НИЯУ МИФИ. Лабораторная работа №2-4

Индексы

Журбенко Василий, Б21-525

2024

На защиту

Попробуйте хотя бы один индекс, не являющийся B+tree.

Для эффективного поиска по названию добавим столбец tsvector и индексирование GIN (оптимален для индексации полей, содержащих множество значений - массивов или текстовых полей)

```
ALTER TABLE products ADD COLUMN name_tsv tsvector GENERATED ALWAYS AS
(to_tsvector('russian', name)) STORED;
CREATE INDEX idx_products_name_tsv ON products USING gin(name_tsv);
```

Результат EXPLAIN ANALYZE подтверждает, что новый индекс используется.

```
jewelry_store=# EXPLAIN ANALYZE
jewelry_store-# SELECT *
jewelry_store-# FROM products
jewelry_store-# WHERE name_tsv @@ plainto_tsquery('russian', 'серебряное
кольцо');
                                                              QUERY PLAN
--------------------------------------------------------------------------------
-------------------------------------------------
 Bitmap Heap Scan on products  (cost=12.00..16.01 rows=1 width=143) (actual
time=0.701..1.360 rows=712 loops=1)
   Recheck Cond: (name_tsv @@ '''серебрян'' & ''кольц'''::tsquery)
   Heap Blocks: exact=264
   ->  Bitmap Index Scan on idx_products_name_tsv  (cost=0.00..12.00 rows=1
width=0) (actual time=0.629..0.630 rows=712 loops=1)
         Index Cond: (name_tsv @@ '''серебрян'' & ''кольц'''::tsquery)
 Planning Time: 0.328 ms
 Execution Time: 1.456 ms
(7 rows)
```

Схема данных

Таблица `departments`

- `id` (Primary Key) - уникальный идентификатор отдела
- `name` - название отдела
- `location` - локация отдела

Таблица `products`

- `id` (Primary Key) - уникальный идентификатор товара
- `name` - название товара
- `description` - описание товара
- `purchase_price` - цена закупки
- `sale_price` - цена продажи
- `supplier_id` (Foreign Key) - идентификатор поставщика

Таблица `suppliers`

- `id` (Primary Key) - уникальный идентификатор поставщика
- `name` - название поставщика
- `contact_info` - контактная информация

Таблица `sales`

- `id` (Primary Key) - уникальный идентификатор продажи
- `product_id` (Foreign Key) - идентификатор товара
- `seller_id` (Foreign Key) - идентификатор продавца
- `sale_date` - дата продажи
- `quantity` - количество проданного товара
- `discount` - скидка

Таблица `sellers`

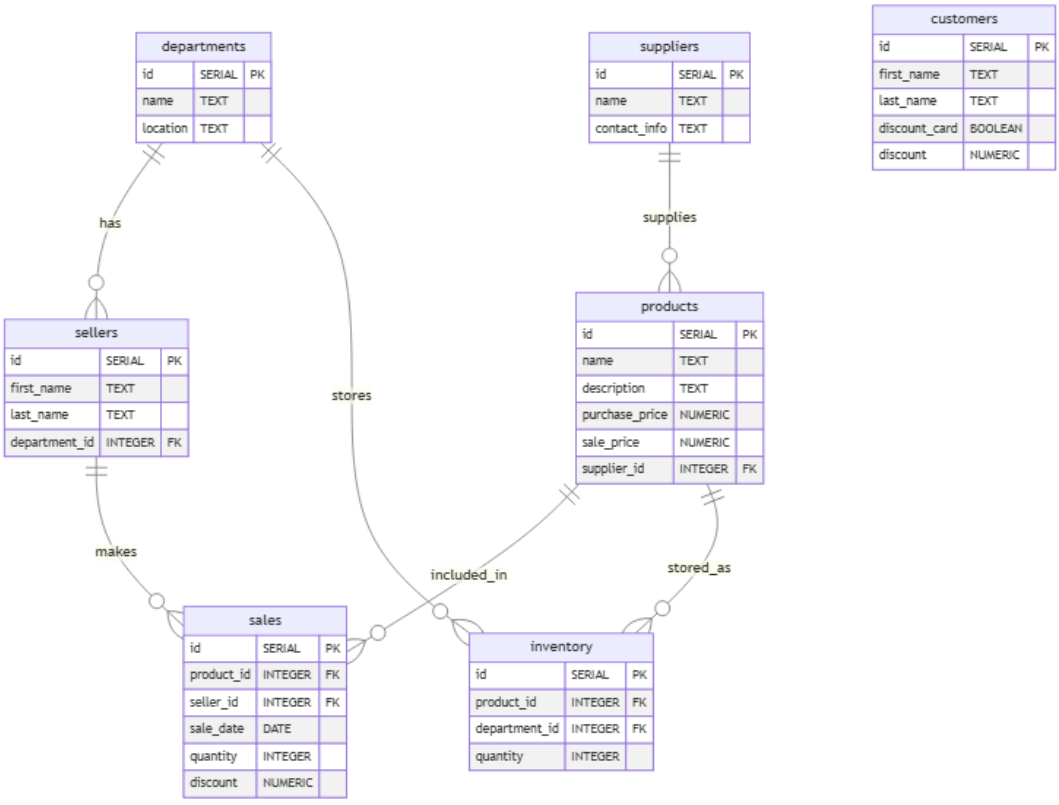
- `id` (Primary Key) - уникальный идентификатор продавца
- `first_name` - имя продавца
- `last_name` - фамилия продавца
- `department_id` (Foreign Key) - идентификатор отдела

Таблица `customers`

- `id` (Primary Key) - уникальный идентификатор покупателя
- `first_name` - имя покупателя
- `last_name` - фамилия покупателя
- `discount_card` - наличие дисконтной карты (да/нет)
- `discount` - скидка

Таблица `inventory`

- `id` (Primary Key) - уникальный идентификатор записи инвентаря
- `product_id` (Foreign Key) - идентификатор товара
- `department_id` (Foreign Key) - идентификатор отдела
- `quantity` - количество товара в отделе



Добавленние индексов

```
-- 1. Индекс для поиска товаров по названию
-- Частый запрос от клиентов и продавцов при поиске конкретных украшений
CREATE INDEX idx_products_name ON products USING btree (name);

-- 2. Составной индекс для анализа продаж по датам
-- Важен для отчетов по продажам, анализа выручки
CREATE INDEX idx_sales_date_product ON sales USING btree (sale_date,
product_id);
```

```sql
-- 3. Составной индекс для проверки наличия товаров
-- Критически важен для быстрой проверки наличия товара в конкретном отделе
CREATE INDEX idx_inventory_prod_dept ON inventory USING btree (product_id,
department_id);

-- 4. Индекс для анализа работы продавцов
-- Необходим для отчетов по эффективности продавцов
CREATE INDEX idx_sales_seller_date ON sales USING btree (seller_id,
sale_date);

-- 5. Индекс для поиска по ценовому диапазону
-- Частый запрос при подборе украшений по бюджету клиента
CREATE INDEX idx_products_price ON products USING btree (sale_price);

-- 6. Индекс для поиска клиентов с картами лояльности
-- Важен для быстрого поиска постоянных клиентов и применения скидок
CREATE INDEX idx_customers_discount ON customers USING btree (discount_card)
WHERE discount_card = true;
```

Генерация данных

Скрипт генерации

Результаты тестирования индексов

Запрос выручки по дням за последние 30 дней

```sql
EXPLAIN ANALYZE
SELECT
    s.sale_date,
    SUM(s.quantity * p.sale_price * (1 - COALESCE(s.discount, 0)/100)) as
revenue
FROM sales s
JOIN products p ON p.id = s.product_id
WHERE s.sale_date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY s.sale_date
ORDER BY s.sale_date;
```

▶ QUERY PLAN

Комментарий:

- Используется составной индекс `idx_sales_date_product` (тип: btree), который покрывает поля `sale_date` и `product_id`.
- Индекс оптимально применяется для фильтрации строк по дате (`sale_date >= CURRENT_DATE - INTERVAL '30 days'`).
- Bitmap Index Scan дополнительно ускоряет выборку строк, соответствующих указанным условиям, до выполнения хэш-соединения.
- Итоговая агрегация и сортировка выполняются эффективно за счет компактных структур данных.
- Время выполнения: 2.128 мс.

Анализ продаж по продавцам за 90 дней

```
EXPLAIN ANALYZE
SELECT
    s.seller_id,
    sel.first_name,
    sel.last_name,
    COUNT(*) as sales_count,
    SUM(s.quantity * p.sale_price * (1 - COALESCE(s.discount, 0)/100)) as
total_revenue
FROM sales s
JOIN sellers sel ON sel.id = s.seller_id
JOIN products p ON p.id = s.product_id
WHERE s.sale_date >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY s.seller_id, sel.first_name, sel.last_name
ORDER BY total_revenue DESC;
```

▶ QUERY PLAN

Комментарий:

- Используется индекс `idx_sales_date_product` (btree) для фильтрации по дате.
- Memoize позволяет кешировать результаты для повторного использования при соединении с таблицей `products`.
- Составное хэш-соединение эффективно распределяет нагрузку между таблицами `sales` и `sellers`.

Поиск товаров в ценовом диапазоне

```
EXPLAIN ANALYZE
SELECT name, sale_price
FROM products
```

```
WHERE sale_price BETWEEN 50000 AND 100000
ORDER BY sale_price;
```

▶ QUERY PLAN

Комментарий:

- Индекс `idx_products_price` (btree) не используется, так как диапазон охватывает значительную часть таблицы (3702 строки из 11008).
- PostgreSQL предпочитает последовательное сканирование (Seq Scan) из-за высокой селективности запроса.

Поиск клиентов с картами лояльности

```
EXPLAIN ANALYZE
SELECT first_name, last_name, discount
FROM customers
WHERE discount_card = true
ORDER BY discount DESC;
```

▶ QUERY PLAN

Комментарий:

- Индекс `idx_customers_discount` (partial btree) не используется из-за полного охвата таблицы (6039 строк).
- Последовательное сканирование предпочтительнее, так как все строки удовлетворяют условию фильтрации.

Проверка наличия конкретного товара в отделах

```
EXPLAIN ANALYZE
SELECT d.name as department, i.quantity
FROM inventory i
JOIN departments d ON d.id = i.department_id
WHERE i.product_id = 500;
```

▶ QUERY PLAN

Комментарий:

- Индекс `idx_inventory_prod_dept` (btree) используется для быстрого доступа к строкам, соответствующим фильтру `product_id = 500`.

- Memoize позволяет избежать повторных обращений к таблице `departments`, но в данном случае не используется, так как результат пустой.
- Время выполнения минимально благодаря индексации (0.029 мс).

Заключение

Проведен анализ производительности запросов в PostgreSQL с использованием индексов. Были протестированы различные типы запросов, включая агрегацию, фильтрацию, сортировку и соединение таблиц, с оценкой их плана выполнения.

Полученные результаты:

1. Для фильтрации по дате использовался индекс idx_sales_date_product, что позволило сократить время выполнения запроса до 2.128 мс.
2. Анализ продаж по продавцам за 90 дней занял 6.517 мс, с применением индекса и оптимизацией за счет Memoize.
3. Последовательное сканирование (Seq Scan) оказалось предпочтительным для запросов с большим охватом данных (например, поиск товаров в ценовом диапазоне занял 3.606 мс).
4. Индексы в небольших таблицах, таких как customers и inventory, были эффективны, но из-за малых объемов данных PostgreSQL иногда выбирал последовательное сканирование.