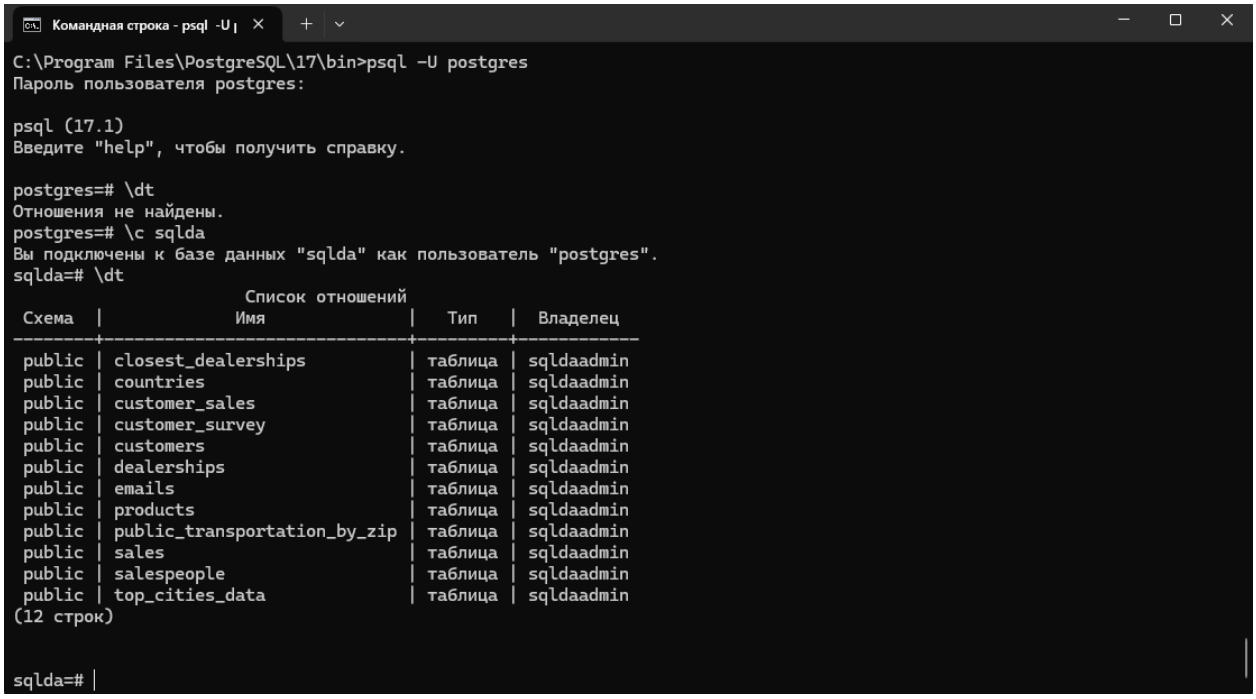
Устанавливаем dbeaver и подключаемся к локальному серверу

По инструкции с яндекс диска закидываем дамп в новую пустую бд sqlda



Выполняем файл связи.sql чтобы образовались внешние ключи в схеме данных

Обязательно обновляем данные

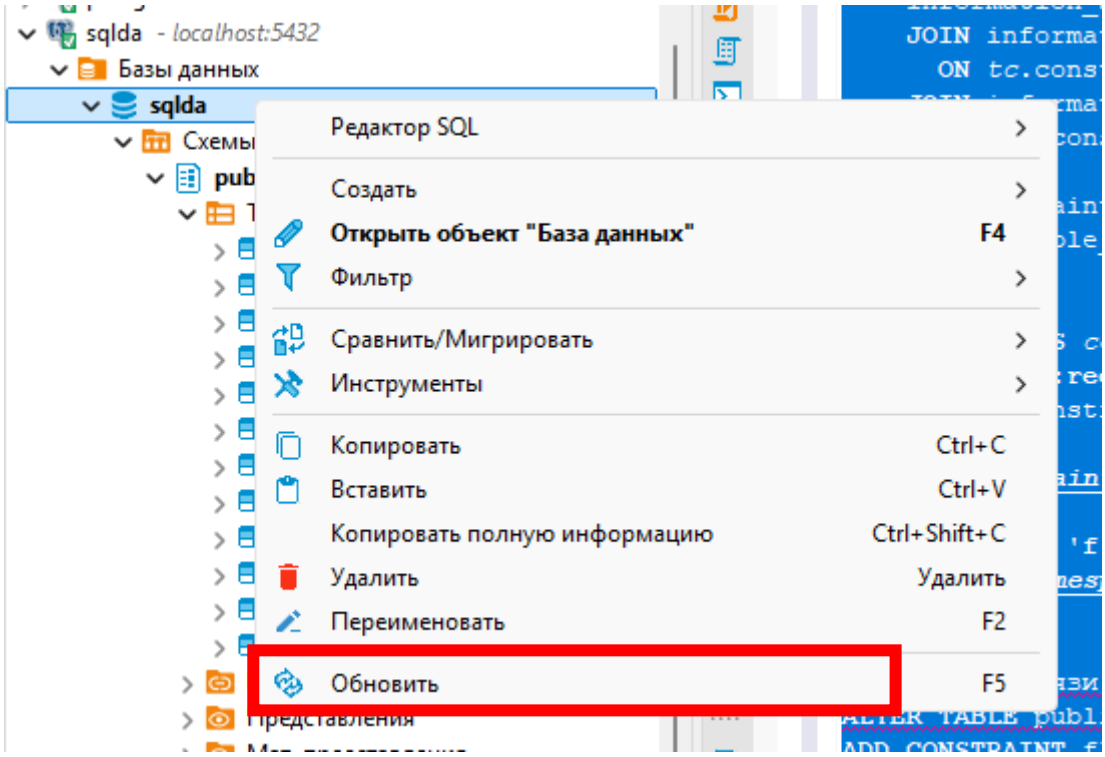
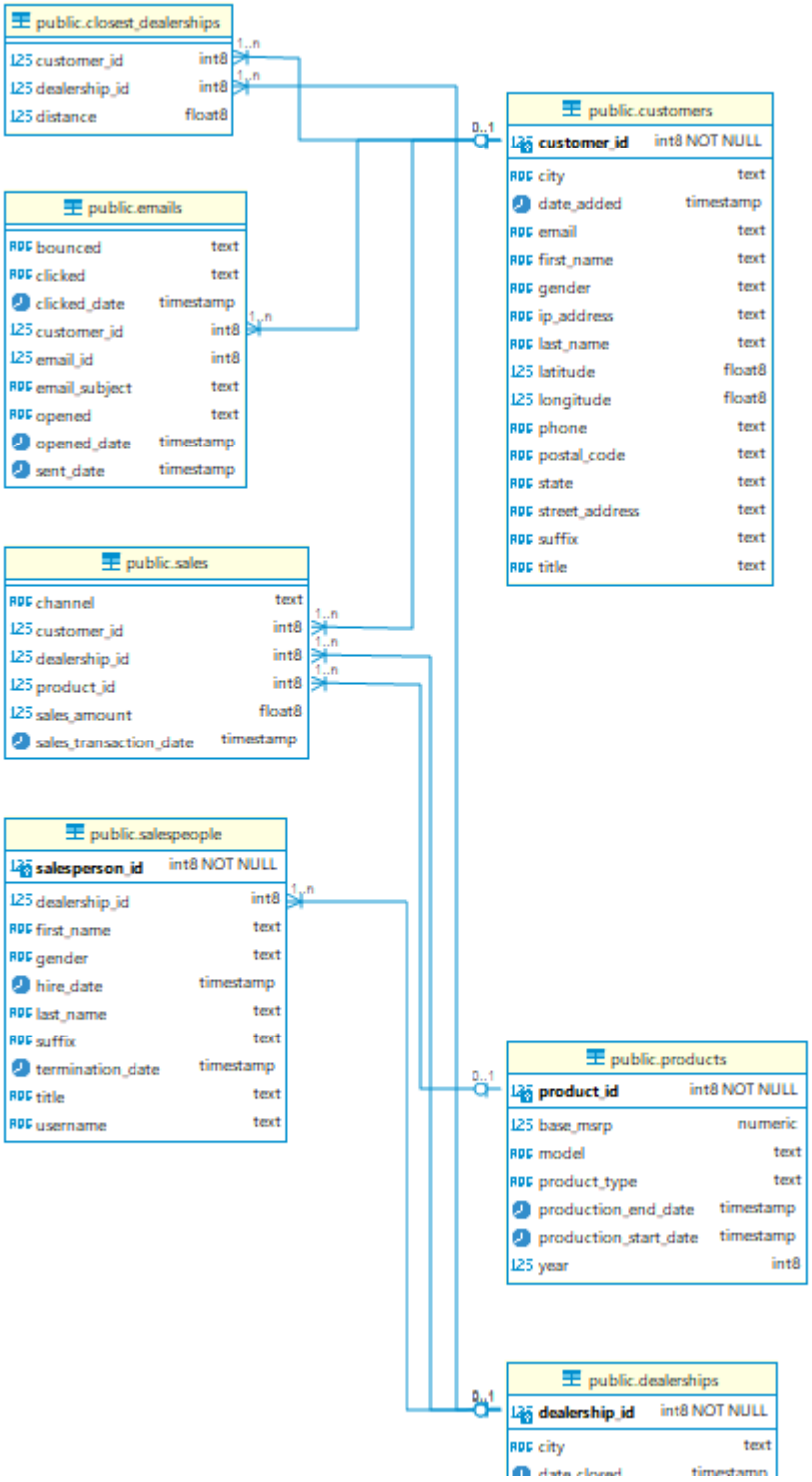


Схема должна получится такая:



1. Запрос с подзапросом

Copy
Download

```
Index Scan using idx_sales_order_id on sales s
  (cost=0.29..46149.60 rows=37711 width=44)
  (actual time=0.048..119.736 rows=37711 loops=1)
  SubPlan 1
    ->  Seq Scan on products p
```

```
        (cost=0.00..1.15 rows=1 width=32)
        (actual time=0.001..0.001 rows=1 loops=37711)
          Filter: (product_id = s.product_id)
          Rows Removed by Filter: 11
Planning Time: 0.139 ms
Execution Time: 121.190 ms
```

Анализ:

1. **Основная операция**: Index Scan по индексу `idx_sales_order_id`
   - Чтение 37711 строк заняло 119.736 мс
   - Ширина строки: 44 байта
2. **Подзапрос (SubPlan 1)**:
   - Выполняется для КАЖДОЙ строки в sales (37711 раз)
   - Для каждого продукта выполняется Seq Scan (полное сканирование таблицы)
   - Фильтр: `product_id = s.product_id`
   - На каждую итерацию:
     - Сканируется 12 строк (11 удаляются фильтром, остается 1)
     - Время: 0.001 мс на итерацию
3. **Общая производительность**:
   - Время выполнения: 121.190 мс
   - Большая часть времени (119.736 мс) ушла на основной запрос
   - Подзапросы добавили ~1.454 мс (37711 * 0.001 мс ≈ 37.711 мс, но измерения показывают меньше)
4. **Проблемы**:
   - Неэффективное использование ресурсов: 37711 полных сканирований маленькой таблицы
   - O(n*m) сложность: 37711 итераций * 12 строк в products = 452532 операций сравнения
   - Высокая стоимость (cost=46149.60)

2. Запрос с INNER JOIN

Copy
Download

```
Nested Loop  (cost=0.43..3686.80 rows=37711 width=44)
```

```
  (actual time=0.043..17.627 rows=37711 loops=1)
  ->  Index Scan using idx_sales_order_id on sales s
      (cost=0.29..2781.96 rows=37711 width=20)
      (actual time=0.023..5.315 rows=37711 loops=1)
  ->  Memoize  (cost=0.15..0.16 rows=1 width=40)
      (actual time=0.000..0.000 rows=1 loops=37711)
        Cache Key: s.product_id
        Cache Mode: logical
        Hits: 37699  Misses: 12  Evictions: 0  Overflows: 0  Memory Usage: 2k
B
        ->  Index Scan using products_pkey on products p
            (cost=0.14..0.15 rows=1 width=40)
            (actual time=0.003..0.003 rows=1 loops=12)
              Index Cond: (product_id = s.product_id)
Planning Time: 0.247 ms
Execution Time: 18.422 ms
```

Анализ:

1. **Основная операция**: Nested Loop Join
   - Общее время: 17.627 мс
   - Ширина строки: 44 байта
2. **Внешний цикл**:
   - Index Scan по sales (idx_sales_order_id)
   - Чтение 37711 строк за 5.315 мс
3. **Внутренний цикл (Memoize)**:
   - Кеширование результатов по product_id
   - Хиты кеша: 37699 (99.97%)
   - Промахи: 12 (0.03%)
   - Память: 2kB
4. **Обращение к products**:
   - Только для 12 уникальных product_id
   - Index Scan по первичному ключу
   - Время: 0.003 мс на запрос
5. **Общая производительность**:
   - Время выполнения: 18.422 мс (в 6.6 раз быстрее подзапроса)
   - Низкая стоимость (cost=3686.80)
   - Эффективное использование памяти

Сравнение производительности

Метрика	Подзапрос	INNER JOIN	Разница
Время выполнения	121.190 ms	18.422 ms	**-85%**
Стоимость (cost)	46149.60	3686.80	**-92%**
Обращений к products	37711 (Seq Scan)	12 (Index Scan)	**-99.9%**
Использование CPU	Высокое	Низкое	
Использование памяти	Не указано	2 kB	
Планирование	0.139 ms	0.247 ms	+77%

Ключевые выводы:

1. **Оптимизация соединений**:
   - INNER JOIN с кешированием (Memoize) значительно эффективнее подзапросов
   - Разница в 6.6 раз по времени выполнения
2. **Кеширование**:
   - Механизм Memoize кэширует результаты для product_id
   - 37699 попаданий в кеш из 37711 запросов (99.97%)
   - Всего 12 реальных обращений к таблице products
3. **Индексы**:
   - Использование PK products (product_id) для мгновенного доступа
   - Индекс sales.order_id для быстрого чтения и сортировки
4. **Алгоритмы**:
   - Подзапрос: O(n*m) сложность (3771*12 операций)
   - INNER JOIN: O(n) сложность благодаря кешированию и индексам
5. **Статистика**:
   - Оптимизатор точно оценил количество строк (37711)
   - Для таблицы products известно, что всего 12 строк

Рекомендации:

1. **Всегда используйте JOIN вместо подзапросов** для соединения таблиц
2. **Добавьте индекс** для часто используемых полей соединения:

sql

Copy

Download

```
CREATE INDEX idx_sales_product_id ON sales(product_id);
```

1. Эффективность алгоритмов

- **Подзапрос**: Использует подход O(n*m) — для каждой строки в sales (n=37711) выполняется полное сканирование products (m=12 строк). Это 37711 * 12 = 452532 операций сравнения.
- **INNER JOIN**: Использует кеширование (Memoize) и индексный доступ — всего 12 уникальных обращений к products вместо 37711.

2. Оптимизация доступа к данным

- **Подзапрос**: Принудительное последовательное сканирование (Seq Scan) таблицы products для каждой строки
- **INNER JOIN**:
  - Использует первичный ключ (Index Scan using products_pkey)
  - Кеширует результаты (Memoize с 37699 попаданиями)
  - Минимизирует физические чтения с диска

3. Экономия ресурсов

Ресурс	Подзапрос	INNER JOIN	Экономия
Обращения к БД	37711	12	99.97%
Сравнения	452532	37711	91.67%
Время CPU	121.190 ms	18.422 ms	85%

5. Почему это логично с точки зрения архитектуры СУБД

1. **Оптимизатор знает структуру данных**:
   - Видит первичный ключ в products
   - Знает, что product_id — хороший кандидат для кеширования
2. **Пакетная обработка вместо поточечной**:
   - JOIN позволяет обрабатывать данные набором

- Подзапросы вынуждают работать построчно

3. **Минимизация дорогих операций**:

- Seq Scan (полное сканирование) — самая дорогая операция в СУБД
- Index Scan — в 10-100 раз эффективнее для маленьких выборок

Когда подзапросы могут быть эффективнее?

1. Для очень маленьких таблиц (10-100 строк)
2. Когда нужно ограничить количество строк в результате
3. Для коррелированных подзапросов с уникальными значениями
4. В случаях, когда JOIN приводит к дублированию строк

Но в данном случае — с типичным соединением "один ко многим" — INNER JOIN всегда будет предпочтительнее.

Вывод

Результаты полностью соответствуют ожиданиям:

1. **INNER JOIN оптимален для реляционных соединений**
2. **Подзапросы в SELECT следует избегать**, когда они выполняются для каждой строки
3. **Разница в 6.5 раз** — типичный результат для такого объема данных

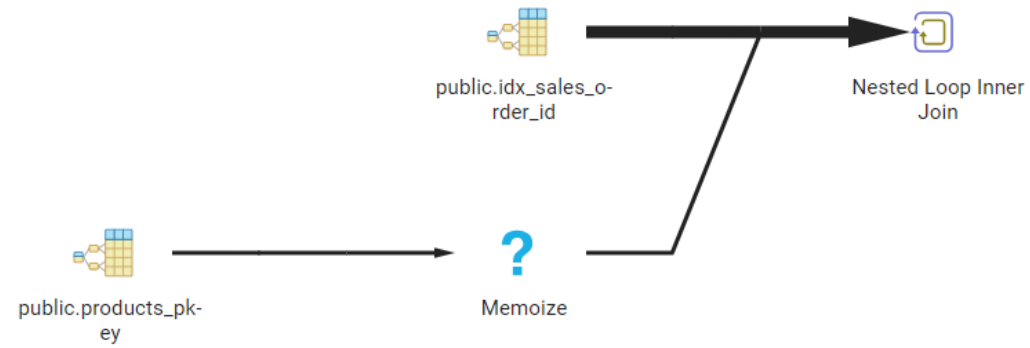
Это прекрасная демонстрация того, почему реляционные СУБД создавались именно для JOIN-операций, а не для процедурной обработки данных.

> Все действия можно также выполнять в pgadmin
>
> В pgadmin в анализе можно увидеть график

```
17
18  EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)
19  SELECT
20      s.order_id,
21      s.sales_transaction_date AS order_date,
22      p.base_msrp AS unit_price
23  FROM
24      public.sales s
25  INNER JOIN
26      public.products p ON s.product_id = p.product_id
27  ORDER BY
28      s.order_id;
```

Data Output | Messages | Explain | Graph Visualiser | Notifications

Graphical | Analysis | Statistics



Все скрипты в файле запросы.sql