



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Искусственного интеллекта

Кафедра Промышленной информатики

Отчет по практическим работам №1-6

по дисциплине

«Разработка баз данных»

Студент группы: ИКБО-15-22

Оганнисяг Г.А.

(Ф.И.О студента)

Преподаватель

Корнеев М.С.

(Ф.И.О. преподавателя)

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Работа в среде MySQL Workbench.....	4
1.1 Построение диаграммы IDEF1X	4
2 Создание базы данных и таблиц	6
3 Заполнение таблиц данными	10
3.1 Процесс заполнения таблиц.....	10
3.2 Содержание заполненных таблиц	14
4 Выборка и сортировка данных	19
4.1 Выборка данных по различным параметрам.....	19
4.2 Изменение данных в таблице.....	26
4.3 Выборка данных при помощи различных операций	30
5 Выборка с помощью процедур, функций и триггеров.....	38
5.1 Хранимые процедуры.....	38
5.2 Функции.....	44
5.3 Триггеры	48
6 Оконные функции.....	55
6.1 Синтаксис оконных функций.....	55
6.2 Агрегатные функции	57
6.3 Ранжирующие функции	59
6.4 Функции смещения.....	62
ЗАКЛЮЧЕНИЕ.....	64

ВВЕДЕНИЕ

Для разработки базы данных пекарни, производящей высококачественные хлебобулочные изделия, необходимо создать несколько таблиц, чтобы охватить все аспекты процесса производства и обслуживания клиентов.

В первую очередь важно учитывать поступление ингредиентов высокого качества от поставщиков. Эти ингредиенты распределяются по различным участкам производства, где пекари могут замешивать тесто, выпекать хлеб и формировать изделия. Важную роль играет также контроль качества, так как хлебобулочные изделия предполагают близкий контакт с пищевыми продуктами. Затем продукция поступает в продажу, где клиенты могут ее приобрести.

База данных содержит таблицы для хранения информации о сотрудниках, клиентах, заказах, готовой продукции, поставщиках и необходимых ингредиентах. Каждая таблица включает ключевые поля, которые обеспечивают идентификацию записей и позволяют связать данные между различными таблицами. К примеру, одна из таблиц хранит данные о персонале, другая — информацию о клиентах, третья — данные о заказах, и ещё одна — информацию о готовой продукции. Эти таблицы связаны между собой через ключевые поля, что обеспечивает целостность и структурированность данных в системе.

Таким образом, разработка базы данных для пекарни обеспечивает полный учет всех этапов производства и взаимодействия с клиентами, начиная от получения ингредиентов до продажи готовой продукции.

1 Работа в среде MySQL Workbench

1.1 Построение диаграммы IDEF1X

Методология IDEF1X использует различные концепции, такие как сущности, атрибуты, отношения и ключи, для построения концептуальных и логических моделей данных. Она акцентирует внимание на ясном и строгом определении структуры данных, что позволяет лучше понимать и управлять информационными ресурсами организации.

Основными элементами методологии IDEF1X являются:

- Сущности (Entity): основные объекты, о которых хранится информация.
- Атрибуты (Attribute): характеристики сущностей, которые содержат данные.
- Отношения (Relationship): связи между сущностями, определяющие логические связи между данными.
- Ключи (Key): уникальные идентификаторы сущностей, используемые для обеспечения уникальности и целостности данных.

IDEF1X является мощным инструментом для анализа, проектирования и реализации баз данных, позволяя создавать четкие и понятные модели данных, которые легко интерпретировать и поддерживать.

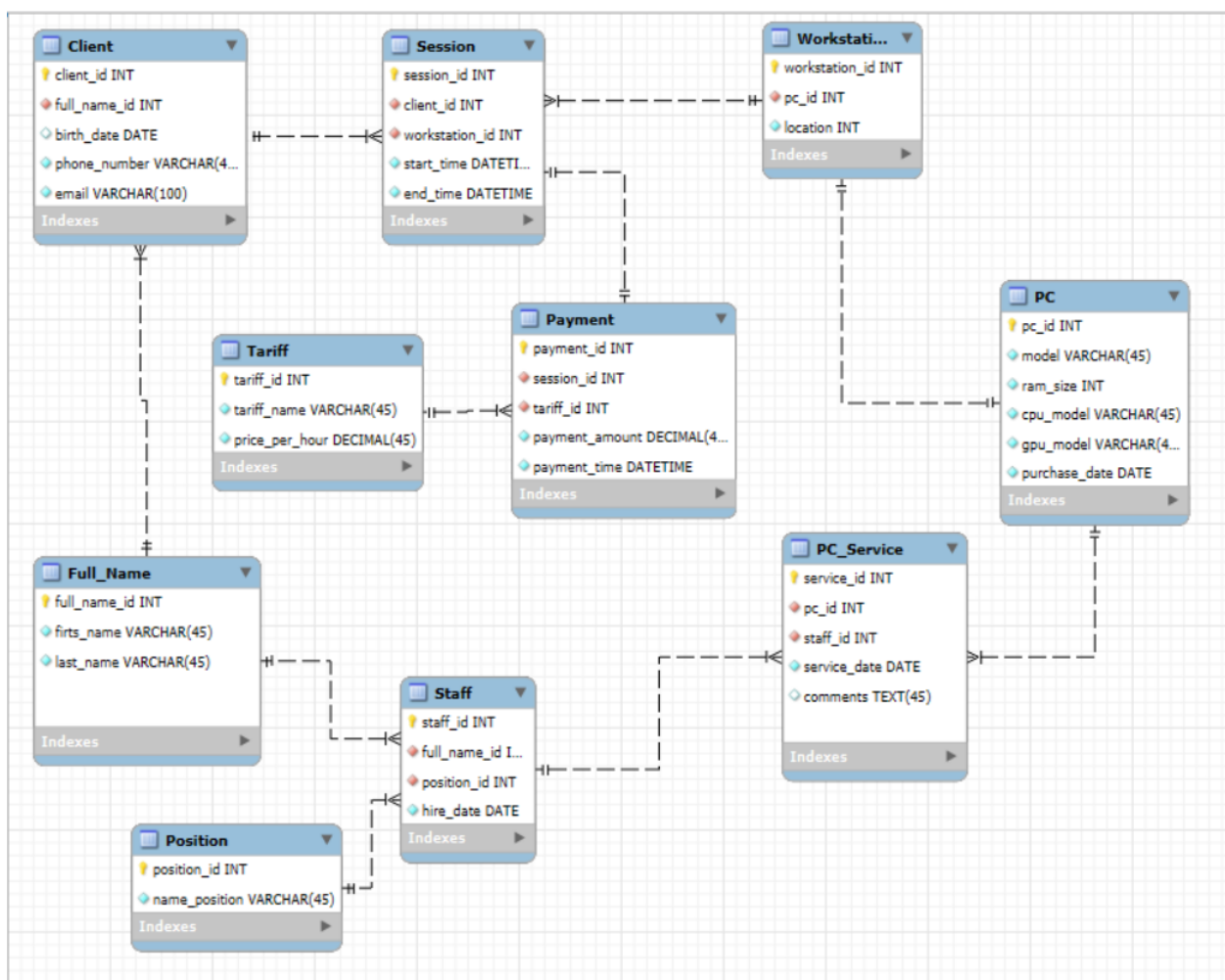


Рисунок 1 — IDEF1X

2 Создание базы данных и таблиц

Для создания базы данных и таблиц используется pgAdmin4.

```
postgres=# CREATE DATABASE pc_club;
CREATE DATABASE
postgres=# \c pc_club;
Вы подключены к базе данных "pc_club" как пользователь "postgres".
pc_club=# |
```

Рисунок 2 — Создание базы данных и подключение к ней

```
pc_club=# CREATE TABLE Client (
pc_club(#      client_id SERIAL PRIMARY KEY,
pc_club(#      full_name_id INT NOT NULL,
pc_club(#      birth_date DATE,
pc_club(#      phone_number VARCHAR(45) NOT NULL,
pc_club(#      email VARCHAR(45) NOT NULL UNIQUE
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE PC (
pc_club(#      pc_id SERIAL PRIMARY KEY,
pc_club(#      model VARCHAR(45) NOT NULL,
pc_club(#      ram_size INT NOT NULL,
pc_club(#      cpu_model VARCHAR(45) NOT NULL,
pc_club(#      gpu_model VARCHAR(45) NOT NULL,
pc_club(#      purchase_date DATE NOT NULL
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE Workstation (
pc_club(#      workstation_id SERIAL PRIMARY KEY,
pc_club(#      pc_id INT NOT NULL,
pc_club(#      location INT NOT NULL,
pc_club(#      FOREIGN KEY (pc_id) REFERENCES PC(pc_id)
pc_club(# );
CREATE TABLE
pc_club=# |
```

Рисунок 3 — Создание таблиц «Client», «PC», «Workstation»

```
pc_club=# CREATE TABLE Session (
pc_club(#      session_id SERIAL PRIMARY KEY,
pc_club(#      client_id INT NOT NULL,
pc_club(#      workstation_id INT NOT NULL,
pc_club(#      start_time TIMESTAMP NOT NULL,
pc_club(#      end_time TIMESTAMP NOT NULL,
pc_club(#      FOREIGN KEY (client_id) REFERENCES Client(client_id),
pc_club(#      FOREIGN KEY (workstation_id) REFERENCES Workstation(workstat
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE Tariff (
pc_club(#      tariff_id SERIAL PRIMARY KEY,
pc_club(#      tariff_name VARCHAR(45) NOT NULL,
pc_club(#      price_per_hour DECIMAL(10, 2) NOT NULL
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE Payment (
pc_club(#      payment_id SERIAL PRIMARY KEY,
pc_club(#      session_id INT NOT NULL,
pc_club(#      tariff_id INT NOT NULL,
pc_club(#      payment_amount DECIMAL(10, 2) NOT NULL,
pc_club(#      payment_time TIMESTAMP NOT NULL,
pc_club(#      FOREIGN KEY (session_id) REFERENCES Session(session_id),
pc_club(#      FOREIGN KEY (tariff_id) REFERENCES Tariff(tariff_id)
pc_club(# );
CREATE TABLE
pc_club=# |
```

Рисунок 4 — Создание таблицы «Session», «Tariff», «Payment»,

```

pc_club=# CREATE TABLE Staff (
pc_club(#      staff_id SERIAL PRIMARY KEY,
pc_club(#      full_name_id INT NOT NULL,
pc_club(#      position_id INT NOT NULL,
pc_club(#      hire_date DATE NOT NULL
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE PC_Service (
pc_club(#      service_id SERIAL PRIMARY KEY,
pc_club(#      pc_id INT NOT NULL,
pc_club(#      staff_id INT NOT NULL,
pc_club(#      service_date DATE NOT NULL,
pc_club(#      service_type VARCHAR(45) NOT NULL,
pc_club(#      comments TEXT,
pc_club(#      FOREIGN KEY (pc_id) REFERENCES PC(pc_id),
pc_club(#      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
pc_club(# );
CREATE TABLE
pc_club=# |

```

Рисунок 5 — Создание таблицы «Staff», «PC_Service»

```

pc_club=# CREATE TABLE Full_Name (
pc_club(#      full_name_id SERIAL PRIMARY KEY,
pc_club(#      first_name VARCHAR(45) NOT NULL,
pc_club(#      last_name VARCHAR(45) NOT NULL
pc_club(# );
CREATE TABLE
pc_club=# CREATE TABLE Position (
pc_club(#      position_id SERIAL PRIMARY KEY,
pc_club(#      name_position VARCHAR(45) NOT NULL
pc_club(# );
CREATE TABLE
pc_club=# |

```

Рисунок 6 — Создание таблицы «Full_Name», «Position»

```

pc_club=# ALTER TABLE Staff
pc_club=# ADD FOREIGN KEY (full_name_id) REFERENCES Full_Name(full_name_id),
pc_club=# ADD FOREIGN KEY (position_id) REFERENCES Position(position_id);
ALTER TABLE
pc_club=#
pc_club=# ALTER TABLE Client
pc_club=# ADD FOREIGN KEY (full_name_id) REFERENCES Full_Name(full_name_id);
ALTER TABLE
pc_club=# |

```

Рисунок 7 — Подключение внешних связей для «Staff», «Client»

Выводим список всех таблиц, созданных в базе данных при помощи команды «\d».

```
pc_club=# \d
               List of relations
Schema |           Name           | Type  | Owner
-----+-----+-----+-----
public | client                   | table  | postgres
public | client_client_id_seq     | sequence | postgres
public | full_name                 | table  | postgres
public | full_name_full_name_id_seq | sequence | postgres
public | payment                   | table  | postgres
public | payment_payment_id_seq   | sequence | postgres
public | pc                         | table  | postgres
public | pc_pc_id_seq              | sequence | postgres
public | pc_service                | table  | postgres
public | pc_service_service_id_seq | sequence | postgres
public | position                  | table  | postgres
public | position_position_id_seq  | sequence | postgres
public | session                   | table  | postgres
public | session_session_id_seq   | sequence | postgres
public | staff                     | table  | postgres
public | staff_staff_id_seq       | sequence | postgres
public | tariff                    | table  | postgres
public | tariff_tariff_id_seq     | sequence | postgres
public | workstation                | table  | postgres
public | workstation_workstation_id_seq | sequence | postgres
(20 rows)
```

Рисунок 8 — Список созданных таблиц

3 Заполнение таблиц данными

3.1 Процесс заполнения таблиц

Для заполнения таблиц также используется pgAdmin4. Заполняем каждую таблицу используя оператор «INSERT INTO имя_таблицы».

```
pc_club=# INSERT INTO full_name (first_name, last_
name)
pc_club=# VALUES
pc_club=# ('John', 'Doe'), ('Jane', 'Smith'), ('Sa
m', 'Brown'),
pc_club=# ('Alex', 'Johnson'), ('Chris', 'Lee'), (
'Emily', 'Davis'),
pc_club=# ('Michael', 'Miller'), ('Sara', 'Wilson'
), ('David', 'Moore'),
pc_club=# ('Laura', 'Taylor');
INSERT 0 10
```

Рисунок 9 — Заполнение таблицы «Full_Name»

```
pc_club=# INSERT INTO client (full_name_id, birth_
date, phone_number, email)
pc_club=# VALUES
pc_club=# (1, '1985-05-12', '1234567890', 'john.do
e@example.com'),
pc_club=# (2, '1990-06-22', '0987654321', 'jane.sm
ith@example.com'),
pc_club=# (3, '1988-03-15', '1122334455', 'sam.bro
wn@example.com'),
pc_club=# (4, '1995-11-30', '5566778899', 'alex.jo
hnson@example.com'),
pc_club=# (5, '1980-12-25', '6677889900', 'chris.l
ee@example.com'),
pc_club=# (6, '1992-08-18', '7788990011', 'emily.d
avis@example.com'),
pc_club=# (7, '1983-01-05', '8899001122', 'michael
.miller@example.com'),
pc_club=# (8, '1997-07-20', '9900112233', 'sara.wi
lson@example.com'),
pc_club=# (9, '1981-09-14', '1122445566', 'david.m
oore@example.com'),
pc_club=# (10, '1999-10-05', '2233556677', 'laura.
taylor@example.com');
INSERT 0 10
pc_club=#
```

Рисунок 10 — Заполнение таблицы «Client»

```

pc_club=# INSERT INTO pc (model, ram_size, cpu_model, gpu_model, purchase_date)
pc_club=# VALUES
pc_club=# ('PC1', 16, 'Intel i5', 'NVIDIA GTX 1650', '2022-01-01'),
pc_club=# ('PC2', 32, 'AMD Ryzen 7', 'NVIDIA RTX 3060', '2023-02-10'),
pc_club=# ('PC3', 16, 'Intel i7', 'NVIDIA GTX 1660', '2022-03-15'),
pc_club=# ('PC4', 8, 'AMD Ryzen 5', 'NVIDIA GTX 1050', '2021-07-25'),
pc_club=# ('PC5', 16, 'Intel i5', 'AMD RX 580', '2022-05-30'),
pc_club=# ('PC6', 32, 'Intel i9', 'NVIDIA RTX 3080', '2023-06-12'),
pc_club=# ('PC7', 16, 'AMD Ryzen 9', 'NVIDIA RTX 3070', '2023-07-21'),
pc_club=# ('PC8', 8, 'Intel i3', 'NVIDIA GTX 950', '2021-09-10'),
pc_club=# ('PC9', 16, 'AMD Ryzen 3', 'NVIDIA GTX 1060', '2020-11-15'),
pc_club=# ('PC10', 32, 'Intel i7', 'NVIDIA RTX 2070', '2022-12-31');
INSERT 0 10

```

Рисунок 11 — Заполнение таблицы «PC»

```

pc_club=# INSERT INTO workstation (pc_id, location)
pc_club=# VALUES
pc_club=# (1, 'Room 101'), (2, 'Room 102'), (3, 'Room 103'),
pc_club=# (4, 'Room 104'), (5, 'Room 105'), (6, 'Room 106'),
pc_club=# (7, 'Room 107'), (8, 'Room 108'), (9, 'Room 109'),
pc_club=# (10, 'Room 110');
INSERT 0 10

```

Рисунок 12 — Заполнение таблицы «Workstation»

```

pc_club=# INSERT INTO session (client_id, workstat
ion_id, start_time, end_time)
pc_club=# VALUES
pc_club=# (1, 2, '2024-09-30 10:00:00', '2024-09-3
0 12:00:00'),
pc_club=# (2, 4, '2024-09-30 11:00:00', '2024-09-3
0 13:00:00'),
pc_club=# (3, 3, '2024-09-30 14:00:00', '2024-09-3
0 16:00:00'),
pc_club=# (4, 5, '2024-10-01 09:00:00', '2024-10-0
1 11:00:00'),
pc_club=# (5, 6, '2024-10-01 13:00:00', '2024-10-0
1 15:00:00'),
pc_club=# (6, 7, '2024-10-02 10:00:00', '2024-10-0
2 12:00:00'),
pc_club=# (7, 8, '2024-10-02 11:00:00', '2024-10-0
2 13:00:00'),
pc_club=# (8, 9, '2024-10-02 14:00:00', '2024-10-0
2 16:00:00'),
pc_club=# (9, 1, '2024-10-03 10:00:00', '2024-10-0
3 12:00:00'),
pc_club=# (10, 10, '2024-10-03 14:00:00', '2024-10
-03 16:00:00');
INSERT 0 10

```

Рисунок 13 — Заполнение таблицы «Session»

```

pc_club=# INSERT INTO tariff (tariff_name, price_p
er_hour)
pc_club=# VALUES
pc_club=# ('Standard', 5.00), ('Premium', 10.00),
('VIP', 15.00),
pc_club=# ('Economy', 3.00), ('Basic', 2.00), ('Ad
vanced', 8.00),
pc_club=# ('Deluxe', 12.00), ('Pro', 6.00), ('Elit
e', 18.00),
pc_club=# ('Student', 4.00);
INSERT 0 10

```

Рисунок 14 — Заполнение таблицы «Tariff»

```

pc_club=# INSERT INTO position (name_position)
pc_club=# VALUES
pc_club=# ('Technician'), ('Manager'), ('Developer
'), ('Support'),
pc_club=# ('Network Administrator'), ('Security Sp
ecialist'),
pc_club=# ('System Analyst'), ('Database Administr
ator'),
pc_club=# ('Hardware Engineer'), ('Helpdesk Specia
list');
INSERT 0 10

```

Рисунок 15 — Заполнение таблицы «Position»

```

pc_club=# INSERT INTO payment (session_id, tariff_
id, payment_amount, payment_time)VALUES(3, 1, 15.0
0, '2024-09-30 12:00:00'), (5, 3, 30.00, '2024-09-
30 13:00:00'), (7, 2, 20.00, '2024-09-30 14:00:00'
), (4, 5, 10.00, '2024-10-01 11:00:00'), (1, 7, 35
.00, '2024-10-01 15:00:00'), (6, 9, 45.00, '2024-1
0-02 12:00:00'), (2, 4, 8.00, '2024-10-02 16:00:00
'), (8, 6, 24.00, '2024-10-02 12:00:00'), (10, 8,
40.00, '2024-10-03 12:00:00'), (9, 10, 50.00, '202
4-10-03 16:00:00');
INSERT 0 10

```

Рисунок 16 — Заполнение таблицы «Payment»

```

pc_club=# INSERT INTO staff (full_name_id, positio
n_id, hire_date)
pc_club=# VALUES
pc_club=# (1, 1, '2023-01-01'), (2, 2, '2022-05-10
'), (3, 3, '2021-03-15'),
pc_club=# (4, 4, '2023-07-20'), (5, 5, '2021-11-25
'), (6, 6, '2020-09-13'),
pc_club=# (7, 7, '2019-12-30'), (8, 8, '2022-04-22
'), (9, 9, '2023-03-03'),
pc_club=# (10, 10, '2021-06-18');
INSERT 0 10

```

Рисунок 17 — Заполнение таблицы «Staff»

```

pc_club=# INSERT INTO pc_service (pc_id, staff_id,
service_date, service_type, comments)
pc_club=# VALUES
pc_club=# (1, 1, '2024-09-15', 'Repair', 'Replaced
faulty RAM'),
pc_club=# (2, 2, '2024-09-18', 'Upgrade', 'Upgrade
d to 32GB RAM'),
pc_club=# (3, 3, '2024-09-20', 'Maintenance', 'Cle
aned the fans and applied new thermal paste'),
pc_club=# (4, 4, '2024-09-22', 'Repair', 'Replaced
GPU'),
pc_club=# (5, 5, '2024-09-25', 'Maintenance', 'Rou
tine check-up'),
pc_club=# (6, 6, '2024-09-27', 'Repair', 'Fixed po
wer supply issue'),
pc_club=# (7, 7, '2024-09-29', 'Upgrade', 'Upgrade
d to SSD storage'),
pc_club=# (8, 8, '2024-10-01', 'Maintenance', 'Cle
aned and applied new thermal paste'),
pc_club=# (9, 9, '2024-10-03', 'Repair', 'Replaced
CPU cooler'),
pc_club=# (10, 10, '2024-10-05', 'Maintenance', 'R
outine cleaning and check-up');
INSERT 0 10

```

Рисунок 18 — Заполнение таблицы «PC_Service»

3.2 Содержание заполненных таблиц

Для того, чтобы посмотреть полное содержание каждой таблицы используем конструкцию «SELECT * FROM имя_таблицы;»

```

pc_club=# SELECT * FROM client;
 client_id | full_name_id | birth_date | phone_number | email
-----+-----+-----+-----+-----
      1 |      1 | 1985-05-12 | 1234567890 | john.doe@example.com
      2 |      2 | 1990-06-22 | 0987654321 | jane.smith@example.com
      3 |      3 | 1988-03-15 | 1122334455 | sam.brown@example.com
      4 |      4 | 1995-11-30 | 5566778899 | alex.johnson@example.com
      5 |      5 | 1980-12-25 | 6677889900 | chris.lee@example.com
      6 |      6 | 1992-08-18 | 7788990011 | emily.davis@example.com
      7 |      7 | 1983-01-05 | 8899001122 | michael.miller@example.com
      8 |      8 | 1997-07-20 | 9900112233 | sara.wilson@example.com
      9 |      9 | 1981-09-14 | 1122445566 | david.moore@example.com
     10 |     10 | 1999-10-05 | 2233556677 | laura.taylor@example.com
(10 rows)

```

Рисунок 19 — Содержание таблицы «Client»

```
pc_club=# SELECT * FROM pc;
```

pc_id	model	ram_size	cpu_model	gpu_model	purchase_date
1	PC1	16	Intel i5	NVIDIA GTX 1650	2022-01-01
2	PC2	32	AMD Ryzen 7	NVIDIA RTX 3060	2023-02-10
3	PC3	16	Intel i7	NVIDIA GTX 1660	2022-03-15
4	PC4	8	AMD Ryzen 5	NVIDIA GTX 1050	2021-07-25
5	PC5	16	Intel i5	AMD RX 580	2022-05-30
6	PC6	32	Intel i9	NVIDIA RTX 3080	2023-06-12
7	PC7	16	AMD Ryzen 9	NVIDIA RTX 3070	2023-07-21
8	PC8	8	Intel i3	NVIDIA GTX 950	2021-09-10
9	PC9	16	AMD Ryzen 3	NVIDIA GTX 1060	2020-11-15
10	PC10	32	Intel i7	NVIDIA RTX 2070	2022-12-31

(10 rows)

Рисунок 20 — Содержание таблицы «PC»

```
pc_club=# SELECT * FROM workstation;
```

workstation_id	pc_id	location
1	1	Room 101
2	2	Room 102
3	3	Room 103
4	4	Room 104
5	5	Room 105
6	6	Room 106
7	7	Room 107
8	8	Room 108
9	9	Room 109
10	10	Room 110

(10 rows)

Рисунок 21 — Содержание таблицы «Workstation»

```
pc_club=# SELECT * FROM session;
```

session_id	client_id	workstation_id	start_time	end_time
1	1	2	2024-09-30 10:00:00	2024-09-30 12:00:00
2	2	4	2024-09-30 11:00:00	2024-09-30 13:00:00
3	3	3	2024-09-30 14:00:00	2024-09-30 16:00:00
4	4	5	2024-10-01 09:00:00	2024-10-01 11:00:00
5	5	6	2024-10-01 13:00:00	2024-10-01 15:00:00
6	6	7	2024-10-02 10:00:00	2024-10-02 12:00:00
7	7	8	2024-10-02 11:00:00	2024-10-02 13:00:00
8	8	9	2024-10-02 14:00:00	2024-10-02 16:00:00
9	9	1	2024-10-03 10:00:00	2024-10-03 12:00:00
10	10	10	2024-10-03 14:00:00	2024-10-03 16:00:00

(10 rows)

Рисунок 22 — Содержание таблицы «Session»

```
pc_club=# SELECT * FROM tariff;
 tariff_id | tariff_name | price_per_hour
-----+-----+-----
          1 | Standard   |           5.00
          2 | Premium    |          10.00
          3 | VIP        |          15.00
          4 | Economy    |           3.00
          5 | Basic      |           2.00
          6 | Advanced   |           8.00
          7 | Deluxe     |          12.00
          8 | Pro        |           6.00
          9 | Elite      |          18.00
         10 | Student    |           4.00
(10 rows)
```

Рисунок 23 — Содержание таблицы «Tariff»

```
pc_club=# SELECT * FROM payment;
 payment_id | session_id | tariff_id | payment_amount | payment_time
-----+-----+-----+-----+-----
          1 |          3 |          1 |          15.00 | 2024-09-30 12:00:00
          2 |          5 |          3 |          30.00 | 2024-09-30 13:00:00
          3 |          7 |          2 |          20.00 | 2024-09-30 14:00:00
          4 |          4 |          5 |          10.00 | 2024-10-01 11:00:00
          5 |          1 |          7 |          35.00 | 2024-10-01 15:00:00
          6 |          6 |          9 |          45.00 | 2024-10-02 12:00:00
          7 |          2 |          4 |           8.00 | 2024-10-02 16:00:00
          8 |          8 |          6 |          24.00 | 2024-10-02 12:00:00
          9 |         10 |          8 |          40.00 | 2024-10-03 12:00:00
         10 |          9 |         10 |          50.00 | 2024-10-03 16:00:00
(10 rows)
```

Рисунок 24 — Содержание таблицы «Payment»

```
pc_club=# SELECT * FROM staff;
 staff_id | full_name_id | position_id | hire_date
-----+-----+-----+-----
          1 |              |            1 | 2023-01-01
          2 |              |            2 | 2022-05-10
          3 |              |            3 | 2021-03-15
          4 |              |            4 | 2023-07-20
          5 |              |            5 | 2021-11-25
          6 |              |            6 | 2020-09-13
          7 |              |            7 | 2019-12-30
          8 |              |            8 | 2022-04-22
          9 |              |            9 | 2023-03-03
         10 |              |           10 | 2021-06-18
(10 rows)
```

Рисунок 25 — Содержание таблицы «Staff»


```

pc_club=# SELECT * FROM pc_service;
 service_id | pc_id | staff_id | service_date | service_type | comments
-----+-----+-----+-----+-----+-----
          1 |    1 |         1 | 2024-09-15   | Repair       | Replaced faulty RAM
          2 |    2 |         2 | 2024-09-18   | Upgrade      | Upgraded to 32GB RAM
          3 |    3 |         3 | 2024-09-20   | Maintenance  | Cleaned the fans and applied new thermal paste
          4 |    4 |         4 | 2024-09-22   | Repair       | Replaced GPU
          5 |    5 |         5 | 2024-09-25   | Maintenance  | Routine check-up
          6 |    6 |         6 | 2024-09-27   | Repair       | Fixed power supply issue
          7 |    7 |         7 | 2024-09-29   | Upgrade      | Upgraded to SSD storage
          8 |    8 |         8 | 2024-10-01   | Maintenance  | Cleaned and applied new thermal paste
          9 |    9 |         9 | 2024-10-03   | Repair       | Replaced CPU cooler
         10 |   10 |        10 | 2024-10-05   | Maintenance  | Routine cleaning and check-up
(10 rows)

```

Рисунок 26 — Содержание таблицы «PC_Service»

```

pc_club=# SELECT * FROM full_name;
 full_name_id | first_name | last_name
-----+-----+-----
           1 | John      | Doe
           2 | Jane      | Smith
           3 | Sam       | Brown
           4 | Alex      | Johnson
           5 | Chris     | Lee
           6 | Emily     | Davis
           7 | Michael   | Miller
           8 | Sara      | Wilson
           9 | David     | Moore
          10 | Laura     | Taylor
(10 rows)

```

Рисунок 27 — Содержание таблицы «Full_Name»

```
pc_club=# SELECT * FROM position;
 position_id |      name_position
-----+-----
          1 | Technician
          2 | Manager
          3 | Developer
          4 | Support
          5 | Network Administrator
          6 | Security Specialist
          7 | System Analyst
          8 | Database Administrator
          9 | Hardware Engineer
         10 | Helpdesk Specialist
(10 rows)
```

Рисунок 28 — Содержание таблицы «Position»

4 Выборка и сортировка данных

4.1 Выборка данных по различным параметрам

Query

Query History

1

▼

SELECT * FROM Client

2

WHERE client_id = 4

Data Output

Messages

Notifications

≡+

▼

▼

SQL

	client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1	4	4	1995-11-30	5566778899	alex.johnson@example.com

Рисунок 29 — Использование оператора «=»

Query

Query History

1

▼

SELECT

*

FROM

Client

2

WHERE

client_id

>

4

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1	5	5	1980-12-25	6677889900	chris.lee@example.com
2	6	6	1992-08-18	7788990011	emily.davis@example.com
3	7	7	1983-01-05	8899001122	michael.miller@example.com
4	8	8	1997-07-20	9900112233	sara.wilson@example.com
5	9	9	1981-09-14	1122445566	david.moore@example.com
6	10	10	1999-10-05	2233556677	laura.taylor@example.com

Рисунок 30 — Использование оператора «>»

Query		Query History				
1	▼	SELECT * FROM Client				
2		WHERE client_id < 4				
Data Output		Messages				
		<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>				
		client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1		1	1	1985-05-12	1234567890	john.doe@example.com
2		2	2	1990-06-22	0987654321	jane.smith@example.com
3		3	3	1988-03-15	1122334455	sam.brown@example.com

Рисунок 31 — Использование оператора «<»

Query		Query History				
1	▼	SELECT * FROM Client				
2		WHERE client_id >= 4				
Data Output		Messages				
		<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>				
		client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1		4	4	1995-11-30	5566778899	alex.johnson@example.com
2		5	5	1980-12-25	6677889900	chris.lee@example.com
3		6	6	1992-08-18	7788990011	emily.davis@example.com
4		7	7	1983-01-05	8899001122	michael.miller@example.com
5		8	8	1997-07-20	9900112233	sara.wilson@example.com
6		9	9	1981-09-14	1122445566	david.moore@example.com
7		10	10	1999-10-05	2233556677	laura.taylor@example.com

Рисунок 32 — Использование оператора «>=»

Query
Query History

1 ▼ **SELECT** * **FROM** Client
2 **WHERE** client_id <= 4

Data Output
Messages
Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	client_id [PK] integer ✎	full_name_id integer ✎	birth_date date ✎	phone_number character varying (20) ✎	email character varying (50) ✎
1	1	1	1985-05-12	1234567890	john.doe@example.com
2	2	2	1990-06-22	0987654321	jane.smith@example.com
3	3	3	1988-03-15	1122334455	sam.brown@example.com
4	4	4	1995-11-30	5566778899	alex.johnson@example.com

Рисунок 33 — Использование оператора «<=»

Query		Query History				
1	✓	SELECT * FROM Client				
2		WHERE client_id != 4				
Data Output		Messages				
		Notifications				
		SQL				
		client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1		1	1	1985-05-12	1234567890	john.doe@example.com
2		2	2	1990-06-22	0987654321	jane.smith@example.com
3		3	3	1988-03-15	1122334455	sam.brown@example.com
4		5	5	1980-12-25	6677889900	chris.lee@example.com
5		6	6	1992-08-18	7788990011	emily.davis@example.com
6		7	7	1983-01-05	8899001122	michael.miller@example.com
7		8	8	1997-07-20	9900112233	sara.wilson@example.com
8		9	9	1981-09-14	1122445566	david.moore@example.com
9		10	10	1999-10-05	2233556677	laura.taylor@example.com

Рисунок 34 — Использование оператора «!=»

Query		Query History				
1	▼	<u>SELECT * FROM Client</u>				
2		<u>WHERE birth_date IS NOT NULL</u>				
Data Output		Messages				
		<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>				
		client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1		1	1	1985-05-12	1234567890	john.doe@example.com
2		2	2	1990-06-22	0987654321	jane.smith@example.com
3		3	3	1988-03-15	1122334455	sam.brown@example.com
4		4	4	1995-11-30	5566778899	alex.johnson@example.com
5		5	5	1980-12-25	6677889900	chris.lee@example.com
6		6	6	1992-08-18	7788990011	emily.davis@example.com
7		7	7	1983-01-05	8899001122	michael.miller@example.com
8		8	8	1997-07-20	9900112233	sara.wilson@example.com
9		9	9	1981-09-14	1122445566	david.moore@example.com
10		10	10	1999-10-05	2233556677	laura.taylor@example.com

Рисунок 35 — Использование оператора «IS NOT NULL»

Query		Query History				
1	▼	<u>SELECT * FROM Client</u>				
2		<u>WHERE email IS NULL</u>				
Data Output		Messages				
		<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>				
		client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)

Рисунок 36 — Использование оператора «IS NULL»

Query Query History	
1	SELECT * FROM PC
2	WHERE ram_size IN(8,32)
Data Output Messages Notifications	
<div> <div>≡</div> <div>+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>	
	<div> <div>pc_id</div> <div>[PK] integer</div> <div>✎</div> </div>
	<div> <div>model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>ram_size</div> <div>integer</div> <div>✎</div> </div>
	<div> <div>cpu_model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>gpu_model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>purchase_date</div> <div>date</div> <div>✎</div> </div>
1	2 PC2 32 AMD Ryzen 7 NVIDIA RTX 3060 2023-02-10
2	4 PC4 8 AMD Ryzen 5 NVIDIA GTX 1050 2021-07-25
3	6 PC6 32 Intel i9 NVIDIA RTX 3080 2023-06-12
4	8 PC8 8 Intel i3 NVIDIA GTX 950 2021-09-10
5	10 PC10 32 Intel i7 NVIDIA RTX 2070 2022-12-31

Рисунок 37 — Использование оператора «IN»

Query Query History	
1	SELECT * FROM PC
2	WHERE ram_size NOT IN(8,32)
Data Output Messages Notifications	
<div> <div>≡</div> <div>+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>	
	<div> <div>pc_id</div> <div>[PK] integer</div> <div>✎</div> </div>
	<div> <div>model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>ram_size</div> <div>integer</div> <div>✎</div> </div>
	<div> <div>cpu_model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>gpu_model</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>purchase_date</div> <div>date</div> <div>✎</div> </div>
1	1 PC1 16 Intel i5 NVIDIA GTX 1650 2022-01-01
2	3 PC3 16 Intel i7 NVIDIA GTX 1660 2022-03-15
3	5 PC5 16 Intel i5 AMD RX 580 2022-05-30
4	7 PC7 16 AMD Ryzen 9 NVIDIA RTX 3070 2023-07-21
5	9 PC9 16 AMD Ryzen 3 NVIDIA GTX 1060 2020-11-15

Рисунок 38 — Использование оператора «NOT IN»

Query

Query History

1

▼

SELECT * FROM PC

2

WHERE cpu_model LIKE '%AMD%'

3

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	pc_id [PK] integer	model character varying (50)	ram_size integer	cpu_model character varying (50)	gpu_model character varying (50)	purchase_date date
1	2	PC2	32	AMD Ryzen 7	NVIDIA RTX 3060	2023-02-10
2	4	PC4	8	AMD Ryzen 5	NVIDIA GTX 1050	2021-07-25
3	7	PC7	16	AMD Ryzen 9	NVIDIA RTX 3070	2023-07-21
4	9	PC9	16	AMD Ryzen 3	NVIDIA GTX 1060	2020-11-15

Рисунок 39 — Использование оператора «LIKE»

Query

Query History

1

▼

SELECT * FROM PC

2

WHERE cpu_model NOT LIKE '%AMD%'

3

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	pc_id [PK] integer	model character varying (50)	ram_size integer	cpu_model character varying (50)	gpu_model character varying (50)	purchase_date date
1	1	PC1	16	Intel i5	NVIDIA GTX 1650	2022-01-01
2	3	PC3	16	Intel i7	NVIDIA GTX 1660	2022-03-15
3	5	PC5	16	Intel i5	AMD RX 580	2022-05-30
4	6	PC6	32	Intel i9	NVIDIA RTX 3080	2023-06-12
5	8	PC8	8	Intel i3	NVIDIA GTX 950	2021-09-10
6	10	PC10	32	Intel i7	NVIDIA RTX 2070	2022-12-31

Рисунок 40 — Использование оператора «NOT LIKE»

4.2 Изменение данных в таблице

Для выполнения данного задания, была создана таблица по примеру уже существующей в БД таблицы, но с другим названием.

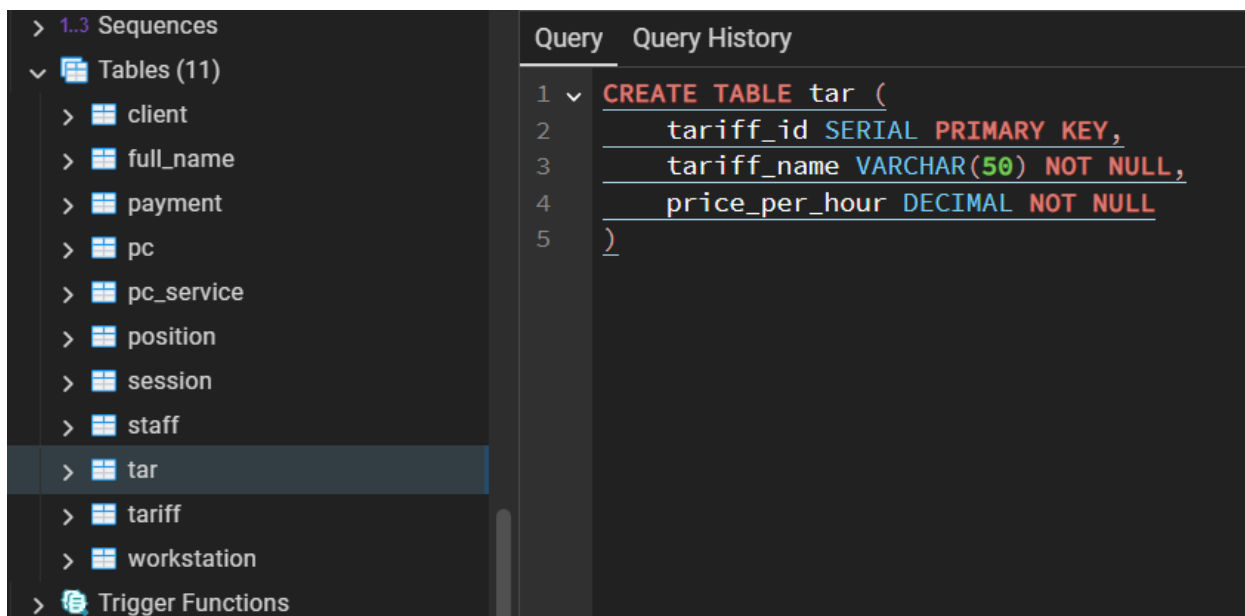


Рисунок 41 — Создание таблицы «tar»

```
pc_club=# INSERT INTO tar (tariff_name, price_per_hour) VALUES  
pc_club-# ('Standard', 5.00),  
pc_club-# ('Premium', 10.00),  
pc_club-# ('VIP', 15.00),  
pc_club-# ('Economy', 3.00),  
pc_club-# ('Elite', 18.00);  
INSERT 0 5
```

Рисунок 42 — Заполнение таблицы «tar»

Query Query History	
1	<code>SELECT * FROM tar</code>
Data Output Messages Notifications	
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>	
	<div> <div>tariff_id</div> <div>[PK] integer</div> <div>✎</div> </div>
	<div> <div>tariff_name</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>price_per_hour</div> <div>numeric</div> <div>✎</div> </div>
1	1 Standard 5.00
2	2 Premium 10.00
3	3 VIP 15.00
4	4 Economy 3.00
5	5 Elite 18.00

Рисунок 43 — Содержание таблицы «tar»

Для добавления столбцов в таблицу используется оператор «ALTER TABLE имя_таблицы ADD COLUMN новый_столбец». Так столбец появится в конце таблицы.

Query Query History	
1	<code>ALTER TABLE tar ADD COLUMN bonus numeric;</code>
2	
3	<code>SELECT * FROM tar</code>
Data Output Messages Notifications	
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>	
	<div> <div>tariff_id</div> <div>[PK] integer</div> <div>✎</div> </div>
	<div> <div>tariff_name</div> <div>character varying (50)</div> <div>✎</div> </div>
	<div> <div>price_per_hour</div> <div>numeric</div> <div>✎</div> </div>
	<div> <div>bonus</div> <div>numeric</div> <div>✎</div> </div>
1	1 Standard 5.00 [null]
2	2 Premium 10.00 [null]
3	3 VIP 15.00 [null]
4	4 Economy 3.00 [null]
5	5 Elite 18.00 [null]

Рисунок 44 — Добавление столбца «bonus» в таблицу «tar»

Установим значения в столбце «bonus» по каждому «tariff_id». Используя оператор «UPDATE».

Query

Query History

1

▼

UPDATE tar

2

SET bonus = CASE

3

WHEN tariff_id = 1 THEN 1.5

4

WHEN tariff_id = 2 THEN 2.0

5

WHEN tariff_id = 3 THEN 2.5

6

WHEN tariff_id = 4 THEN 1.0

7

WHEN tariff_id = 5 THEN 0.5

8

END

9

WHERE tariff_id IN (1, 2, 3, 4, 5);

10

11

12

SELECT * FROM tar;

Data Output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

SQL

	tariff_id [PK] integer	tariff_name character varying (50)	price_per_hour numeric	bonus numeric
1	1	Standard	5.00	1.5
2	2	Premium	10.00	2.0
3	3	VIP	15.00	2.5
4	4	Economy	3.00	1.0
5	5	Elite	18.00	0.5

Рисунок 45 — Заполнение столбца «bonus» используя оператор «UPDATE»

Установим бонус «10» в столбце «bonus» тарифам, цена которых больше 10.

Query

Query History

1

▼

2

3

4

5

UPDATE

tar

SET

bonus

=

10

WHERE

tariff_id

IN

(3, 5);

SELECT

*

FROM

tar;

Data Output

Messages

Notifications

≡+

▼

▼

SQL

	tariff_id [PK] integer	tariff_name character varying (50)	price_per_hour numeric	bonus numeric
1	1	Standard	5.00	1.5
2	2	Premium	10.00	2.0
3	4	Economy	3.00	1.0
4	3	VIP	15.00	10
5	5	Elite	18.00	10

Рисунок 46 — Изменение столбца «bonus»

Query

Query History

1

ALTER TABLE tar

2

RENAME COLUMN tariff_name TO tariff_plan;

3

4

SELECT * FROM tar;

Data Output

Messages

Notifications

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	tariff_id [PK] integer	tariff_plan character varying (50)	price_per_hour numeric	bonus numeric
1	1	Standard	5.00	1.5
2	2	Premium	10.00	2.0
3	4	Economy	3.00	1.0
4	3	VIP	15.00	10
5	5	Elite	18.00	10

Рисунок 47 — Изменение названия столбца «tariff_name»

Удалим из таблицы все данные, которые соответствуют условию

«tariff_id = (2, 4)». Для этого используется оператор «DELETE».

Query		Query History		
1	▼	DELETE FROM tar		
2		WHERE tariff_id IN (2, 4);		
3				
4		SELECT * FROM tar		
5				

Data Output		Messages		Notifications	
≡+	📄	▼	📋	▼	🗑️
			🗄️	⬇️	📈
					SQL

	tariff_id	tariff_plan	price_per_hour	bonus
	[PK] integer	character varying (50)	numeric	numeric
1	1	Standard	5.00	1.5
2	3	VIP	15.00	10
3	5	Elite	18.00	10

Рисунок 48 — Удаление данных с указанием условия

4.3 Выборка данных при помощи различных операций

Query		Query History		
1	▼	SELECT * FROM Workstation		
2		WHERE pc_id > 4 AND pc_id < 6;		
3				
4				

Data Output		Messages		Notifications	
≡+	📄	▼	📋	▼	🗑️
			🗄️	⬇️	📈
					SQL

	workstation_id	pc_id	location
	[PK] integer	integer	character varying (50)
1	5	5	Room 105

Рисунок 49 — Операция соединения

Query		Query History		Scratch Pad	
1	SELECT * FROM client				
2	INNER JOIN c ON client.full_name_id = c.full_name_id;				
3					
Data Output					
Messages					
Notifications					
	client_id	full_name_id	birth_date	phone_number	email
	integer	integer	date	character varying	character varying (50)
1	1	1	1985-05-12	1234567890	john.doe@example.com
2	3	3	1988-03-15	1122334455	sam.brown@example.com
3	5	5	1980-12-25	6677889900	chris.lee@example.com
4	7	7	1983-01-05	8899001122	michael.miller@example.com
5	9	9	1981-09-14	1122445566	david.moore@example.com

Рисунок 50 — Операция объединения «INNER JOIN»

Query		Query History		Scratch Pad	
1	SELECT * FROM client				
2	RIGHT JOIN c ON client.full_name_id = c.full_name_id;				
3					
Data Output					
Messages					
Notifications					
	client_id	full_name_id	birth_date	phone_number	email
	integer	integer	date	character var	character varying (50)
1	1	1	1985-05-12	12345678...	john.doe@example....
2	3	3	1988-03-15	11223344...	sam.brown@exam...
3	5	5	1980-12-25	66778899...	chris.lee@example....
4	7	7	1983-01-05	88990011...	michael.miller@exa...
5	9	9	1981-09-14	11224455...	david.moore@exa...

Рисунок 51 — Операция объединения «RIGHT JOIN»

Query		Query History		Scratch Pad	
1	SELECT	*	FROM client		
2	LEFT JOIN	c	ON client.full_name_id = c.full_name_id;		
3					

Data Output		Messages		Notifications	
SQL					

	client_id integer	full_name_id integer	birth_date date	phone_number character varying (10)	email character varying (50)	client_id integer	full_name_id integer	birth_date date	phone_number character varying (10)	email character varying (50)
1	1	1	1985-05-12	1234567890	john.doe@example.com	1	1	1985-05-12	1234567890	john.doe@example.com
2	2	2	1990-06-22	0987654321	jane.smith@example.com	[null]	[null]	[null]	[null]	[null]
3	3	3	1988-03-15	1122334455	sam.brown@example.com	2	3	1988-03-15	1122334455	sam.brown@example.com
4	4	4	1995-11-30	5566778899	alex.johnson@example.com	[null]	[null]	[null]	[null]	[null]
5	5	5	1980-12-25	6677889900	chris.lee@example.com	3	5	1980-12-25	6677889900	chris.lee@example.com
6	6	6	1992-08-18	7788990011	emily.davis@example.com	[null]	[null]	[null]	[null]	[null]
7	7	7	1983-01-05	8899001122	michael.miller@example.com	4	7	1983-01-05	8899001122	michael.miller@example.com
8	8	8	1997-07-20	9900112233	sara.wilson@example.com	[null]	[null]	[null]	[null]	[null]
9	9	9	1981-09-14	1122445566	david.moore@example.com	5	9	1981-09-14	1122445566	david.moore@example.com
10	10	10	1999-10-05	2233556677	laura.taylor@example.com	[null]	[null]	[null]	[null]	[null]

Рисунок 52 — Операция объединения «LEFT JOIN»

Query		Query History	
1	SELECT	full_name_id, birth_date, phone_number, email	FROM client
2	UNION		
3	SELECT	full_name_id, birth_date, phone_number, email	FROM c;
4			

Data Output		Messages		Notifications	
SQL					

	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1	10	1999-10-05	2233556677	laura.taylor@example.com
2	8	1997-07-20	9900112233	sara.wilson@example.com
3	9	1981-09-14	1122445566	david.moore@example.com
4	4	1995-11-30	5566778899	alex.johnson@example.com
5	2	1990-06-22	0987654321	jane.smith@example.com
6	3	1988-03-15	1122334455	sam.brown@example.com
7	6	1992-08-18	7788990011	emily.davis@example.com
8	7	1983-01-05	8899001122	michael.miller@example.com
9	5	1980-12-25	6677889900	chris.lee@example.com
10	1	1985-05-12	1234567890	john.doe@example.com

Рисунок 53 — Операция объединения «UNION»

1

2

3

4

5

SELECT * FROM client

WHERE EXISTS (

SELECT 1 FROM c WHERE client.full_name_id = c.full_name_id

);

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1	1	1	1985-05-12	1234567890	john.doe@example.com
2	3	3	1988-03-15	1122334455	sam.brown@example.com
3	5	5	1980-12-25	6677889900	chris.lee@example.com
4	7	7	1983-01-05	8899001122	michael.miller@example.com
5	9	9	1981-09-14	1122445566	david.moore@example.com

Рисунок 54 — Операция пересечения «EXISTS»

Query

Query History

1

2

3

4

5

SELECT * FROM client

WHERE NOT EXISTS (

SELECT 1 FROM c WHERE client.full_name_id = c.full_name_id

);

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	client_id [PK] integer	full_name_id integer	birth_date date	phone_number character varying (20)	email character varying (50)
1	2	2	1990-06-22	0987654321	jane.smith@example.com
2	4	4	1995-11-30	5566778899	alex.johnson@example.c...
3	6	6	1992-08-18	7788990011	emily.davis@example.com
4	8	8	1997-07-20	9900112233	sara.wilson@example.com
5	10	10	1999-10-05	2233556677	laura.taylor@example.com

Рисунок 55 — Операция разности «NOT EXISTS»

Query Query History

```

1  SELECT
2      MAX(birth_date) max_birth_date,
3      MIN(birth_date) AS min_birth_date,
4      AVG(EXTRACT(YEAR FROM AGE(birth_date))) AS avg_age,
5      SUM(EXTRACT(YEAR FROM AGE(birth_date))) AS total_years
6  FROM client;
7

```

Data Output Messages Notifications

	max_birth_date date	min_birth_date date	avg_age numeric	total_years numeric
1	1999-10-05	1980-12-25	34.8000000000000000	348

Рисунок 56 — Агрегирующие функции «MAX(), MIN(), AVG(), SUM()»

Query Query History

```

1  SELECT COUNT(*) AS total_clients FROM client
2  WHERE client_id >=3
3

```

Data Output Messages Notifications

	total_clients bigint
1	8

Рисунок 57 — Агрегирующая функция «COUNT()»

Query

Query History

1

2

3

4

5

SELECT EXTRACT(YEAR FROM birth_date)

AS birth_year, COUNT(*) AS count_clients

FROM client

GROUP BY EXTRACT(YEAR FROM birth_date);

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	birth_year numeric 🔒	count_clients bigint 🔒
1	1981	1
2	1980	1
3	1997	1
4	1992	1
5	1983	1
6	1990	1
7	1995	1
8	1985	1
9	1988	1
10	1999	1

Рисунок 58 — Операция группировки «GROUP BY»

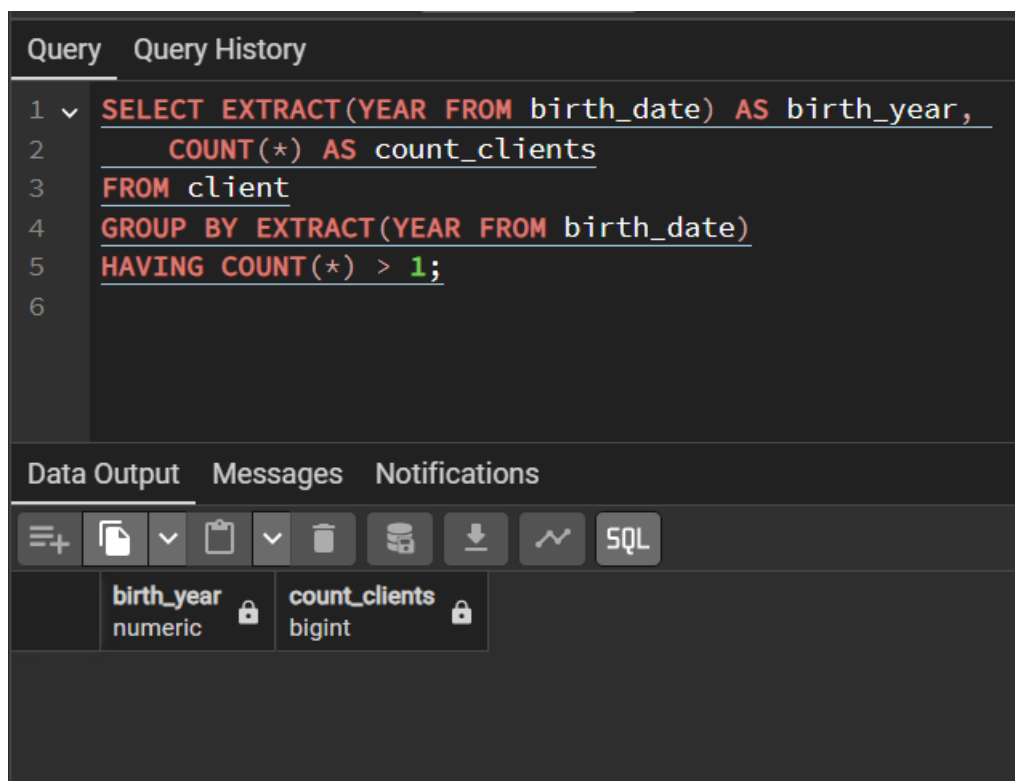


Рисунок 59 — Операция группировки с конструкцией «HAVING»
(Ничего не вернуло, т.к. у нас все разного года)

Query		Query History	
1	SELECT *	FROM client	
2	ORDER BY birth_date	DESC;	
3			
Data Output		Messages	
		Notifications	
	client_id [PK] integer	full_name_id integer	birth_date date
1	10	10	1999-10-05
2	8	8	1997-07-20
3	4	4	1995-11-30
4	6	6	1992-08-18
5	2	2	1990-06-22
6	3	3	1988-03-15
7	1	1	1985-05-12
8	7	7	1983-01-05
9	9	9	1981-09-14
10	5	5	1980-12-25

Рисунок 60 — Операция сортировки «ORDER BY»

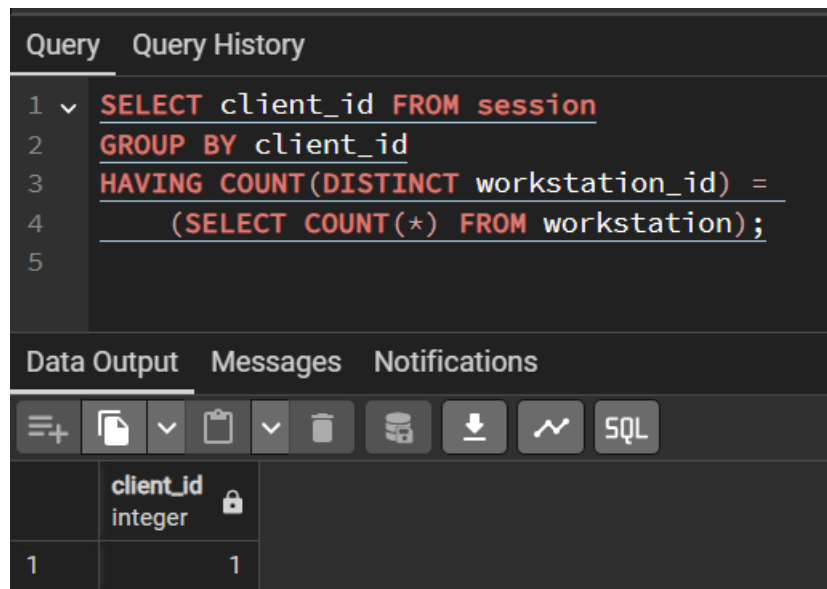


Рисунок 61 — Операция деления

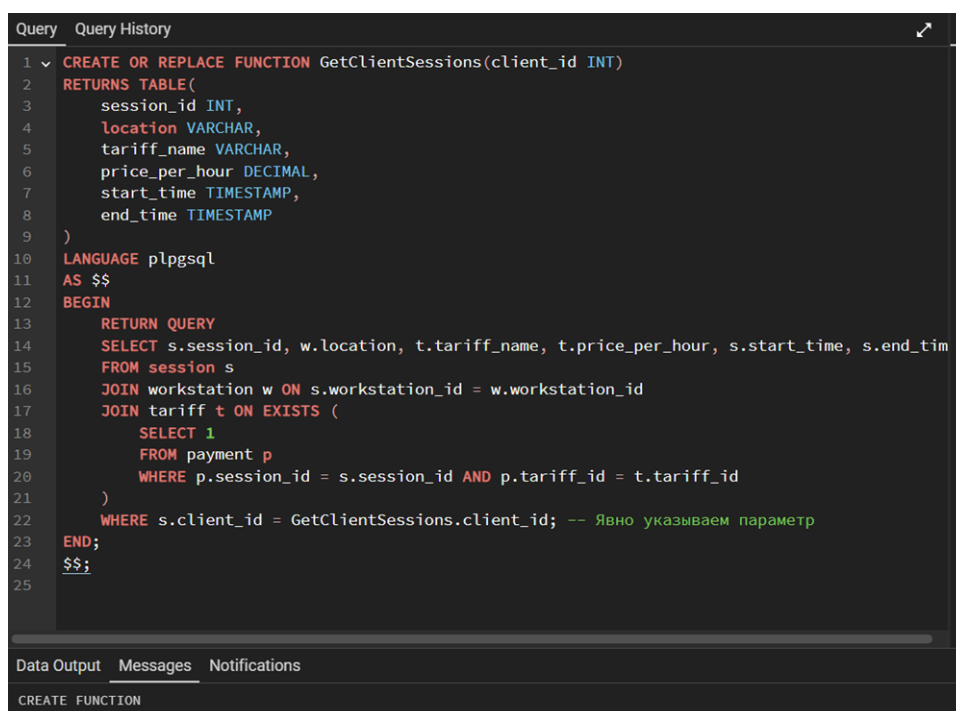
Этот запрос вернет клиента, который использовал все доступные рабочие станции.

5 Выборка с помощью процедур, функций и триггеров

5.1 Хранимые процедуры

Хранимые процедуры, функции и триггеры вводятся в базу данных для обеспечения бизнес-логики приложения на уровне серверной его компоненты.

Обычно хранимые процедуры и функции представляют собой утилиты, которые определенным образом обрабатывают данные или реализуют достаточно сложный алгоритм вычисления некоторых показателей.



```
1 CREATE OR REPLACE FUNCTION GetClientSessions(client_id INT)
2 RETURNS TABLE(
3     session_id INT,
4     location VARCHAR,
5     tariff_name VARCHAR,
6     price_per_hour DECIMAL,
7     start_time TIMESTAMP,
8     end_time TIMESTAMP
9 )
10 LANGUAGE plpgsql
11 AS $$
12 BEGIN
13     RETURN QUERY
14     SELECT s.session_id, w.location, t.tariff_name, t.price_per_hour, s.start_time, s.end_time
15     FROM session s
16     JOIN workstation w ON s.workstation_id = w.workstation_id
17     JOIN tariff t ON EXISTS (
18         SELECT 1
19         FROM payment p
20         WHERE p.session_id = s.session_id AND p.tariff_id = t.tariff_id
21     )
22     WHERE s.client_id = GetClientSessions.client_id; -- Явно указываем параметр
23 END;
24 $$;
```

Рисунок 62 — Хранимая процедура «GetClientSessions»

Эта хранимая процедура «GetClientSessions» предназначена для получения данных о сессиях клиента, включая информацию о рабочей станции и тарифе.

Входные параметры:

- «client_id» (тип: INT) — идентификатор клиента.

Описание логики:

1. Основной запрос: Процедура выполняет SQL-запрос, который соединяет таблицы session, workstation и tariff.

2. Выборка включает следующие данные:

- session_id
- location
- tariff_name
- price_per_hour
- start_time
- end_time



The screenshot shows a database query interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query: `SELECT * FROM GetClientSessions(1);`. Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. The table has seven columns: session_id, location, tariff_name, price_per_hour, start_time, and end_time. The data types for each column are listed in the header row. The first row of data shows session_id 1, location 'Room 102', tariff_name 'Deluxe', price_per_hour 12.00, start_time '2024-09-30 10:00:00', and end_time '2024-09-30 12:00:00'.

	session_id integer	location character varying	tariff_name character varying	price_per_hour numeric	start_time timestamp without time zone	end_time timestamp without time zone
1	1	Room 102	Deluxe	12.00	2024-09-30 10:00:00	2024-09-30 12:00:00

Рисунок 63 — Пример работы хранимой процедуры «GetClientSessions»

```
Query  Query History
1  CREATE OR REPLACE FUNCTION GetTariffStatistics()
2  RETURNS TABLE(
3      tariff_name VARCHAR,
4      total_usage BIGINT,
5      total_revenue NUMERIC,
6      average_revenue NUMERIC
7  )
8  LANGUAGE plpgsql
9  AS $$
10 BEGIN
11     RETURN QUERY
12     SELECT
13         t.tariff_name,
14         COUNT(p.payment_id) AS total_usage,
15         SUM(p.payment_amount) AS total_revenue,
16         AVG(p.payment_amount) AS average_revenue
17     FROM tariff t
18     LEFT JOIN payment p ON t.tariff_id = p.tariff_id
19     GROUP BY t.tariff_name;
20 END;
21 $$;
22

Data Output  Messages  Notifications
CREATE FUNCTION
```

Рисунок 64 — Хранимая процедура «GetTariffStatistics»

Данная хранимая процедура «GetTariffStatistics» возвращает статистику по тарифам, включая количество их использования, общую выручку и среднюю стоимость.

1. Основной запрос: Процедура выполняет SQL-запрос, который соединяет таблицы tariff и payment.
2. Выборка включает следующие данные: для каждого тарифа вычисляются количество платежей, общая сумма платежей и средняя стоимость
 - tariff_name
 - total_usage
 - total_revenue
3. average_revenue

Query

Query History

1

2

SELECT * FROM GetTariffStatistics();

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	tariff_name character varying	total_usage bigint	total_revenue numeric	average_revenue numeric
1	Deluxe	1	35.00	35.0000000000000000
2	Basic	1	10.00	10.0000000000000000
3	VIP	1	30.00	30.0000000000000000
4	Standard	2	515.00	257.5000000000000000
5	Advanced	1	24.00	24.0000000000000000
6	Economy	1	8.00	8.0000000000000000
7	Pro	1	40.00	40.0000000000000000
8	Student	1	50.00	50.0000000000000000
9	Elite	1	45.00	45.0000000000000000
10	Premium	1	20.00	20.0000000000000000

Рисунок 65 — Пример работы хранимой процедуры «GetTariffStatistics»

```
Query  Query History
1  ✓ CREATE OR REPLACE FUNCTION GetStaffSessions(input_staff_id INT)
2  RETURNS TABLE(
3      session_id INT,
4      client_id INT,
5      client_email VARCHAR,
6      workstation_location VARCHAR,
7      start_time TIMESTAMP,
8      end_time TIMESTAMP
9  )
10 LANGUAGE plpgsql
11 AS $$
12 BEGIN
13     RETURN QUERY
14     SELECT
15         s.session_id,
16         c.client_id,
17         c.email AS client_email,
18         w.location AS workstation_location,
19         s.start_time,
20         s.end_time
21     FROM session s
22     JOIN workstation w ON s.workstation_id = w.workstation_id
23     JOIN client c ON s.client_id = c.client_id
24     WHERE w.pc_id IN (
25         SELECT pc_service.pc_id
26         FROM pc_service
27         WHERE pc_service.staff_id = input_staff_id
28     );
29 END;
```

Data Output Messages Notifications

CREATE FUNCTION

Рисунок 66 — Хранимая процедура «GetOrderStatistics»

1. Основной запрос: Процедура выполняет SQL-запрос предназначена для получения информации о сессиях, связанных с сотрудником, обслуживающим рабочие станции.

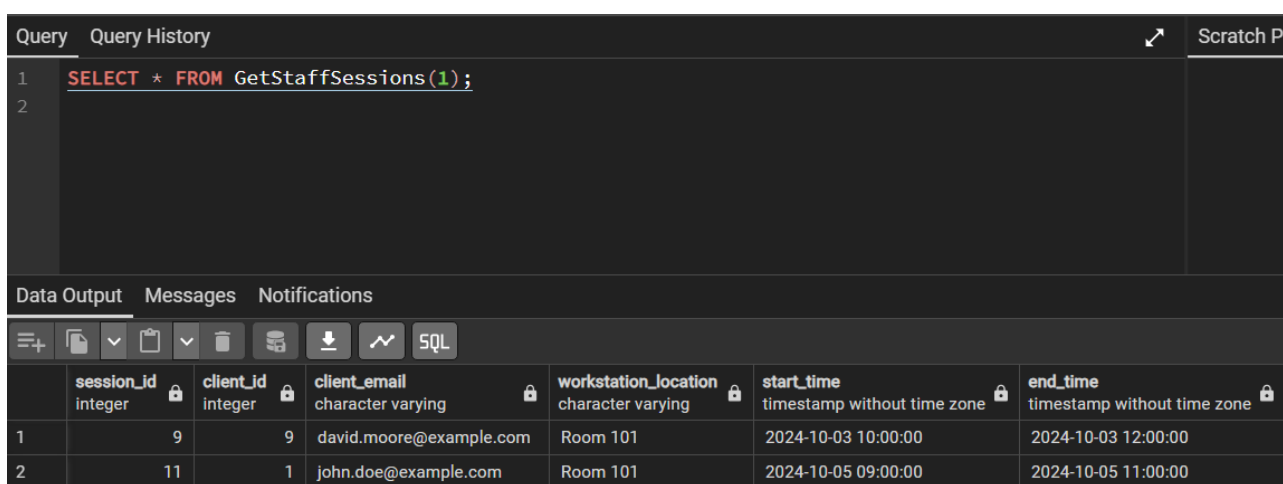
Входные параметры:

- input_staff_id (тип: INT) — идентификатор сотрудника

Описание логики:

1. Основной запрос: Процедура выполняет SQL-, соединяющий таблицы session, workstation, client и pc_service.

2. Выборка включает следующие данные, де рабочие станции обслуживаются указанным сотрудником:
- session_id
 - client_id
 - client_email
 - workstation_location
 - start_time
 - end_time

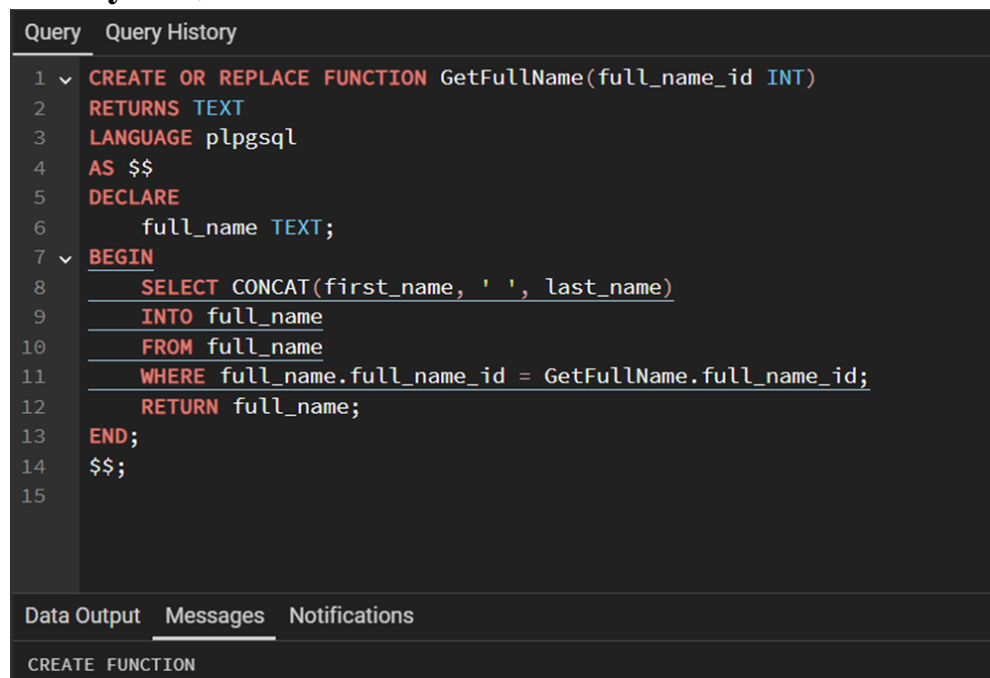


The screenshot shows a database query interface. At the top, there's a 'Query' tab with a text area containing the SQL query: `SELECT * FROM GetStaffSessions(1);`. Below the query area, there's a 'Data Output' tab showing the results of the query. The results are displayed in a table with 7 columns: session_id, client_id, client_email, workstation_location, start_time, and end_time. The table has 2 rows of data.

	session_id integer	client_id integer	client_email character varying	workstation_location character varying	start_time timestamp without time zone	end_time timestamp without time zone
1	9	9	david.moore@example.com	Room 101	2024-10-03 10:00:00	2024-10-03 12:00:00
2	11	1	john.doe@example.com	Room 101	2024-10-05 09:00:00	2024-10-05 11:00:00

Рисунок 67 — Пример работы хранимой процедуры «GetStaffSessions»

5.2 Функции



```
Query  Query History
1  CREATE OR REPLACE FUNCTION GetFullName(full_name_id INT)
2  RETURNS TEXT
3  LANGUAGE plpgsql
4  AS $$
5  DECLARE
6      full_name TEXT;
7  BEGIN
8      SELECT CONCAT(first_name, ' ', last_name)
9      INTO full_name
10     FROM full_name
11     WHERE full_name.full_name_id = GetFullName.full_name_id;
12     RETURN full_name;
13 END;
14 $$;
15

Data Output  Messages  Notifications
CREATE FUNCTION
```

Рисунок 68 — Функция « GetFullName »

Функция предназначена для получения полного имени пользователя (имя и фамилия) из таблицы `full_name` по его идентификатору. Это упрощает работу с данными, исключая необходимость регулярного объединения имени и фамилии.

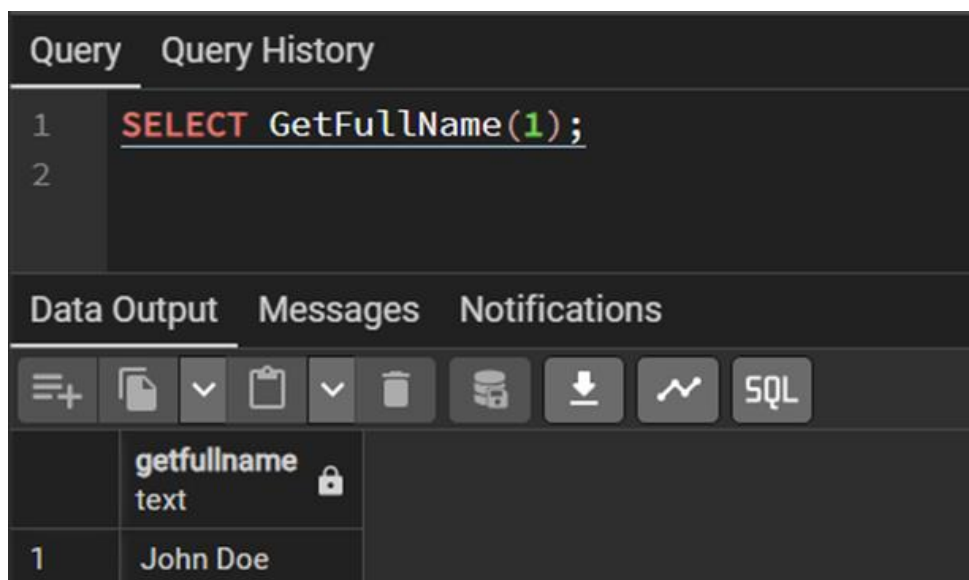


Рисунок 69 — Пример работы функции «GetFullName»

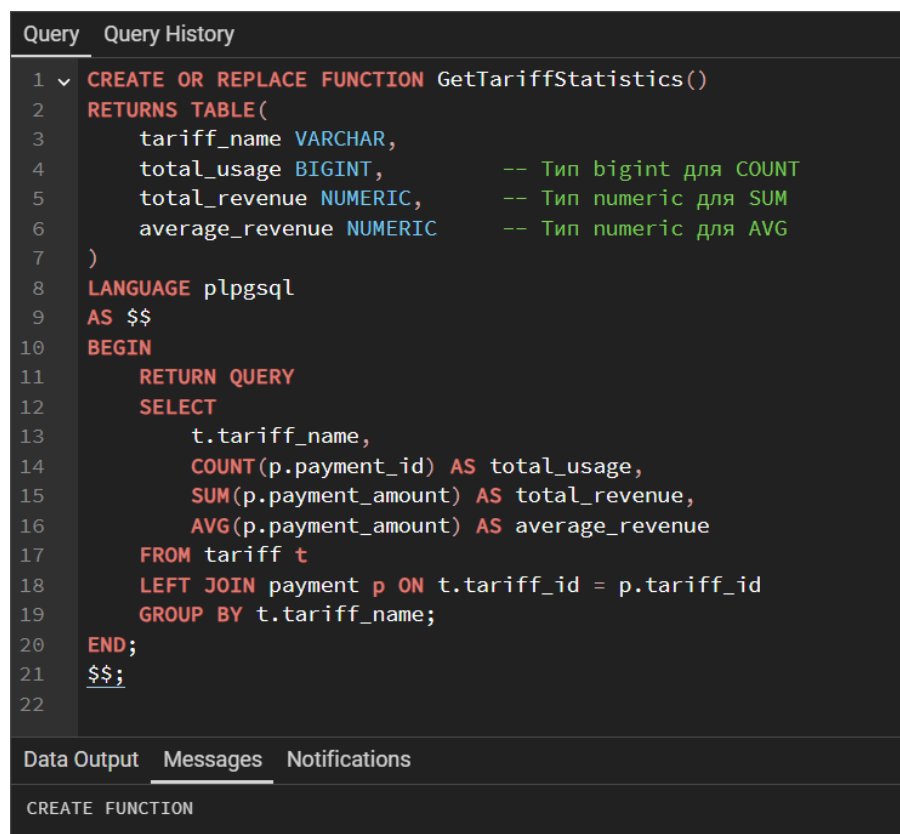


Рисунок 70 — Функция «GetTariffStatistics»

Функция возвращает статистику по каждому тарифу, включая количество его использования, общую выручку и среднюю стоимость платежей

Query

Query History

1

SELECT * FROM GetTariffStatistics();

2

Data Output

Messages

Notifications

SQL

	tariff_name character varying	total_usage bigint	total_revenue numeric	average_revenue numeric
1	Deluxe	1	35.00	35.0000000000000000
2	Basic	1	10.00	10.0000000000000000
3	VIP	1	30.00	30.0000000000000000
4	Standard	2	515.00	257.5000000000000000
5	Advanced	1	24.00	24.0000000000000000
6	Economy	1	8.00	8.0000000000000000
7	Pro	1	40.00	40.0000000000000000
8	Student	1	50.00	50.0000000000000000
9	Elite	1	45.00	45.0000000000000000
10	Premium	1	20.00	20.0000000000000000

Рисунок 71 — Пример работы функции «GetTariffStatistics»

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

Query

Query History

```
CREATE OR REPLACE FUNCTION GetStaffSessions(input_staff_id INT)
RETURNS TABLE(
    session_id INT,
    client_id INT,
    client_email VARCHAR,
    workstation_location VARCHAR,
    start_time TIMESTAMP,
    end_time TIMESTAMP
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.session_id,
        c.client_id,
        c.email AS client_email,
        w.location AS workstation_location,
        s.start_time,
        s.end_time
    FROM session s
    JOIN workstation w ON s.workstation_id = w.workstation_id
    JOIN client c ON s.client_id = c.client_id
    WHERE w.pc_id IN (
        SELECT pc_service.pc_id
        FROM pc_service
        WHERE pc_service.staff_id = input_staff_id
    );
END;
```

Data Output

Messages

Notifications

CREATE FUNCTION

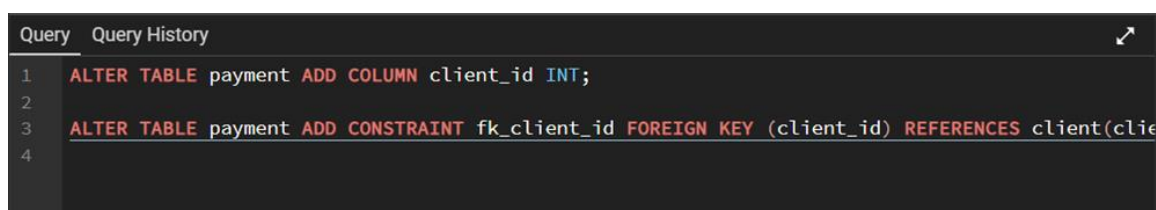
Рисунок 72 — Функция «GetStaffSessions»

Функция предназначена для получения информации о сессиях, связанных с сотрудником, обслуживающим рабочие станции.

5.3 Триггеры

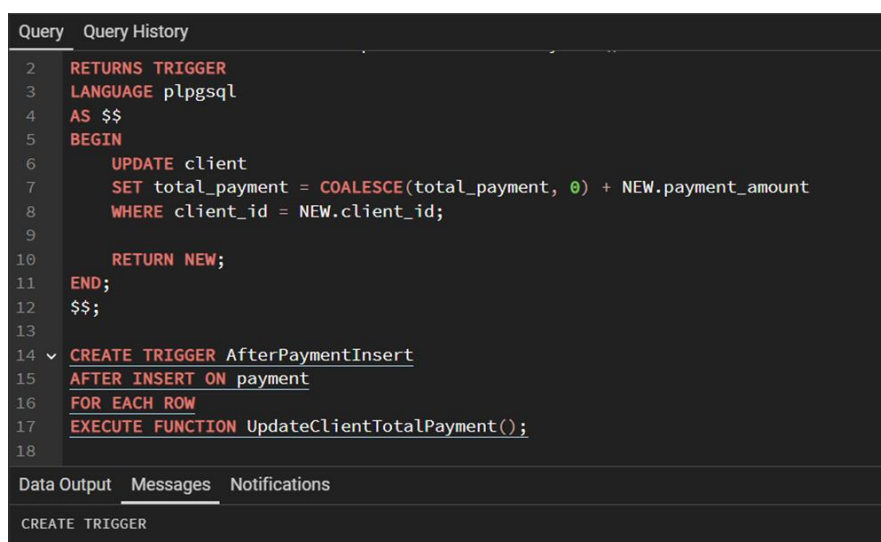
Триггеры – это частный случай хранимой процедуры, который выполняется автоматически при выполнении команд обновления данных (INSERT, DELETE, UPDATE). Триггеры привязываются к конкретным таблицам базы данных. Для каждой команды должны быть свои триггеры.

Для начала создаем новую колонку для хранения общей суммы заказов каждого клиента, при помощи команды «ALTER TABLE» и «ADD COLUMN».



```
Query Query History
1 ALTER TABLE payment ADD COLUMN client_id INT;
2
3 ALTER TABLE payment ADD CONSTRAINT fk_client_id FOREIGN KEY (client_id) REFERENCES client(client_id);
4
```

Рисунок 74 — Создание связей между таблицами



```
Query Query History
2 RETURNS TRIGGER
3 LANGUAGE plpgsql
4 AS $$
5 BEGIN
6     UPDATE client
7     SET total_payment = COALESCE(total_payment, 0) + NEW.payment_amount
8     WHERE client_id = NEW.client_id;
9
10    RETURN NEW;
11 END;
12 $$;
13
14 CREATE TRIGGER AfterPaymentInsert
15 AFTER INSERT ON payment
16 FOR EACH ROW
17 EXECUTE FUNCTION UpdateClientTotalPayment();
18
```

Data Output Messages Notifications

CREATE TRIGGER

Рисунок 75 — Триггер «AfterPaymentInsert»

Данный триггер предназначен для автоматического обновления общей суммы платежей клиента в базе данных. Функция UpdateClientTotalPayment срабатывает после добавления новой записи в таблицу payment и обновляет соответствующее значение в таблице client.

Когда происходит добавление новой строки в таблицу payment, триггер вызывает функцию UpdateClientTotalPayment, которая выполняет два действия:

1. Сначала функция обновляет значение поля total_payment в таблице client для клиента, идентификатор которого совпадает с client_id из новой записи.
2. При обновлении используется функция COALESCE, чтобы корректно

обработать случаи, когда поле `total_payment` ещё не заполнено. Значение нового платежа (`NEW.payment_amount`) добавляется к текущей сумме (`total_payment`).

В результате работы триггера в таблице `client` всегда актуально хранится общая сумма всех платежей для каждого клиента. Это позволяет оптимизировать дальнейшие запросы и отчёты, исключая необходимость повторного расчёта сумм платежей.

Триггер обеспечивает автоматическое и точное обновление данных в таблице `client`, упрощая управление финансовой информацией и поддержание её целостности.

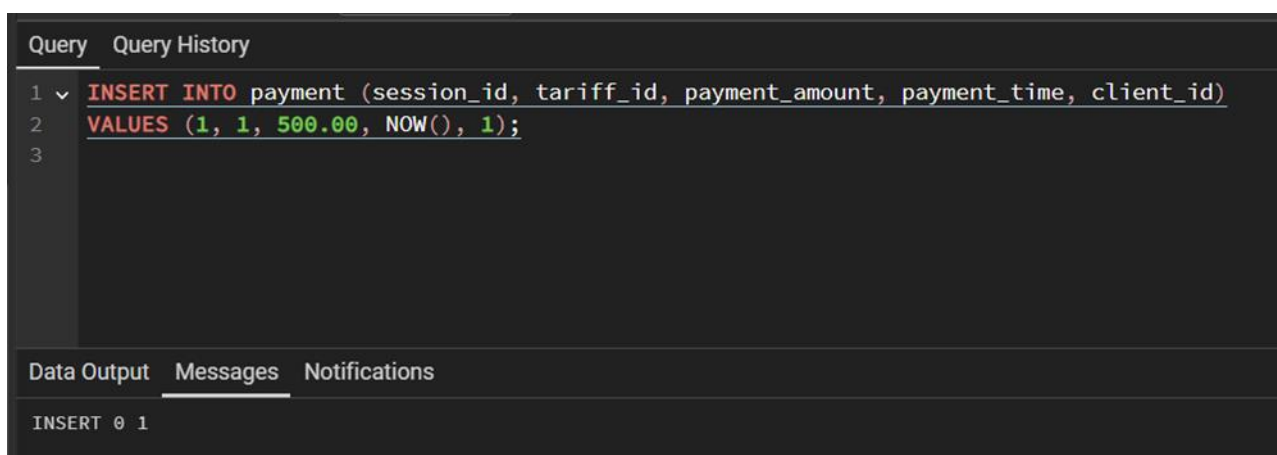


Рисунок 76 — Новая запись в таблице

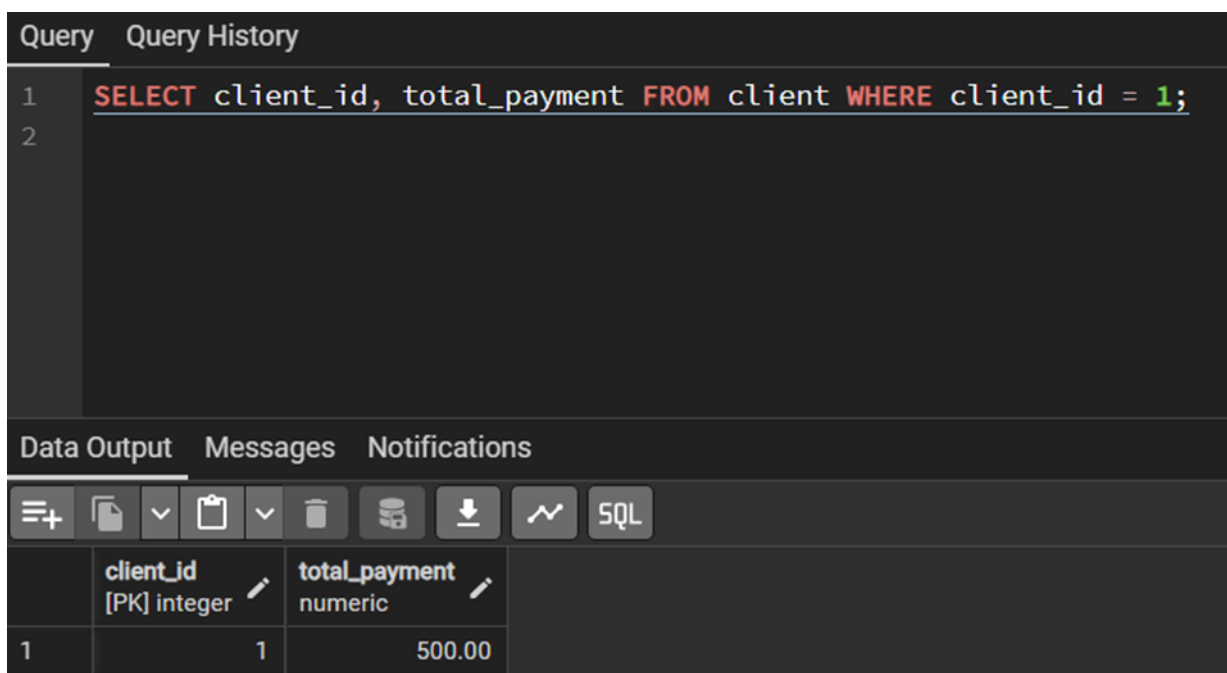
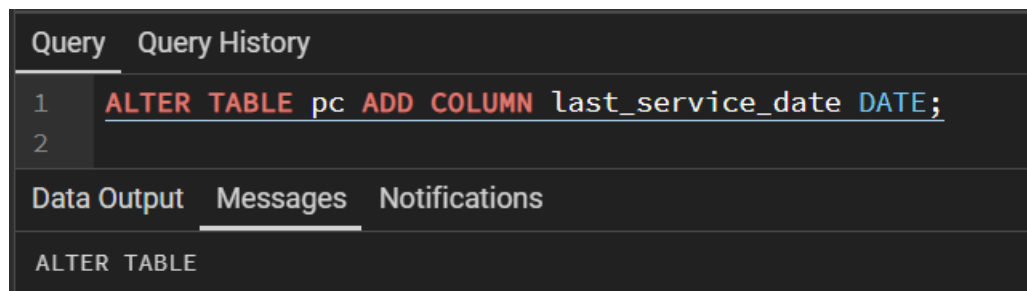
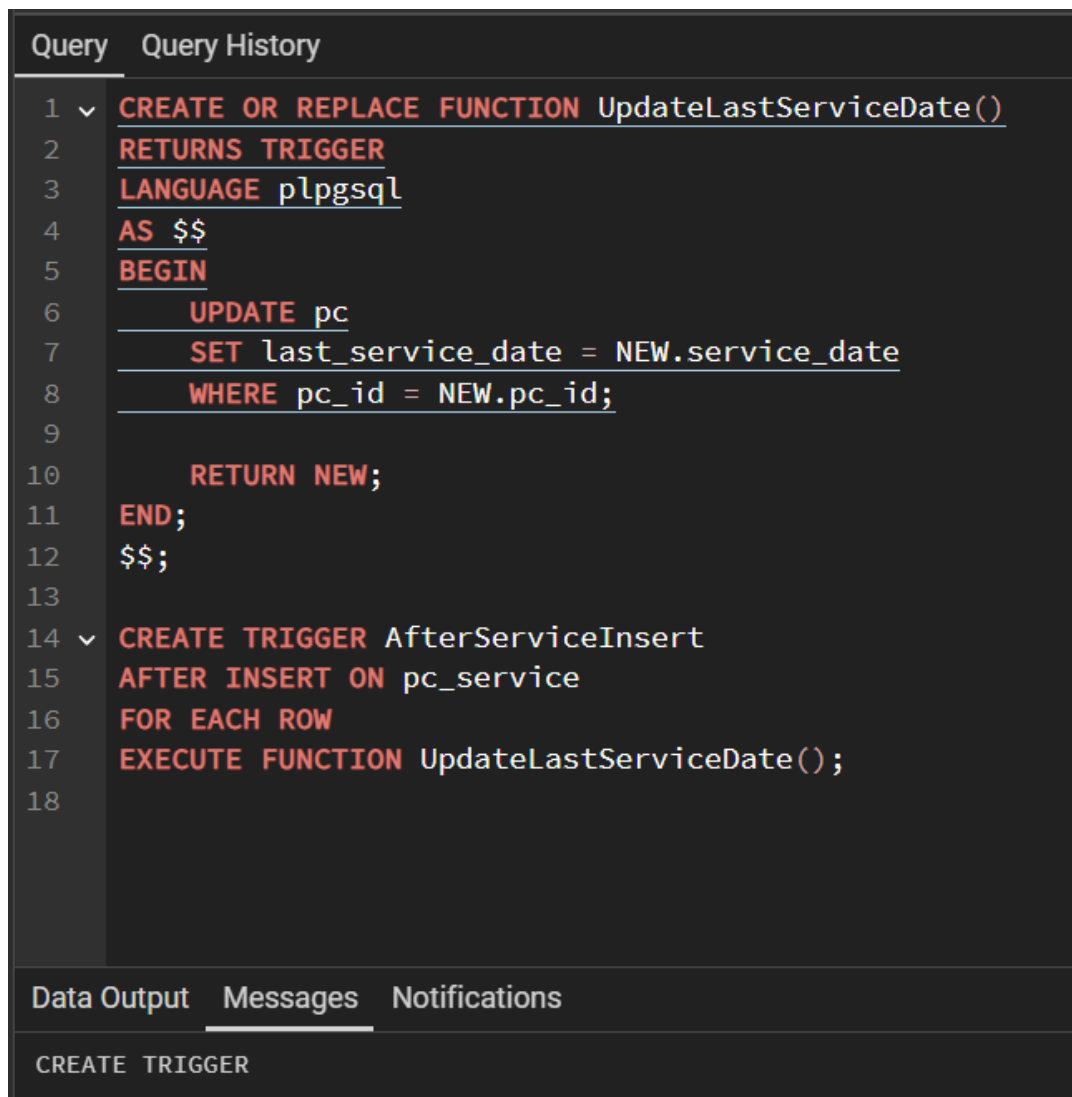


Рисунок 77 — Содержание таблицы «Client» после срабатывания триггера



```
Query  Query History
1 ALTER TABLE pc ADD COLUMN last_service_date DATE;
2
Data Output  Messages  Notifications
ALTER TABLE
```

Рисунок 78 — Добавление колонны для таблицы «РС»



```
Query  Query History
1 CREATE OR REPLACE FUNCTION UpdateLastServiceDate()
2 RETURNS TRIGGER
3 LANGUAGE plpgsql
4 AS $$
5 BEGIN
6     UPDATE pc
7     SET last_service_date = NEW.service_date
8     WHERE pc_id = NEW.pc_id;
9
10    RETURN NEW;
11 END;
12 $$;
13
14 CREATE TRIGGER AfterServiceInsert
15 AFTER INSERT ON pc_service
16 FOR EACH ROW
17 EXECUTE FUNCTION UpdateLastServiceDate();
18
Data Output  Messages  Notifications
CREATE TRIGGER
```

Рисунок 79 — Триггер «AfterServiceInsert»

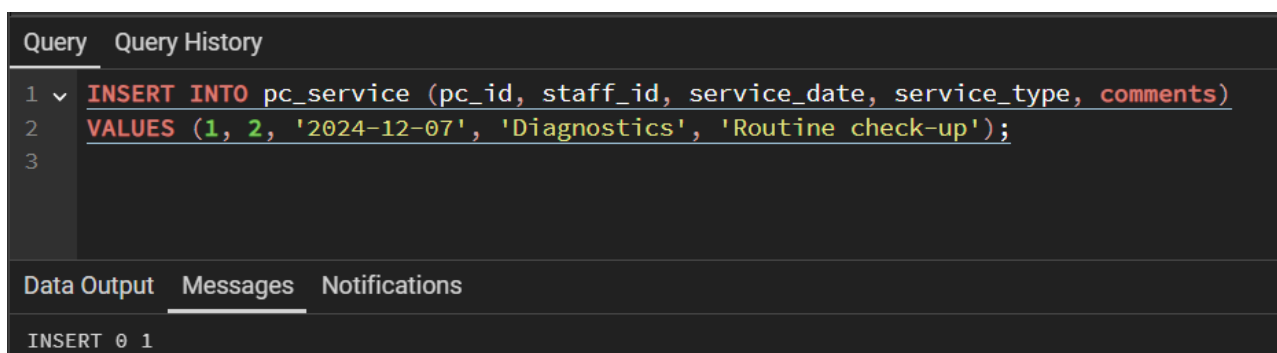
Данный триггер предназначен для автоматического обновления даты последнего обслуживания компьютера в базе данных. Функция UpdateLastServiceDate срабатывает после добавления новой записи в таблицу pc_service и обновляет соответствующее значение в таблице pc.

Когда происходит добавление новой строки в таблицу `pc_service`, триггер вызывает функцию `UpdateLastServiceDate`, которая выполняет следующие действия:

1. Функция обновляет поле `last_service_date` в таблице `pc` для компьютера, идентификатор которого совпадает с `pc_id` из новой записи.
2. Значение поля `service_date` из новой записи автоматически записывается в поле `last_service_date`.

В результате работы триггера в таблице `pc` всегда актуально хранится информация о последней дате обслуживания каждого компьютера. Это позволяет исключить дублирование данных и упростить доступ к информации о текущем состоянии оборудования.

Триггер обеспечивает автоматическое поддержание актуальности данных о техническом обслуживании, что важно для контроля состояния оборудования и планирования работ.



The screenshot shows a SQL query editor with a dark theme. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying an SQL INSERT statement. The statement is as follows:

```
1  INSERT INTO pc_service (pc_id, staff_id, service_date, service_type, comments)
2  VALUES (1, 2, '2024-12-07', 'Diagnostics', 'Routine check-up');
3
```

Below the query editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the result of the query: 'INSERT 0 1'.

Рисунок 80 — Изменяем данные в таблице «`pc_service`»

Query

Query History

1

2

3

SELECT * FROM pc

Scratch Pad

Data Output

Messages

Notifications

SQL

	pc_id [PK] integer	model character varying (50)	ram_size integer	cpu_model character varying (50)	gpu_model character varying (50)	purchase_date date	last_service_date date
1	2	PC2	32	AMD Ryzen 7	NVIDIA RTX 3060	2023-02-10	[null]
2	3	PC3	16	Intel i7	NVIDIA GTX 1660	2022-03-15	[null]
3	4	PC4	8	AMD Ryzen 5	NVIDIA GTX 1050	2021-07-25	[null]
4	5	PC5	16	Intel i5	AMD RX 580	2022-05-30	[null]
5	6	PC6	32	Intel i9	NVIDIA RTX 3080	2023-06-12	[null]
6	7	PC7	16	AMD Ryzen 9	NVIDIA RTX 3070	2023-07-21	[null]
7	8	PC8	8	Intel i3	NVIDIA GTX 950	2021-09-10	[null]
8	9	PC9	16	AMD Ryzen 3	NVIDIA GTX 1060	2020-11-15	[null]
9	10	PC10	32	Intel i7	NVIDIA RTX 2070	2022-12-31	[null]
10	1	PC1	16	Intel i5	NVIDIA GTX 1650	2022-01-01	2024-12-07

Рисунок 81 — Содержание таблицы «PC» после срабатывания триггера

Query	Query History	
1	✓ CREATE OR REPLACE FUNCTION DeleteClientSessions()	
2	RETURNS TRIGGER	
3	LANGUAGE plpgsql	
4	AS \$\$	
5	BEGIN	
6	DELETE FROM payment	
7	WHERE session_id IN (
8	SELECT session_id	
9	FROM session	
10	WHERE client_id = OLD.client_id	
11);	
12		
13	✓ DELETE FROM session	
14	WHERE client_id = OLD.client_id;	
15		
16	RETURN OLD;	
17	END;	
18	\$\$;	
19		
20	✓ CREATE TRIGGER BeforeClientDelete	
21	BEFORE DELETE ON client	
22	FOR EACH ROW	
23	EXECUTE FUNCTION DeleteClientSessions();	
24		
Data Output	Messages	Notifications
CREATE TRIGGER		

Рисунок 82 — Триггер «BeforeClientDelete»

Данный триггер предназначен для автоматического удаления связанных данных о сессиях и платежах при удалении клиента из базы данных. Функция DeleteClientSessions срабатывает перед удалением записи в таблице client и выполняет очистку зависимых данных в таблицах session и payment.

Когда происходит удаление строки из таблицы client, триггер вызывает функцию DeleteClientSessions, которая выполняет следующие действия:

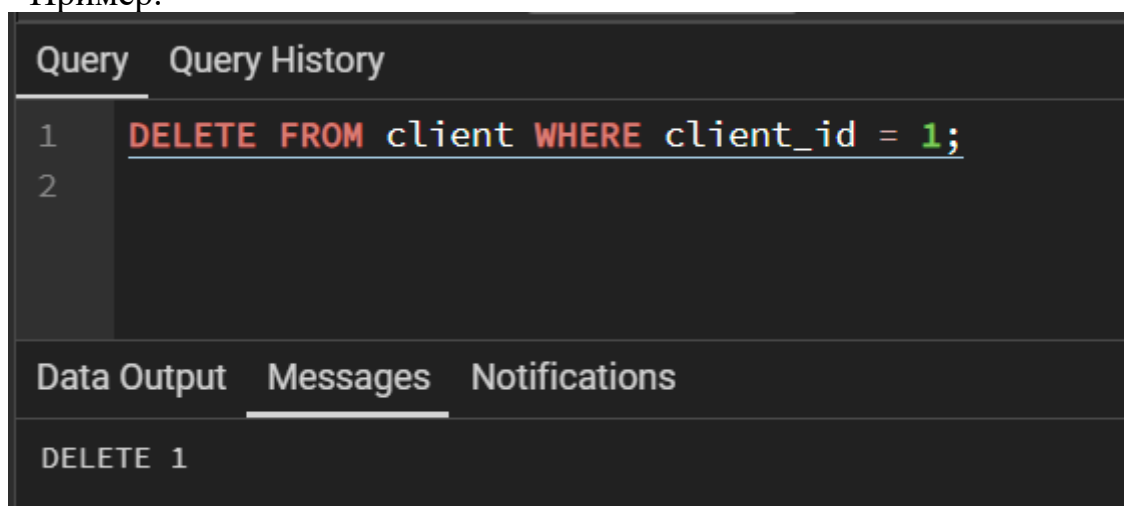
Сначала из таблицы payment удаляются все записи, связанные с сессиями указанного клиента. Это выполняется с помощью подзапроса, выбирающего все session_id для удаляемого клиента.

1. После этого из таблицы session удаляются все сессии, связанные с данным клиентом (client_id).

2. В результате работы триггера удаление клиента из таблицы client автоматически удаляет все связанные записи из таблиц payment и session, поддерживая целостность базы данных и исключая орфанные (несвязанные) записи.

Триггер гарантирует, что удаление клиента выполняется корректно, с полной очисткой всех связанных данных, что особенно важно для соблюдения целостности данных и упрощения их управления.

Пример:



The screenshot shows a database query interface with a dark theme. At the top, there are two tabs: 'Query' and 'Query History'. The 'Query' tab is active, showing a SQL statement: `DELETE FROM client WHERE client_id = 1;`. Below the query, there are two empty lines numbered 1 and 2. At the bottom, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing the result: `DELETE 1`.

Рисунок 83 — Выполняем удаление из таблицы «Client»

Query

Query History

1

2

3

SELECT * FROM session

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	session_id [PK] integer	client_id integer	workstation_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	2	2	4	2024-09-30 11:00:00	2024-09-30 13:00:00
2	3	3	3	2024-09-30 14:00:00	2024-09-30 16:00:00
3	4	4	5	2024-10-01 09:00:00	2024-10-01 11:00:00
4	5	5	6	2024-10-01 13:00:00	2024-10-01 15:00:00
5	6	6	7	2024-10-02 10:00:00	2024-10-02 12:00:00
6	7	7	8	2024-10-02 11:00:00	2024-10-02 13:00:00
7	8	8	9	2024-10-02 14:00:00	2024-10-02 16:00:00
8	9	9	1	2024-10-03 10:00:00	2024-10-03 12:00:00
9	10	10	10	2024-10-03 14:00:00	2024-10-03 16:00:00

Рисунок 84 — Содержание таблицы «Session» после срабатывания триггера

6 Оконные функции

6.1 Синтаксис оконных функций

Query		Query History
1	SELECT	
2	SUM(payment_amount) OVER () AS total_payment	
3	FROM payment;	
4		

Data Output		Messages	Notifications
+	▼	▼	SQL
	total_payment numeric		
1	242.00		
2	242.00		
3	242.00		
4	242.00		
5	242.00		
6	242.00		
7	242.00		
8	242.00		
9	242.00		

Рисунок 85 — Открытие окна при помощи инструкции «OVER()»

Query		Query History
1	SELECT	
2	SUM(payment_amount) OVER	
3	(PARTITION BY client_id) AS client_total_payment	
4	FROM payment;	

Data Output		Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	client_total_payment		
	numeric		
1	242.00		
2	242.00		
3	242.00		
4	242.00		
5	242.00		
6	242.00		
7	242.00		
8	242.00		
9	242.00		

Рисунок 86 — Применение инструкции «PARTITION BY()»

Query		Query History
1	SELECT	
2	payment_time,	
3	SUM(payment_amount) OVER	
4	(ORDER BY payment_time) AS running_total	
5	FROM payment;	

Data Output		Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	payment_time	running_total	
	timestamp without time zone	numeric	
1	2024-09-30 12:00:00	15.00	
2	2024-09-30 13:00:00	45.00	
3	2024-09-30 14:00:00	65.00	
4	2024-10-01 11:00:00	75.00	
5	2024-10-02 12:00:00	144.00	
6	2024-10-02 12:00:00	144.00	
7	2024-10-02 16:00:00	152.00	
8	2024-10-03 12:00:00	192.00	
9	2024-10-03 16:00:00	242.00	

Рисунок 87 — Применение «ORDER BY()»

6.2 Агрегатные функции

Query

Query History

1

2

3

4

5

6

SELECT

tariff_id,

AVG(payment_amount) OVER (PARTITION BY tariff_id) AS avg_payment,


COUNT(*) OVER (PARTITION BY tariff_id) AS count_payments


FROM payment;


Data Output


Messages


Notifications
























SQL

	tariff_id integer	avg_payment numeric	count_payments bigint
1	1	15.0000000000000000	1
2	2	20.0000000000000000	1
3	3	30.0000000000000000	1
4	4	8.0000000000000000	1
5	5	10.0000000000000000	1
6	6	24.0000000000000000	1
7	8	40.0000000000000000	1
8	9	45.0000000000000000	1
9	10	50.0000000000000000	1

Рисунок 88 — Применение агрегатных функций «AVG()» и «COUNT()»

Query

Query History

1  SELECT

2 SUM(payment_amount) OVER (PARTITION BY client_id) AS total_payment,

3 COUNT(*) OVER (PARTITION BY client_id) AS count_payments


4 FROM payment;

5

Data Output

Messages

Notifications





	total_payment numeric 	count_payments bigint 
1	242.00	9
2	242.00	9
3	242.00	9
4	242.00	9
5	242.00	9
6	242.00	9
7	242.00	9
8	242.00	9
9	242.00	9

Рисунок 89 — Применение агрегатных функций «SUM()» и «COUNT()»

Query

Query History

1

2

3

4

SELECT

MAX(payment_amount) OVER (PARTITION BY client_id) AS max_payment,

COUNT(*) OVER (PARTITION BY client_id) AS count_payments

FROM payment;

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑

🗄

⬇


📈

SQL

	total_payment numeric	count_payments bigint
1	242.00	9
2	242.00	9
3	242.00	9
4	242.00	9
5	242.00	9
6	242.00	9
7	242.00	9
8	242.00	9
9	242.00	9

Рисунок 90 — Применение агрегатных функций «MAX()» и «COUNT()»

Query Query History

1  SELECT


2 MIN(payment_amount) OVER (PARTITION BY client_id) AS min_payment,

3 COUNT(*) OVER (PARTITION BY client_id) AS count_payments

4 FROM payment;

5

Data Output Messages Notifications





	min_payment numeric 	count_payments bigint 
1	8.00	9
2	8.00	9
3	8.00	9
4	8.00	9
5	8.00	9
6	8.00	9
7	8.00	9
8	8.00	9
9	8.00	9

Рисунок 91 — Применение агрегатных функций «MIN ()» и «COUNT()»

6.3 Ранжирующие функции

Query

Query History

1

▼

SELECT

payment_amount,

ROW_NUMBER() OVER

(PARTITION BY client_id ORDER BY payment_amount DESC) AS row_num

FROM payment;

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	payment_amount numeric	row_num bigint
1	50.00	1
2	45.00	2
3	40.00	3
4	30.00	4
5	24.00	5
6	20.00	6
7	15.00	7
8	10.00	8
9	8.00	9

Рисунок 92 — Применение ранжирующей функции «ROW_NUMBER»

Query		Query History	
1	SELECT		
2	payment_amount,		
3	RANK() OVER		
4	(PARTITION BY client_id ORDER BY payment_amount DESC) AS rank		
5	FROM payment;		
Data Output		Messages	Notifications
	payment_amount numeric		rank bigint
1	50.00		1
2	45.00		2
3	40.00		3
4	30.00		4
5	24.00		5
6	20.00		6
7	15.00		7
8	10.00		8
9	8.00		9

Рисунок 93 — Применение ранжирующей функции «RANK»

Query		Query History	
1	SELECT		
2	payment_amount,		
3	DENSE_RANK() OVER		
4	(PARTITION BY client_id ORDER BY payment_amount DESC) AS dense_rank		
5	FROM payment;		
Data Output		Messages	Notifications
	payment_amount numeric		dense_rank bigint
1	50.00		1
2	45.00		2
3	40.00		3
4	30.00		4
5	24.00		5
6	20.00		6
7	15.00		7
8	10.00		8
9	8.00		9

Рисунок 94 — Применение ранжирующей функции «DENSE_RANK»

Query		Query History	
1	SELECT		
2	payment_amount,		
3	NTILE(4) OVER		
4	(PARTITION BY client_id ORDER BY payment_amount DESC) AS quartile		
5	FROM payment;		
6			
Data Output		Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	payment_amount numeric	quartile integer	
1	50.00	1	
2	45.00	1	
3	40.00	1	
4	30.00	2	
5	24.00	2	
6	20.00	3	
7	15.00	3	
8	10.00	4	
9	8.00	4	

Рисунок 95 — Применение ранжирующей функции «NTILE()»

6.4 Функции смещения

Query		Query History
1	▼	SELECT
2		payment_amount,
3		LAG (payment_amount) OVER
4		(PARTITION BY client_id ORDER BY payment_time) AS previous_payment
5		FROM payment;
6		
Data Output		Messages Notifications
<div><div>≡+</div><div>📄</div><div>▼</div><div>📋</div><div>🗑</div><div>🗄</div><div>⬇</div><div>📈</div><div>SQL</div></div>		
	payment_amount numeric	previous_payment numeric
1	15.00	[null]
2	30.00	15.00
3	20.00	30.00
4	10.00	20.00
5	45.00	10.00
6	24.00	45.00
7	8.00	24.00
8	40.00	8.00
9	50.00	40.00

Рисунок 96 — Применение функции смещение «LAG()»

Query		Query History
1	▼	SELECT
2		payment_amount,
3		LEAD (payment_amount) OVER
4		(PARTITION BY client_id ORDER BY payment_time) AS next_payment
5		FROM payment;
6		
Data Output		Messages Notifications
<div><div>≡+</div><div>📄</div><div>▼</div><div>📋</div><div>🗑</div><div>🗄</div><div>⬇</div><div>📈</div><div>SQL</div></div>		
	payment_amount numeric	next_payment numeric
1	15.00	30.00
2	30.00	20.00
3	20.00	10.00
4	10.00	45.00
5	45.00	24.00
6	24.00	8.00
7	8.00	40.00
8	40.00	50.00
9	50.00	[null]

Рисунок 97 — Применение функции смещение «LEAD()»

Query		Query History	
1	SELECT		
2	payment_amount,		
3	FIRST_VALUE(payment_amount) OVER		
4	(PARTITION BY client_id ORDER BY payment_time) AS first_payment		
5	FROM payment;		
6			
Data Output		Messages	Notifications
<div> <div>+</div> <div>SQL</div> </div>			
	payment_amount numeric	first_payment numeric	
1	15.00	15.00	
2	30.00	15.00	
3	20.00	15.00	
4	10.00	15.00	
5	45.00	15.00	
6	24.00	15.00	
7	8.00	15.00	
8	40.00	15.00	
9	50.00	15.00	

Рисунок 98 — Применение функции смещение «FIRST_VALUE()»

Query		Query History	
1	SELECT		
2	payment_amount,		
3	LAST_VALUE(payment_amount) OVER		
4	(PARTITION BY client_id ORDER BY payment_time		
5	ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_payment		
6	FROM payment;		
Data Output		Messages	Notifications
<div> <div>+</div> <div>SQL</div> </div>			
	payment_amount numeric	last_payment numeric	
1	15.00	50.00	
2	30.00	50.00	
3	20.00	50.00	
4	10.00	50.00	
5	45.00	50.00	
6	24.00	50.00	
7	8.00	50.00	
8	40.00	50.00	
9	50.00	50.00	

Рисунок 99 — Применение функции смещение «LAST_VALUE()»

ЗАКЛЮЧЕНИЕ

Работа с SQL показала важность его использования для управления данными и выполнения сложных операций в реляционных базах данных. SQL предоставляет мощные инструменты для создания таблиц, манипуляции данными и выполнения запросов для анализа информации.

Создание таблиц для пекарни с использованием SQL продемонстрировало необходимость нормализации данных и построения связей между ними через ключевые поля. Это обеспечило структурированность данных и их согласованность. Важным этапом стало моделирование данных с помощью диаграммы IDEF1X в MySQL Workbench, которое позволило предусмотреть связи между таблицами и избежать логических ошибок.

Заполнение таблиц данными подчеркнуло значимость правильного определения типов данных и связей между ними, что обеспечило стабильность работы базы. SQL-запросы для выборки данных по различным параметрам и их сортировки с использованием основных команд позволили эффективно обрабатывать информацию, фильтруя и анализируя её по заданным критериям.

Использование хранимых процедур, функций и триггеров автоматизировало процессы, связанные с обновлением данных и выполнением операций, таких как контроль заказов и обновление запасов. Оконные функции, включая агрегатные, ранжирующие и смещающие, были полезны для проведения сложной аналитики данных, что важно для анализа продаж и управления запасами.

Таким образом, SQL — это мощный инструмент для управления данными, который требует тщательного моделирования структуры базы перед началом разработки. Грамотное использование SQL улучшает целостность данных, упрощает анализ и автоматизирует бизнес-процессы, как в случае с пекарней.