



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Дисциплина «Разработка баз данных»

Практическая работа №3.

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



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание 1: использование оператора CASE

1. 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**.
2. Составить запрос, в котором **оператор CASE** используется **внутри агрегатной функции** (например, **SUM** или **COUNT**) для выполнения **условной агрегации**.

Задание 2: использование подзапросов (часть 1)

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. **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**).

(продолжение на следующем слайде)

Практическая работа №3.

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



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание 2: использование подзапросов (часть 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**.
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).

1. **Стандартное CTE:** переписать запрос из Задания 2.3 («Коррелированный подзапрос с EXISTS») с использованием **обобщенного табличного выражения (CTE)**.
2. **Рекурсивное CTE:** 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.

(начало на предыдущем слайде)

Практическая работа №3.

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



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.

* **Пример кода** для создания подобной таблицы есть в **файле задания**

1. Реализация условной логики: оператор CASE



Оператор **CASE** позволяет реализовать логику «если-то-иначе» прямо в **SQL-запросе**.

Простой CASE:

```
CASE
    WHEN 'строка_1' THEN результат1
    WHEN 42 THEN результат2
    ELSE результат_по_умолчанию
END
```

«Поисковый» CASE:

```
CASE
    WHEN условие1 THEN результат1
    WHEN условие2 THEN результат2
    ELSE результат_по_умолчанию
END
```

Краткое описание:

CASE ... END – начало и конец блока условий.
WHEN ... THEN ... – условие и результат (*кол-во не ограничено*).
ELSE – результат, если не сработало ни одно другое условие.

Основные применения:

- **Категоризация данных** – присвоение меток строкам («дорогой»/«дешевый»).
- **Условная агрегация** – подсчет итогов (**SUM**, **COUNT**) только для некоторой части данных.

1. Реализация условной логики: оператор CASE – применение



CASE – это универсальная конструкция, позволяющая реализовать логику «если-то-иначе» внутри SQL-запроса.

Его можно использовать **почти везде**, где **ожидается значение**:

- ✓ **SELECT** – для вывода нового столбца, где для каждой строки **вычисляется значение** (самый частый сценарий).

Пример: категоризация товаров **по цене** («дешёвый», «средний», «дорогой»).

- ✓ **WHERE** – для построения **динамической логики фильтрации**.

Пример: искать **активных** пользователей среди **VIP-клиентов**, но всех – среди **обычных**.

- ✓ **ORDER BY** – для **нестандартной сортировки**.

Пример: всегда **показывать** определённый статус (*например, «в работе»*) **первым в списке**.

- ✓ **GROUP BY** – для группировки **по условному признаку**.

Пример: сгруппировать продажи не по городам, а **по регионам** («центр», «юг»).

- ✓ **UPDATE** – для **условного обновления данных**.

Пример: обновить скидку **по-разному** для **разных категорий** клиентов.

1. Реализация условной логики: оператор CASE – примеры запросов



Далее представлены **примеры запросов с использованием CASE**.

Исходные данные для запросов:

id	fio	department	position	salary
1	Иванов Б.Д.	IT	Разработчик	160000
2	Петров С.А.	Sales	Менеджер	110000
3	Сидоров Д.Т.	IT	Директор	250000
4	Смирнова А.Е.	Support	Специалист	65000
5	Кузнецов К.В.	Sales	Специалист	80000

1. Реализация условной логики: оператор CASE – примеры запросов



SELECT – для вывода нового столбца, где для каждой строки **вычисляется значение** (самый частый сценарий).

Задача: разделить всех сотрудников на категории по уровню зарплаты.

```
SELECT  
    fio,  
    salary,  
CASE  
    WHEN salary > 200000 THEN 'БОСС'  
    WHEN salary >= 100000 THEN 'Высокая'  
    ELSE 'Стандартная'  
END AS salary_grade  
FROM  
employees;
```

fio	salary	salary_grade
Иванов Б.Д.	160000	Высокая
Петров С.А.	110000	Высокая
Сидоров Д.Т.	250000	БОСС
Смирнова А.Е.	65000	Стандартная
Кузнецов К.В.	80000	Стандартная

1. Реализация условной логики: оператор CASE – примеры запросов



WHERE – для построения **динамической логики фильтрации**.

Задача: выбрать директоров из IT, а из отдела продаж (Sales) – только специалистов.

```
SELECT
    fio, department, position
FROM
    employees
WHERE
CASE
    WHEN department = 'IT' THEN position = 'Директор'
    WHEN department = 'Sales' THEN position = 'Специалист'
    -- Остальные отделы и должности нас не интересуют
    ELSE FALSE
END;
```

fio	department	position
Сидоров Д.Т.	IT	Директор
Кузнецов К.В.	Sales	Специалист

1. Реализация условной логики: оператор CASE – примеры запросов



ORDER BY – для нестандартной сортировки.

Задача: отсортировать сотрудников так, чтобы сначала шли директора, потом менеджеры, а потом все остальные.

```
SELECT
    fio, position
FROM
    employees
ORDER BY
    CASE
        WHEN position = 'Директор' THEN 1 -- Высший приоритет
        WHEN position = 'Менеджер' THEN 2 -- Средний приоритет
        ELSE 3 -- Низший приоритет
    END;
```

fio	position
Сидоров Д.Т.	Директор
Петров С.А.	Менеджер
Иванов Б.Д.	Разработчик
Смирнова А.Е.	Специалист
Кузнецов К.В.	Специалист

1. Реализация условной логики: оператор CASE – примеры запросов



GROUP BY – для группировки по условному признаку.

Задача: посчитать, сколько у нас сотрудников с высокой зарплатой ($\geq 100\ 000$) и со стандартной ($< 100\ 000$).

```
SELECT
CASE
    WHEN salary >= 100000 THEN 'Высокая ЗП'
    ELSE 'Стандартная ЗП'
END AS salary_group,
COUNT(id) as employee_count
FROM
employees
GROUP BY
salary_group;
```

salary_group	employee_count
Высокая ЗП	3
Стандартная ЗП	2

1. Реализация условной логики: оператор CASE – примеры запросов



UPDATE – для **условного обновления данных**.

Задача: повысить зарплату на 10% специалистам и на 5% – всем остальным сотрудникам.

```
UPDATE  
    employees  
SET  
    salary = salary * CASE  
        -- +10%  
        WHEN position = 'Специалист' THEN 1.10  
        -- +5%  
        ELSE 1.05  
END;
```

fio	salary	salary_new	%
Иванов Б.Д.	160000	168000	+5%
Петров С.А.	110000	115500	+5%
Сидоров Д.Т.	250000	262500	+5%
Смирнова А.Е.	65000	71500	+5%
Кузнецов К.В.	80000	88000	+10%

2. Использование подзапросов: отдельные подзапросы



Подзапрос – это **SELECT**-запрос, вложенный внутрь другого запроса.

Позволяет использовать результат одной выборки для фильтрации или вычисления данных в другой.

Отдельные* запросы, возвращающие значения:

1. Скалярный подзапрос – возвращает ячейку (одно значение)

... `WHERE column > (SELECT AVG(column) FROM table)`

2. Многострочный подзапрос – возвращает список (много строк, 1 столбец)

... `WHERE id IN (SELECT id FROM other table)`

3. Табличный подзапрос – возвращает таблицу (много строк, много столбцов)

... `FROM (SELECT column1, column2 FROM table) AS temp_table`

* Под «отдельными» понимается, что эти запросы могут быть использованы сами по себе.

2. Использование подзапросов: коррелированный подзапрос



Помимо **отдельных** подзапросов, существует ещё один тип.

```
SELECT  
    table1.name  
FROM  
    table1  
WHERE EXISTS (  
    SELECT 1  
    FROM  
        table2  
    WHERE  
        -- Корреляция (связь)  
        table2.id = table1.id  
);
```

Зависимый (коррелированный) подзапрос:

- Не может быть выполнен отдельно, так как ссылается на столбцы внешнего запроса.
- Выполняется многократно, для каждой строки внешнего запроса, используя значения из этой строки.
- Чаще всего применяется в **WHERE** с оператором **EXISTS** для проверки **наличия связанных записей**.

Обычно используется для проверки **условия**, специфичного для **каждой строки** (например, "найти сотрудников, чья зарплата выше средней по их отделу").

3. Использование обобщенных табличных выражений (СТЕ)



СТЕ (Common Table Expression) – это, фактически, именованный подзапрос.

Он позволяет определить временный набор данных с помощью **WITH** и затем ссылаться на него по имени в основном запросе.

```
WITH aspirin AS (
    SELECT DISTINCT manufacturer_id
    FROM medicines
    WHERE LOWER(name) LIKE '%аспирин%'
)
SELECT
    m.manufacturer_name
FROM
    manufacturers AS m
JOIN
    aspirin AS a
ON
    m.manufacturer_id = a.manufacturer_id;
```

Правила и особенности:

- СТЕ «живёт» только **в рамках одного запроса**, который следует **сразу за ним**.
`SELECT * FROM aspirin;` -- первый запрос будет работать
`SELECT * FROM aspirin;` -- второй запрос выдаст ошибку
- На **один и тот же СТЕ** можно **ссылаться** несколько раз в последующих частях **одного и того же запроса**.
- СТЕ **можно использовать** не только **в SELECT**, но и **в INSERT, UPDATE, DELETE**.

3. Использование обобщенных табличных выражений (СТЕ) - Рекурсия



Рекурсивный СТЕ – особый вид СТЕ, который может **ссылаться сам на себя**.

Обязательная структура из двух частей:

- **Якорный запрос** – базовый **SELECT**, который **выполняется один раз**.
Формирует **стартовый набор строк**.
Его задача - найти «**корневые**» элементы иерархии
- **Рекурсивный запрос** – **SELECT**, который **ссылается** на имя **самого СТЕ**.
Выполняется **многократно**.
На каждой итерации он **присоединяет** к результатам **предыдущего шага** следующий уровень.

После выполнения, две эти части обязательно **соединяются** через **UNION ALL**.

Ключевой момент: у рекурсивной части должно быть **условие завершения**.

Рекурсия **останавливается**, когда рекурсивный запрос **перестает возвращать новые строки**.

```
WITH RECURSIVE Hierarchy AS (
    -- 1. Якорь: Находим «корень» иерархии
    SELECT
        id, name, manager_id, 0 AS lvl
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    -- 2. Рекурсия: Находим подчиненных
    SELECT
        e.id, e.name, e.manager_id, h.lvl+1
    FROM employees AS e
    JOIN Hierarchy AS h
    ON e.manager_id = h.id
)
```

SELECT * FROM Hierarchy;

3. Использование обобщенных табличных выражений (СТЕ) - Рекурсия



Таблица "Сотрудники"

ID	Имя	Фамилия	ID руководителя
1	Иван	Сорокин	NULL
2	Мария	Петровна	1
3	Алексей	Орлов	2
4	Евгений	Смирнов	1
5	Евгений	Смирнов	4



Якорный запрос

ID	Имя	Фамилия	ID руководителя	level
1	Иван	Сорокин	NULL	0

Рекурсия (итерация 1)

ID	Имя	Фамилия	ID руководителя	level
2	Мария	Петровна	1	1
4	Евгений	Смирнов	1	1

Рекурсия (итерация 2)

ID	Имя	Фамилия	ID руководителя	level
3	Алексей	Орлов	2	2
5	Наталья	Голубкина	4	2



Итоговая таблица иерархии сотрудников (рекурсивный СТЕ)

ID	Имя	Фамилия	ID руководителя	level
1	Иван	Сорокин	NULL	0
2	Мария	Петровна	1	1
4	Евгений	Смирнов	1	1
3	Алексей	Орлов	2	2
5	Наталья	Голубкина	4	2

Рекурсия (итерация 3)

ID	Имя	Фамилия	ID руководителя	level

3. Использование обобщенных табличных выражений (СТЕ)



Рекурсивный СТЕ или SELF JOIN?

SELF JOIN (*простое соединение таблицы с собой*):

- Находит только прямую связь на **один уровень вглубь**.
- Отвечает на вопрос: «Кто мой **непосредственный** руководитель?».
- **Не может** построить всю цепочку подчиненности.

Рекурсивный СТЕ:

- Нужен именно для построения полной, **многоуровневой иерархии**.
- Начинает с **«вершины»** и итеративно спускается по всем **«ветвям» до самого конца**.
- Отвечает на вопрос: «Покажи **всю структуру подчиненности**, начиная с генерального директора».



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Спасибо за внимание