ЛАБОРАТОРНАЯ РАБОТА №6

**ИНДЕКСЫ. ПРЕДСТАВЛЕНИЯ.  
ОБОБЩЁННЫЕ ТАБЛИЧНЫЕ ВЫРАЖЕНИЯ**

**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

# **ИНДЕКСЫ**

Индекс — структура данных для ускорения поиска и сортировки в таблицах. Как оглавление в книге. Роль индексов состоит в том, чтобы облегчить поиск подмножества строк и столбцов таблицы без необходимости сканировать каждую строку в таблице.

*CREATE INDEX* *idx\_ord\_amt ON ord(amt);*

# **Типы индексов в PostgreSQL**

1. ***B-Tree*** (по умолчанию) - для диапазонных запросов ( =, <, >, BETWEEN) и сортировки.

*CREATE INDEX idx\_ord\_amt ON ord USING* ***BTREE*** *(amt);*

2. ***Hash индекс*** – только для точных совпадений (=). Быстрее, чем B-Tree, но не поддерживает диапазоны.

*CREATE INDEX idx\_ord\_amt ON ord USING* ***HASH*** *(amt);*

3. ***GiST*** (Generalized Search Tree) – для геоданных, полнотекстового поиска.

*CREATE INDEX idx\_client\_geo ON client USING* ***GIST*** *(location);*

4. ***GIN*** (Generalized Inverted Index) – для составных данных (массивы, JSON).

*CREATE INDEX idx\_client\_tags ON client USING GIN (tags);*

# **Специальные возможности индексов**

Уникальный индекс

*CREATE UNIQUE INDEX idx\_ord\_amt\_un ON ord(amt);*

Индекс без пустых полей

*CREATE INDEX idx\_ord\_date\_nn ON ord(ord\_date)   
WHERE ord\_date IS NOT NULL;*

Частичный индекс по условию

*CREATE UNIQUE INDEX idx\_ord\_date\_0109 ON ord(ord\_date)*

*WHERE ord\_date = ‘2025-09-01’ ;*

Индекс по выражениям

*CREATE INDEX idx\_cust\_name ON cust(LOWER(name));*

*CREATE INDEX idx\_name\_city ON cust((name || ' ' || city));*

# **Управление индексами**

Все индексы схемы

*SELECT \* FROM pg\_indexes WHERE schemaname = 'my\_schema';*

Перестроение индекса

*REINDEX INDEX idx\_ord\_amt;*

Перестроение всех индексов таблицы

*REINDEX TABLE ord;*

Удаление индекса

*DROP INDEX IF EXISTS idx\_ord\_amt;*

# **Анализ использования индексов**

План запроса и проверка использования индекса

*EXPLAIN ANALYZE  
SELECT \* FROM ord ORDER BY amt;*

Статистика использования индексов

*SELECT schemaname, relname, indexrelname, idx\_scan   
FROM pg\_stat\_all\_indexes  
WHERE relname = 'ord';*

***Не рекомендуется создавать индексы:***

1. Часто обновляемые таблицы

2. Маленькие таблицы (< 1000 строк)

3. Редко запрашиваемые столбцы

# **ПРЕДСТАВЛЕНИЯ**

**Представление (VIEW)** — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.

Представления называют «виртуальными таблицами», поскольку представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. В памяти хранится **запрос**, а не строки выборки.

Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится.

Используется для инкапсуляции сложных запросов, повышения безопасности.

**Создание представления**

*CREATE [OR REPLACE] VIEW big\_ords AS  
SELECT onum, pnum, cnum, snum, amt  
FROM ord  
WHERE amt > 30;*

**Просмотр представления** и его **удаление** производится так же, как и для таблицы:

*SELECT \* FROM big\_ords;*

*DROP VIEW IF EXISTS big\_ords;*

# **Материализованные представления**

В отличии от обычного представления, материализованное представление хранит результат выполнения запроса на диске, сохраняя его в виде таблицы. Однако запрос, с которым создавалось представление, сохраняется так же, как и в обычном. Используется для улучшения производительности для не часто изменяющихся данных.

**Создание представления**

*CREATE* ***MATERIALIZED*** *VIEW big\_ords\_mat AS  
SELECT onum, pnum, cnum, snum, amt  
FROM ord  
WHERE amt > 30;*

**Обновление материализованного представления**

*REFRESH MATERIALIZED VIEW big\_ords\_mat;*

# **ОБОБЩЕННЫЕ ТАБЛИЧНЫЕ ВЫРАЖЕНИЯ**

CTE (Common Table Expressions) – это временный набор результатов, который определяется и используется в рамках выполнения одного оператора (SELECT, INSERT, UPDATE или DELETE).

CTE позволяют упрощать выполнение сложных запросов, делая их более читаемыми и поддерживаемыми. Используется также для рекурсивных запросов, для улучшения читаемости, когда нужно повторно использовать результат в одном основном запросе.

**Создание CTE**

Для создания используется оператор WITH.

***WITH*** *big\_ords AS (*

*SELECT onum, pnum, cnum, snum, amt  
FROM ord  
WHERE amt > 30)*

*SELECT \* FROM big\_ords;*

Использование нескольких CTE

*WITH* ***ords\_2024*** *AS (*

*SELECT \*   
FROM ord   
WHERE ord\_date >= '2024-01-01' ),*

***big\_ords*** *AS (*

*SELECT onum, pnum, cnum, snum, amt  
FROM ord  
WHERE amt > 30)*

*SELECT \* FROM big\_ords;*

# **Рекурсивные CTE**

Позволяют выполнять рекурсивные запросы — запросы, где результат одного шага зависит от результата предыдущего.

 Тело CTE представляет собой запрос с UNION ALL, который объединяет один или несколько подзапросов, называемых закреплёнными элементами, которые заполняют первичный набор результатов. Кроме закреплённых элементов есть один или несколько рекурсивных подзапросов, которые возвращают оставшуюся часть результирующего набора. Эти рекурсивные подзапросы ссылаются на сам рекурсивный CTE. То есть есть один или несколько закреплённых подзапросов и один или несколько рекурсивных подзапросов, объединенных UNION ALL.

**Пример для работы с иерархическими данными**

Выводится уровень менеджера в зависимости от имеющихся у него подчиненных. Если подчиненных нет, то уровень = 1, что указано с закрепленном подзапросе. Если подчиненные, то уровень увеличивается на 1, что указано в рекурсивном подзапросе. Если у этих подчиненных есть еще подчиненные, уровень еще увеличивается и так далее (аналог рекурсии в SQL).

***WITH RECURSIVE*** *employee\_hierarchy AS (*

*-- закрепленный подзапрос, начальные данные*

*SELECT id, name, manager\_id, 1 as level*

*FROM employees*

*WHERE manager\_id IS NULL*

*UNION [ ALL ]*

*-- рекурсивный подзапрос*

*SELECT e.id, e.name, e.manager\_id, eh.level + 1*

*FROM employees e*

*INNER JOIN employee\_hierarchy eh ON e.manager\_id = eh.id*

*)*

*SELECT \* FROM employee\_hierarchy;*

**ПРАКТИЧЕСКОЕ ЗАДАНИЕ**

*Вариант 1-6*

1. Вывести список существующих индексов в БД. Являются ли индексы кластеризованными? Какую структуру в памяти они имеют?
2. Создать индекс для ускорения сортировки таблицы покупателей по городу. Проверьте, используется ли созданный индекс при сортировке.
3. Добавьте в таблицу покупателей 1000 новых записей, например, используя команду, формирующую последовательности:

*insert into cust   
values ( generate\_series(1, 1000), 'test\_name', 100, 'test\_city' );*

Проверьте, используется ли теперь созданный индекс при сортировке?

В заданиях 4-7 необходимо выполнить одинаковый запрос, сохраняя его содержимое в различных структурах. Полученную (сохраненную) выборку вывести на экран. В чем отличие этих структур?

*Вывести полную информацию о заказе, его продавце, его продукте и его покупателе, только если:*

*количество товара (amt) в этом заказе больше, чем среднее по таблице,  
 товар не из Санкт-Петербурга,   
 продавец совершил не более 10 заказов за все время,*

*рейтинг покупателя не ниже, чем хотя бы у одного покупателя из Москвы.*

1. Сохранить результат как новую таблицу и вывести ее содержимое.
2. Сохранить результат как представление и вывести его содержимое.
3. Сохранить результат как материализованное представление и вывести его содержимое.
4. Использовать для запроса блок оператора WITH и вывести результат.