



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

---

**Институт информационных технологий (ИИТ)  
Кафедра цифровой трансформации (ЦТ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**  
по дисциплине «Разработка баз данных»

**Практическое занятие № 3**

Студенты группы                    ИКБО-66-23 Смирнов А.Ю.

---

(подпись)

Ассистент                            Копылова Я.А.

---

(подпись)

Отчет представлен                    «\_\_\_»\_\_\_\_2025 г.

Москва 2025 г.

# ПРАКТИЧЕСКАЯ РАБОТА №3.

## Условная логика, подзапросы и обобщенные табличные выражения (CTE) в POSTGRES PRO

### Задание 1: Использование оператора CASE

#### 1.1 Поисковое выражение CASE для категоризации данных

Составить запрос, использующий поисковое выражение CASE для категоризации данных по какому-либо числовому признаку из вашей БД (например, цена, количество, возраст). Запрос должен содержать не менее трех условий WHEN и ветку ELSE.

#### 1.2 CASE внутри агрегатной функции

Составить запрос, в котором оператор CASE используется внутри агрегатной функции (например, SUM или COUNT) для выполнения условной агрегации.

### Задание 2: Использование подзапросов (часть 1)

#### 2.1 Скалярный подзапрос

Найти все записи в таблице, у которых значение в некотором числовом столбце превышает среднее (или максимальное/минимальное) значение по этому столбцу.

#### 2.2 Многострочный подзапрос с IN

Вывести информацию из одной таблицы на основе идентификаторов, полученных из связанной таблицы по определенному критерию (в данном случае, обязательно по дате).

### Задание 2: Использование подзапросов (часть 2)

#### 2.3 Коррелированный подзапрос с EXISTS

Найти все записи из родительской таблицы, для которых существует хотя бы одна связанная запись в дочерней таблице, удовлетворяющая текстовому условию.

#### 2.4 Альтернативное решение с JOIN

Решите задачу из пункта выше (2.3, «Коррелированный подзапрос с EXISTS»), но на этот раз с использованием оператора соединения JOIN.

### Задание 3: Использование обобщенных табличных выражений (CTE)

#### 3.1 Стандартное CTE

Переписать запрос из Задания 2.3 («Коррелированный подзапрос с EXISTS») с использованием обобщенного табличного выражения (CTE).

### 3.2 Рекурсивное СТЕ

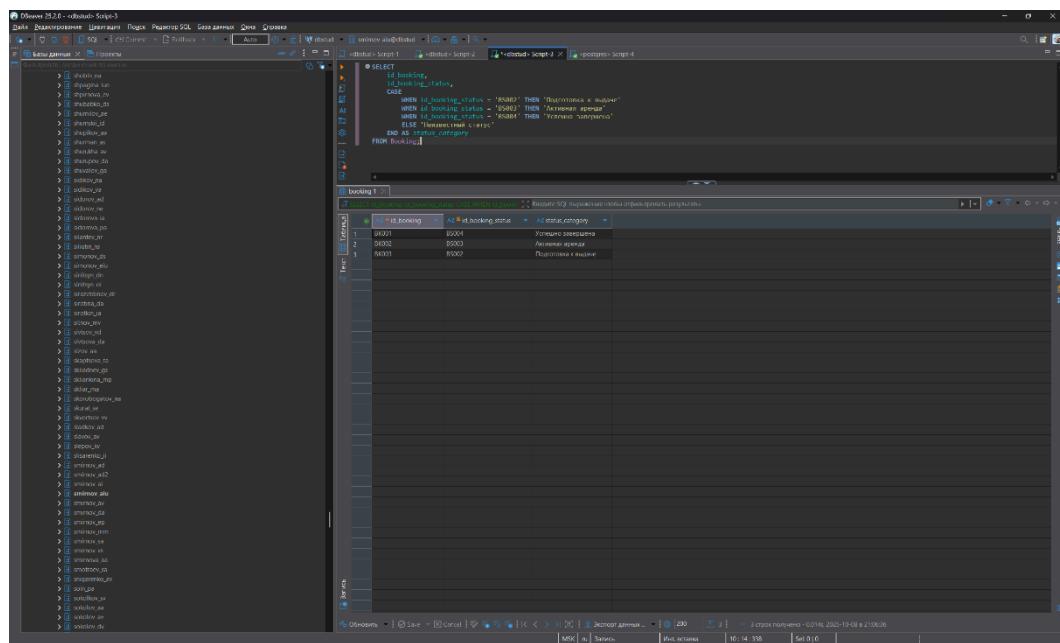
Используя имеющуюся в вашей схеме данных таблицу с иерархической структурой, написать рекурсивный запрос с помощью WITH RECURSIVE для вывода всей иерархии с указанием уровня вложенности.

**ПРИМЕЧАНИЕ:** если в Вашей схеме данных отсутствует таблица с иерархической структурой (т.е. таблица, которая ссылается сама на себя), Вам необходимо создать демонстрационную таблицу для выполнения этого задания. Вы можете выбрать один из двух подходов:

1. Модифицировать существующую таблицу: если у вас есть таблица employees, staff или подобная, Вы можете добавить в неё столбец (например, manager\_id) и внешний ключ, ссылающийся на первичный ключ этой же таблицы.
2. Создать новую таблицу: создайте простую таблицу для демонстрации иерархии, например, для категорий товаров.

## Задание 1: Использование оператора CASE

### 1.1 Поисковое выражение CASE для Анализ статусов бронирований с категоризацией

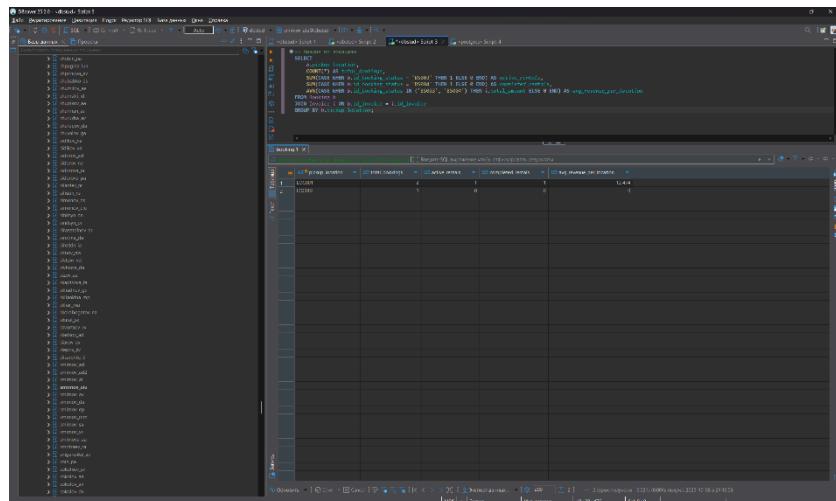


```
SELECT id_booking,
       id_booking_status,
       CASE
         WHEN id_booking_status = 'BOOKED' THEN 'Броньирована и подана'
         WHEN id_booking_status = 'USERS' THEN 'Активна время'
         ELSE 'Несколько забронено'
       END AS status_category
  FROM booking;
```

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Базы данных' (Databases) tree view lists various schemas and tables. In the center, the 'Запросы' (Queries) tab displays the SQL query above. Below it, the 'booking' table is shown with three rows of data. The 'status\_category' column contains the results of the CASE expression: 'Броньирована и подана' for row 1, 'Активна время' for row 2, and 'Несколько забронено' for row 3.

Рисунок 1 - Результат запроса категоризирует результат бронирования

### 1.2 CASE внутри агрегатной функции для анализа подписок



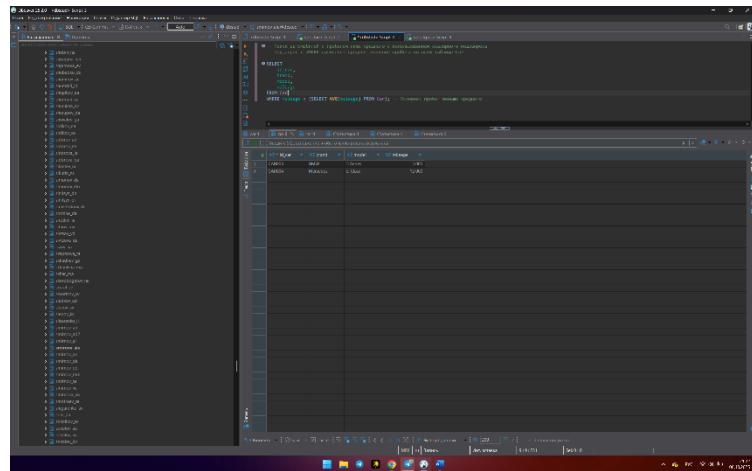
```
SELECT location_id,
       COUNT(CASE WHEN id_booking_status = 'BOOKED' THEN 1 ELSE 0 END) AS total_bookings,
       COUNT(CASE WHEN id_booking_status = 'USERS' THEN 1 ELSE 0 END) AS active_bookings,
       COUNT(CASE WHEN id_booking_status = 'CANCELED' THEN 1 ELSE 0 END) AS canceled_bookings
  FROM booking
 GROUP BY location_id;
```

The screenshot shows the Oracle SQL Developer interface. The 'Базы данных' tree view is visible on the left. The central area shows the SQL query above. The results are displayed in a table titled 'Итоги' (Results) with columns: 'location\_id', 'total\_bookings', 'active\_bookings', and 'canceled\_bookings'. The data shows the count of bookings in each status for each location.

Рисунок 2 - Результат запрос какие локации приносят доход, сколько у них текущих и завершенных заказов.

## Задание 2: Использование подзапросов

### 2.1 Поиск автомобилей с пробегом ниже среднего с использованием скалярного подзапроса



The screenshot shows the MySQL Workbench interface. On the left is a tree view of database schemas and tables. In the center, a query editor window displays a SQL query:

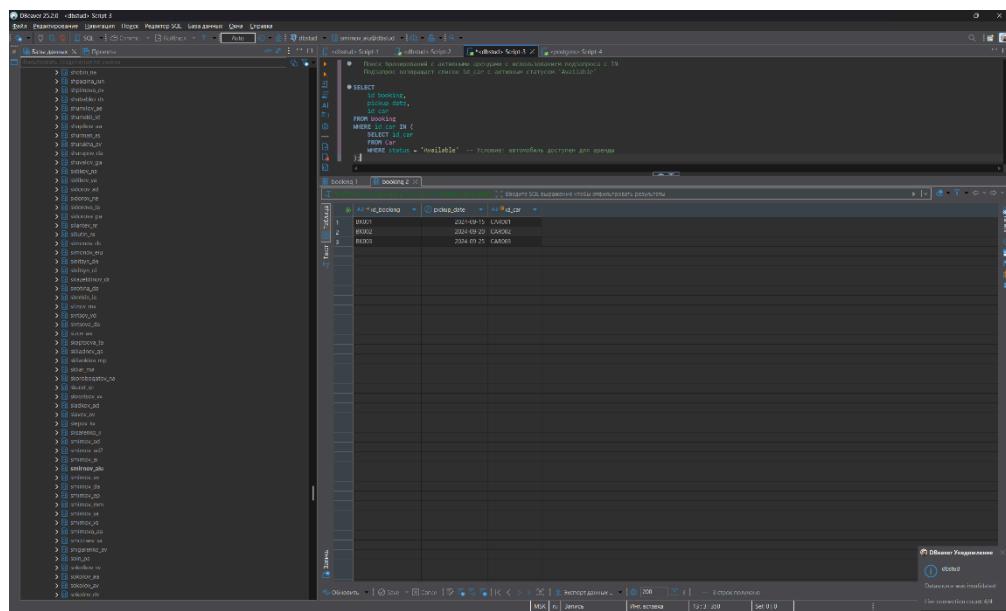
```
SELECT id, name, mileage
FROM cars
WHERE mileage < (SELECT AVG(mileage) FROM cars);
```

Below the query editor is a results grid with three rows of data:

| ID   | NAME | MILEAGE |
|------|------|---------|
| BMW1 | BMW  | 10000   |
| BMW2 | BMW  | 10000   |
| BMW3 | BMW  | 10000   |

Рисунок 3 - Найдены все автомобили, с пробегом ниже среднего с использованием.

### 2.2 Многострочный подзапрос с IN для Поиск бронирований с активными арендами



The screenshot shows the MySQL Workbench interface. On the left is a tree view of database schemas and tables. In the center, a query editor window displays a SQL query:

```
SELECT booking_id, polig_date
FROM bookings
WHERE booking_id IN (
    SELECT id_car
    FROM bookings
    WHERE status = "Available" -- Условие: автомобиль доступен для аренды
```

Below the query editor is a results grid with three rows of data:

| BOOKING_ID | POLIG_DATE |
|------------|------------|
| BMW1       | 2023-09-15 |
| BMW2       | 2023-09-20 |
| BMW3       | 2024-09-25 |

Рисунок 4 - Выведены сессии пользователей, у которых есть активные подписки на текущую дату.

### 2.3 EXISTS проверяет наличие связанных записей в таблице Booking

```

SELECT
    client_id,
    first_name,
    last_name,
    email
FROM Client c
WHERE EXISTS (
    SELECT ...
    FROM Booking b
    WHERE b.id_client = c.id_client -- Коррекция: ссылка на id_client между таблицами
);

```

The screenshot shows a DBeaver interface with three tabs: Script1, Script2, and Script3. The SQL tab contains the provided SQL query. The Results tab displays a table titled 'client' with three rows:

| id_client | first_name | last_name | email            |
|-----------|------------|-----------|------------------|
| C000      | Лариса     | Кутикова  | lars@gmail.com   |
| C001      | Сергей     | Волков    | volkov@yandex.ru |
| C002      | Олег       | Морозов   | oleg@mail.ru     |

Рисунок 5 - Находит клиентов, у которых есть бронирования

## 2.4 Альтернативное решение с использованием JOIN вместо EXISTS

```

SELECT
    client_id,
    first_name,
    last_name,
    email
FROM Client c
JOIN Booking b ON c.id_client = b.id_client;

```

The screenshot shows a DBeaver interface with three tabs: Script1, Script2, and Script3. The SQL tab contains the provided SQL query. The Results tab displays a table with the same data as in Figure 5:

| id_client | first_name | last_name | email            |
|-----------|------------|-----------|------------------|
| C000      | Лариса     | Кутикова  | lars@gmail.com   |
| C001      | Сергей     | Волков    | volkov@yandex.ru |
| C002      | Олег       | Морозов   | oleg@mail.ru     |

Рисунок 6 - JOIN соединяет таблицы Client и Booking по id\_client.

## Задание 3: Использование обобщенных табличных выражений (CTE)

### 3.1 Использование СТЕ для анализа доходности автомобилей

```

CREATE OR REPLACE FUNCTION car_revenue()
RETURNS TABLE(class_id integer, class_name character varying, revenue double precision)
AS $$
    WITH RECURSIVE car_revenue AS (
        SELECT
            c.class_id,
            c.class_name,
            SUM(s.quantity * s.price) AS revenue
        FROM
            car_sales s
            JOIN car_classes c ON s.class_id = c.class_id
        WHERE
            s.quantity > 1000
        GROUP BY
            c.class_id,
            c.class_name
        UNION ALL
        SELECT
            c.class_id,
            c.class_name,
            car_revenue.revenue + SUM(s.quantity * s.price)
        FROM
            car_sales s
            JOIN car_classes c ON s.class_id = c.class_id
            JOIN car_revenue ON c.class_id = car_revenue.class_id
        WHERE
            s.quantity > 1000
        GROUP BY
            c.class_id,
            c.class_name,
            car_revenue.revenue
    )
    SELECT * FROM car_revenue;
$$ LANGUAGE plpgsql;

```

Рисунок 7 - СТЕ создает временный результат car\_revenue

### 3.2 Рекурсивное СТЕ для иерархии жанров фильмов

ШАГ 1: Создание таблицы car\_class\_hierarchy

```

CREATE TABLE car_class_hierarchy (
    parent_id integer,
    child_id integer
);

```

Рисунок 8 - Таблица car\_class\_hierarchy успешно создана в схеме smirnov\_aiu. Таблица содержит необходимые поля для построения иерархии.

## ШАГ 2: Заполнение таблицы данными

Рисунок 9 - Таблица заполнена 9 тестовыми записями, образующими древовидную структуру с 4 уровнями вложенности.

The screenshot shows the pgAdmin 4 interface with a database named 'smirnov\_infobase'. A script editor window displays the following SQL code:

```
INSERT INTO smirnov_infobase.hierarchy (class_id, class_name, parent_class_id) VALUES
-- Основные категории (корневые элементы)
(1, 'Легковые автомобили', NULL),
(2, 'Фургоны', NULL),
(3, 'Минивэны', NULL),
(4, 'Коммерческие', NULL),
-- Подкатегории для легковых автомобилей
(5, 'Эконом класс', 1),
(6, 'Бизнес класс', 1),
(7, 'Премиум класс', 1),
(8, 'Спортивные', 1),
-- Подкатегории для внедорожников
(9, 'Компактные SUV', 2),
(10, 'Средние SUV', 2),
(11, 'Полноразмерные SUV', 2),
(12, 'Кроссоверы', 2),
-- Подкатегории для минивэнов
(13, 'Компактные минивэны', 3),
(14, 'Средние минивэны', 3),
-- Подкатегории для коммерческих
(15, 'Микроавтобусы', 4),
(16, 'Грузопассажирские', 4);

-- Заполнение таблицы class_hierarchy данными
INSERT INTO smirnov_infobase.class_hierarchy (class_id, parent_id, class_id) VALUES
-- Основные категории (корневые элементы)
(1, 'Легковые автомобили', NULL),
(2, 'Фургоны', NULL),
(3, 'Минивэны', NULL),
-- Подкатегории для легковых автомобилей
(5, 'Эконом класс', 1),
(6, 'Бизнес класс', 1),
(7, 'Премиум класс', 1),
(8, 'Спортивные', 1),
-- Подкатегории для внедорожников
```

Below the code, a statistics panel titled 'Статистика 1' provides execution details:

| Name         | Value  |
|--------------|--|
| Updated Rows | 17   |
| Execute time | 0.025s   |
| Start time   | Wed Oct 08 23:48:59 MSK 2015   |
| Finish time  | Wed Oct 08 23:48:59 MSK 2015   |
| Query        | INSERT INTO smirnov_infobase.class_hierarchy (class_id, parent_id, class_id) VALUES<br>-- Основные категории (корневые элементы)<br>(1, 'Легковые автомобили', NULL),<br>(2, 'Фургоны', NULL),<br>(3, 'Минивэны', NULL),<br>-- Подкатегории для легковых автомобилей<br>(5, 'Эконом класс', 1),<br>(6, 'Бизнес класс', 1),<br>(7, 'Премиум класс', 1),<br>(8, 'Спортивные', 1),<br>-- Подкатегории для внедорожников |

## ШАГ 3: Рекурсивный запрос

The screenshot shows the DBeaver interface with several tabs open. The main tab displays a recursive SQL query:

```
WITH RECURSIVE class_tree AS (
    SELECT
        class_id,
        class_name,
        parent_class_id,
        0 as level,
        CAST(class_name AS TEXT) as path
    FROM salinov_auto.car_classes_hierarchy
    WHERE parent_class_id IS NULL
    UNION ALL
    ... Рекурсивный случай: добавляем элементы
    SELECT
        c.class_id,
        c.class_name,
        c.parent_class_id,
        t.level + 1 as level,
        t.path || ' -> ' || c.class_name
    FROM salinov_auto.car_classes_hierarchy c
    JOIN class_tree t ON c.parent_class_id = t.class_id
)
SELECT * FROM class_tree
ORDER BY level, class_name;
```

The results are displayed in a table titled "Результат 1" (Result 1):

| #class_id | class_name          | parent_class_id | level | path                                    |
|-----------|---------------------|-----------------|-------|---|
| 1         | Внедорожники        | [NULL]          | 0     | Внедорожники                            |
| 2         | Коммерческие        | [NULL]          | 0     | Коммерческие                            |
| 3         | Легковые автомобили | [NULL]          | 0     | Легковые автомобили                     |
| 4         | Минивэны            | [NULL]          | 0     | Минивэны                                |
| 5         | Бизнес класс        | 1               | 1     | Легковые автомобили -> Бизнес класс     |
| 6         | Грузопассажирские   | 4               | 1     | Коммерческие -> Грузопассажирские       |
| 7         | Компактные SUV      | 2               | 2     | Внедорожники -> Компактные SUV          |
| 8         | Лифтбэк             | 3               | 2     | Легковые автомобили -> Лифтбэк          |
| 9         | Купе                | 3               | 2     | Легковые автомобили -> Купе             |
| 10        | Кроссоверы          | 2               | 2     | Внедорожники -> Кроссоверы              |
| 11        | Минивэны            | 3               | 2     | Легковые автомобили -> Минивэны         |
| 12        | Микроавтобусы       | 4               | 1     | Коммерческие -> Микроавтобусы           |
| 13        | Полноразмерные SUV  | 2               | 2     | Внедорожники -> Полноразмерные SUV      |
| 14        | Престижный класс    | 1               | 1     | Легковые автомобили -> Престижный класс |

Рисунок 10 - Рекурсивный запрос

## АНКЕРНАЯ

## ЧАСТЬ

Находит: Все классы автомобилей без родителя (корневые элементы)

Пример: ('Легковые автомобили', NULL) → Уровень 0, Путь: "Легковые автомобили"

## РЕКУРСИВНАЯ

## ЧАСТЬ

Находит: Все дочерние классы для текущего уровня

Пример: Для "Легковые автомобили" находит → "Эконом класс", "Бизнес класс", "Премиум класс"

### ПРОЦЕСС РЕКУРСИИ:

- **Итерация 1 (Уровень 0):** Находит: Легковые автомобили, Внедорожники, Минивэны, Коммерческие (Level=0)
- **Итерация 2 (Уровень 1):**
  - Для "Легковые автомобили" находит: Эконом класс, Комфорт класс, Бизнес класс, Премиум класс, Спортивные
  - Для "Внедорожники" находит: Компактные SUV, Среднеразмерные SUV, Полноразмерные SUV, Кроссоверы
  - Для "Минивэны" находит: Компактные минивэны, Семейные минивэны
  - Для "Коммерческие" находит: Микроавтобусы, Грузопассажирские
- **Итерация 3 (Уровень 2):**
  - Для "Эконом класс" находит подкатегории (если бы они были)
  - Для "Бизнес класс" находит подкатегории (если бы они были)
  - И так далее по всем классам 1-го уровня

### Пример путей:

- Легковые автомобили → Эконом класс (Level=1)
- Внедорожники → Компактные SUV (Level=1)
- Легковые автомобили → Бизнес класс → Представительские (Level=2, если бы были такие подкатегории)

## ВЫВОД

Все задания успешно выполнены. В ходе работы продемонстрированы практические навыки работы с различными конструкциями SQL:

**По оператору CASE:** Освоено использование поискового выражения CASE для категоризации данных и применение CASE внутри агрегатных функций для условной агрегации.

**По подзапросам:** Успешно применены различные типы подзапросов - скалярные, многострочные с IN, коррелированные с EXISTS, а также показана возможность альтернативной реализации с использованием JOIN.

**По СТЕ:** Освоена работа с обобщенными табличными выражениями, включая сложные рекурсивные запросы для обработки иерархических данных. Создана демонстрационная таблица и выполнен рекурсивный обход дерева категорий.

Все запросы адаптированы под конкретную схему базы данных и выполняются корректно. Задания выполнены в полном объеме согласно требованиям технического задания.

