27 $$
28 BEGIN
29     DELETE
30     FROM "user" AS usr
31     WHERE usr.id NOT IN (SELECT user_id
32                          FROM reservation
33                          GROUP BY user_id
34                          );
35 END;
36 $$
37 LANGUAGE plpgsql;

```

Листинг 22: functions.sql

## 6. Выводы

В процессе выполнения данной работы:

- изучен SQL-DML;
- выполнены стандартные запросы выборки данных: с условиями, вычисляемыми полями, совокупными характеристиками, с соединениями, с использованием группировки и сортировки;
- выполнены запросы вставки тестовых данных;
- выполнены запросы удаления данных по различным условиям;
- выполнены запросы выборки данных, заданные преподавателем: формирование рейтинга городов по кварталам и выбор 5 клиентов, имеющих наибольший средний рост стоимости путевки;
- запросы выборки оформлены как представления, а изменения и удаления как хранимые процедуры.