

Лабораторна робота №6

Оптимізація та масштабування бази даних TechMarket

Виконав: Кіщук Ярослав

Зміст

1 Вступ	3
1.1 Проблематика	3
1.2 Мета та завдання	3
2 Методологія аналізу продуктивності	3
2.1 Інструмент EXPLAIN ANALYZE	3
2.2 Метрики для оцінки	4
3 Базові виміри продуктивності	4
3.1 Стан бази даних	4
3.2 Базові запити для аналізу	4
3.3 Результати EXPLAIN ANALYZE до оптимізації	4
4 Стратегія оптимізації	5
4.1 Рівень 1: Індексація	5
4.1.1 Композитні індекси	5
4.1.2 Індекси для COUNT DISTINCT	5
4.1.3 Покриваючі індекси (Covering Indexes)	5
4.1.4 BRIN індекси	6
4.2 Рівень 2: Матеріалізовані представлення	6
4.2.1 Концепція	6
4.2.2 MV 1: Місячна статистика продажів	6
4.2.3 MV 2: Аналітика продуктів	7
4.2.4 MV 3: Регіональна статистика	7
4.2.5 MV 4: Клієнтська аналітика	7
4.3 Рівень 3: Оновлення статистики	8
5 Результати оптимізації	8
5.1 Створені структури	8
5.2 Просторові витрати	9
5.3 Порівняння продуктивності	9
5.4 Приклад оптимізованого запиту	9
6 Автоматизація підтримки	10
6.1 Python скрипт для оновлення MV	10
6.2 Інтеграція з Airflow	10
6.3 Стратегія оновлення	11
7 Масштабування для великих даних	11
7.1 Поточний стан (1,600 рядків)	11
7.2 Прогноз для 10,000+ рядків	11
8 Моніторинг та діагностика	11
8.1 Перевірка використання індексів	11
8.2 Перевірка свіжості MV	12
8.3 Виявлення повільних запитів	12
9 Висновки	12
9.1 Досягнуті результати	12

1 Вступ

Метою шостої лабораторної роботи є оптимізація продуктивності аналітичних запитів до сховища даних TechMarket DWH. У попередніх лабораторних роботах було створено повноцінну архітектуру даних: операційні бази даних (OLTP), процес ETL для завантаження даних, аналітичне сховище даних (DWH) та систему бізнес-аналітики з п'ятьма ключовими показниками ефективності (KPI).

Однак зі зростанням обсягу даних продуктивність аналітичних запитів може погіршуватися. Тому необхідно провести систематичний аналіз продуктивності, виявити вузькі місця та застосувати відповідні техніки оптимізації.

1.1 Проблематика

Типові проблеми продуктивності в аналітичних системах:

- **Повні скани таблиць** (Sequential Scan) замість використання індексів
- **Повторювані складні обчислення** при кожному запиті
- **Неефективні з'єднання** великих таблиць без індексів
- **Агрегації** на великих обсягах даних при кожному запиті
- **COUNT DISTINCT** операції з сортуванням/хешуванням

1.2 Мета та завдання

Завдання 1: Провести аналіз продуктивності поточних KPI-запитів за допомогою EXPLAIN ANALYZE.

Завдання 2: Ідентифікувати вузькі місця: повні скани таблиць, неоптимальні з'єднання, повільні агрегації.

Завдання 3: Створити композитні та покриваючі індекси для прискорення найчастіших запитів.

Завдання 4: Реалізувати матеріалізовані представлення для попередньо обчислених агрегацій.

Завдання 5: Порівняти продуктивність запитів до та після оптимізації.

Завдання 6: Розробити стратегію підтримки оптимізацій (оновлення статистики, refresh views).

Завдання 7: Створити автоматизацію для регулярного оновлення оптимізаційних структур.

2 Методологія аналізу продуктивності

2.1 Інструмент EXPLAIN ANALYZE

PostgreSQL надає потужний інструмент EXPLAIN ANALYZE, який виконує запит та повертає детальну інформацію про план виконання:

- **Planning Time** — час на побудову плану виконання
- **Execution Time** — фактичний час виконання запиту
- **Node Types** — типи операцій (Seq Scan, Index Scan, Hash Join, тощо)
- **Buffers** — інформація про використання кешу та дискових операцій
- **Cost Estimates** — оцінки вартості операцій планувальником

2.2 Метрики для оцінки

Для кожного запиту аналізуємо:

1. **Час виконання** — основна метрика продуктивності
2. **Типи сканувань** — Seq Scan (погано) vs Index Scan (добре)
3. **Типи з'єднань** — Hash Join, Nested Loop, Merge Join
4. **Використання пам'яті** — work_mem, shared_buffers
5. **Кількість прочитаних рядків** — actual rows vs estimated rows

3 Базові виміри продуктивності

3.1 Стан бази даних

Перед початком оптимізації:

- Кількість записів у fact_sales: 1,614
- Кількість індексів: тільки первинні ключі та unique constraints
- Матеріалізовані представлення: відсутні
- Загальний розмір таблиць: ~500 KB

3.2 Базові запити для аналізу

Проаналізовано п'ять KPI-запитів з попередньої лабораторної роботи:

Табл. 1: KPI запити для оптимізації

Запит	Характеристики
Revenue by Month	JOIN 3 таблиць, GROUP BY, SUM агрегації, фільтр за датою
Orders by Region	JOIN 2 таблиць, COUNT DISTINCT, GROUP BY регіонам
Average Order Value	3 JOIN'и, COUNT DISTINCT, ділення, NULL-безпечність
Margin Percentage	2 JOIN'и, SUM агрегації, обчислення відсотку
Top Products	3 JOIN'и, GROUP BY продуктам, ORDER BY, LIMIT 10

3.3 Результати EXPLAIN ANALYZE до оптимізації

Типовий план виконання для запиту "Revenue by Month":

Лістинг 1: Приклад неоптимізованого запиту

```
1 EXPLAIN (ANALYZE, BUFFERS, TIMING)
2 SELECT
3     d.year, d.month,
```

```

4     SUM(f.revenue) AS revenue
5 FROM fact_sales f
6 JOIN dim_date d ON f.date_key = d.date_key
7 WHERE d.date >= '2024-01-01'
8 GROUP BY d.year, d.month;

```

Проблеми виявлені:

- **Seq Scan on fact_sales** — повне сканування 1,614 рядків
- **Hash Join** — побудова хеш-таблиці в пам'яті
- **HashAggregate** — додаткове хешування для GROUP BY
- **Час виконання:** ~25 мс (для невеликого датасету)

4 Стратегія оптимізації

4.1 Рівень 1: Індексація

4.1.1 Композитні індекси

Створення індексів на комбінації колонок, які часто використовуються разом у WHERE та JOIN:

Лістинг 2: Композитні індекси для fact_sales

```

1 CREATE INDEX idx_fact_sales_date_product
2   ON fact_sales(date_key, product_key);
3
4 CREATE INDEX idx_fact_sales_date_region
5   ON fact_sales(date_key, region_key);
6
7 CREATE INDEX idx_fact_sales_date_customer
8   ON fact_sales(date_key, customer_key);

```

Обґрунтування: Більшість аналітичних запитів фільтрують за датою та агрегують за іншим виміром. Композитний індекс дозволяє ефективно знайти потрібний діапазон дат та одразу отримати відповідні ключі інших вимірів.

4.1.2 Індекси для COUNT DISTINCT

Операції COUNT(DISTINCT order_id) вимагають унікалізації значень:

Лістинг 3: Індекс для оптимізації COUNT DISTINCT

```

1 CREATE INDEX idx_fact_sales_order_id
2   ON fact_sales(order_id);

```

4.1.3 Покриваючі індекси (Covering Indexes)

Індекси, що включають додаткові колонки для уникнення звернень до основної таблиці:

Лістинг 4: Покриваючий індекс для агрегацій

```

1 CREATE INDEX idx_fact_sales_product_revenue
2   ON fact_sales(product_key)
3   INCLUDE (revenue, quantity, margin, discount_amount);

```

Перевага: PostgreSQL може виконати Index-Only Scan, не звертаючись до основної таблиці.

4.1.4 BRIN індекси

Для послідовних даних (наприклад, date_key), BRIN індекси забезпечують компактність:

Лістинг 5: BRIN індекс для date_key

```
1 CREATE INDEX idx_fact_sales_date_brin  
2   ON fact_sales USING BRIN(date_key);
```

Переваги BRIN:

- Займають у 10-100 разів менше місця, ніж B-tree індекси
- Ефективні для послідовно впорядкованих даних
- Підходять для великих таблиць із природним порядком

4.2 Рівень 2: Матеріалізовані представлення

4.2.1 Концепція

Матеріалізовані представлення (Materialized Views) — це попередньо обчислені та збережені результати запитів. На відміну від звичайних VIEW, вони фізично зберігають дані на диску.

Переваги:

- Запити виконуються у 10-1000 разів швидше
- Складні агрегації обчислюються один раз
- Можна створювати індекси на матеріалізованих views

Недоліки:

- Потребують додаткового місця на диску
- Дані не оновлюються автоматично (потрібен REFRESH)
- Можливий lag між реальними даними та MV

4.2.2 MV 1: Місячна статистика продажів

Лістинг 6: Матеріалізоване представлення для місячних агрегацій

```
1 CREATE MATERIALIZED VIEW mv_monthly_sales AS  
2 SELECT  
3   d.year, d.month, d.quarter,  
4   r.region_key, r.name AS region_name,  
5   COUNT(DISTINCT f.order_id) AS orders_count,  
6   SUM(f.revenue) AS total_revenue,  
7   SUM(f.discount_amount) AS total_discount,  
8   SUM(f.margin) AS total_margin,  
9   AVG(f.revenue) AS avg_item_revenue  
10  FROM fact_sales f
```

```

11 JOIN dim_date d ON f.date_key = d.date_key
12 LEFT JOIN dim_region r ON f.region_key = r.region_key
13 GROUP BY d.year, d.month, d.quarter,
14     r.region_key, r.name;
15
16 CREATE INDEX idx_mv_monthly_sales_year_month
17     ON mv_monthly_sales(year, month);

```

Використання: Замість щоразу з'єднувати fact_sales з dim_date, запитуємо готові агрегати з mv_monthly_sales.

4.2.3 MV 2: Аналітика продуктів

Лістинг 7: Продуктова аналітика

```

1 CREATE MATERIALIZED VIEW mv_product_performance AS
2 SELECT
3     p.product_key, p.name AS product_name,
4     c.name AS category_name,
5     COUNT(DISTINCT f.order_id) AS orders_count,
6     SUM(f.quantity) AS total_quantity,
7     SUM(f.revenue) AS total_revenue,
8     SUM(f.margin) AS total_margin,
9     MAX(d.date) AS last_sale_date
10    FROM fact_sales f
11   JOIN dim_product p ON f.product_key = p.product_key
12  LEFT JOIN dim_category c ON p.category_key = c.category_key
13  JOIN dim_date d ON f.date_key = d.date_key
14 GROUP BY p.product_key, p.name, c.name;
15
16 CREATE INDEX idx_mv_product_performance_revenue
17     ON mv_product_performance(total_revenue DESC);

```

4.2.4 MV 3: Регіональна статистика

Лістинг 8: Регіональна аналітика

```

1 CREATE MATERIALIZED VIEW mvRegionalPerformance AS
2 SELECT
3     r.region_key, r.name AS region_name,
4     COUNT(DISTINCT f.order_id) AS orders_count,
5     COUNT(DISTINCT f.customer_key) AS unique_customers,
6     SUM(f.revenue) AS total_revenue,
7     SUM(f.margin) AS total_margin,
8     AVG(f.revenue) AS avg_order_value
9     FROM fact_sales f
10    JOIN dim_region r ON f.region_key = r.region_key
11 GROUP BY r.region_key, r.name;

```

4.2.5 MV 4: Клієнтська аналітика

Лістинг 9: Поведінка клієнтів

```

1 CREATE MATERIALIZED VIEW mv_customer_performance AS
2 SELECT
3     c.customer_key, c.email,
4     r.name AS region_name,
5     COUNT(DISTINCT f.order_id) AS orders_count,
6     SUM(f.revenue) AS total_revenue,
7     AVG(f.revenue) AS avg_order_value,
8     MAX(d.date) AS last_purchase_date
9 FROM fact_sales f
10 JOIN dim_customer c ON f.customer_key = c.customer_key
11 LEFT JOIN dim_region r ON c.region_key = r.region_key
12 JOIN dim_date d ON f.date_key = d.date_key
13 GROUP BY c.customer_key, c.email, r.name;

```

4.3 Рівень 3: Оновлення статистики

Лістинг 10: Database maintenance commands

```

1 VACUUM ANALYZE fact_sales;
2 VACUUM ANALYZE dim_date;
3 VACUUM ANALYZE dim_product;
4
5 ANALYZE mv_monthly_sales;
6 ANALYZE mv_product_performance;

```

Призначення:

- VACUUM — очищає застарілі версії рядків (dead tuples)
- ANALYZE — оновлює статистику для оптимізатора запитів
- Планувальник використовує актуальну статистику для вибору оптимального плану

5 Результати оптимізації

5.1 Створені структури

Табл. 2: Створені оптимізаційні структури

Компонент	Кількість	Опис
Індекси на fact_sales	14	Композитні, покриваючі, BRIN індекси
Індекси на dimensions	11	Первинні ключі + додаткові для JOIN'ів
Матеріалізовані views	4	Місячні, продуктові, регіональні, клієнтські
Індекси на MV	10	Для швидкого доступу до агрегатів
Всього	39	Індексів + матеріалізованих представлень

5.2 Просторові витрати

Табл. 3: Використання дискового простору

Компонент	Розмір	Примітки
fact_sales (таблиця)	500 KB	Базові дані
fact_sales (індекси)	400 KB	14 індексів
mv_monthly_sales	80 KB	113 рядків
mv_product_performance	80 KB	25 рядків
mvRegional_performance	64 KB	5 рядків
mvCustomer_performance	72 KB	50 рядків
Накладні витрати	696 KB	Індекси + MV
Загальний розмір	1196 KB	Приріст +139%

Висновок: Додаткові 700 KB — прийнятна ціна за покращення продуктивності у 100-300 разів.

5.3 Порівняння продуктивності

Табл. 4: Виміри продуктивності до та після оптимізації

Запит	До (мс)	Після (мс)	Покращення
Revenue by Month	~25	0.081	308x швидше
Orders by Region	~18	~0.1	180x швидше
Average Order Value	~20	~0.1	200x швидше
Margin Percentage	~20	~0.1	200x швидше
Top Products	~30	~0.2	150x швидше
Середнє	22.6	0.114	198x швидше

5.4 Приклад оптимізованого запиту

Оригінальний запит (25 мс):

```
1 SELECT d.year, d.month, SUM(f.revenue)
2 FROM fact_sales f
3 JOIN dim_date d ON f.date_key = d.date_key
4 WHERE d.date >= '2024-01-01'
5 GROUP BY d.year, d.month;
```

Оптимізований запит (0.081 мс):

```
1 SELECT year, month, SUM(total_revenue) AS revenue
2 FROM mv_monthly_sales
3 WHERE year = 2024
4 GROUP BY year, month;
```

Чому швидше:

- Немає JOIN — дані вже з'єднані

- Немає складних агрегацій — вже обчислені
- Сканується 12 рядків замість 1,614
- Sequential Scan оптимальний для малих таблиць

6 Автоматизація підтримки

6.1 Python скрипт для оновлення MV

Створено scripts/refresh_materialized_views.py:

Лістинг 11: Використання скрипта оновлення

```
# Setup environment
export DWH_HOST=localhost
export DWH_USER=dwh_user
export DWH_PASS=dwh_pass

# Run refresh
python scripts/refresh_materialized_views.py
```

Функціонал скрипта:

- Підключення до DWH через SQLAlchemy
- Послідовне оновлення всіх 4 матеріалізованих views
- Підтримка CONCURRENTLY (без блокування читання)
- Логування часу оновлення кожного view
- Відображення розмірів та кількості рядків

6.2 Інтеграція з Airflow

Додавання task до ETL DAG:

Лістинг 12: Інтеграція в Airflow

```
refresh_views = BashOperator(
    task_id='refresh_materialized_views',
    bash_command='python /opt/airflow/scripts/
        refresh_materialized_views.py',
    dag=dag
)

# Dependencies
run_etl >> refresh_views >> success_notification
```

6.3 Стратегія оновлення

Табл. 5: План підтримки оптимізацій

Частота	Операція	Команда
Після кожного ETL	Refresh MV	python scripts/refresh_materialized_views.py
Щотижня	Update statistics	VACUUM ANALYZE fact_sales
Щомісяця	Check index usage	SELECT * FROM pg_stat_user_indexes
Щокварталу	Reindex (за потреби)	REINDEX TABLE fact_sales

7 Масштабування для великих даних

7.1 Поточний стан (1,600 рядків)

- ✓ Всі запити < 1 мс
- ✓ Індекси ефективно використовуються
- ✓ Матеріалізовані views забезпечують оптимальну продуктивність

7.2 Прогноз для 10,000+ рядків

- Поточні оптимізації залишаються ефективними
- Час виконання зросте пропорційно: 0.1 мс → 0.5-1 мс
- BRIN індекси стануть ще більш корисними
- Розмір MV збільшиться незначно (агрегати)

8 Моніторинг та діагностика

8.1 Перевірка використання індексів

Лістинг 13: Статистика використання індексів

```
1 SELECT
2   schemaname ,
3   tablename ,
4   indexname ,
5   idx_scan AS times_used ,
6   idx_tup_read ,
7   idx_tup_fetch
8   FROM pg_stat_user_indexes
9   WHERE schemaname = 'public' AND idx_scan > 0
10  ORDER BY idx_scan DESC;
```

Метрики:

- `idx_scan` — скільки разів індекс використовувався
- `idx_tup_read` — кількість прочитаних записів з індексу
- `idx_tup_fetch` — кількість записів, отриманих з таблиці

8.2 Перевірка свіжості MV

Лістинг 14: Статус матеріалізованих views

```

1  SELECT
2      matviewname,
3      pg_size.pretty(pg_total_relation_size('public.' || matviewname)) AS
4          size,
5      n_live_tup AS row_count
6  FROM pg_matviews
7  LEFT JOIN pg_stat_user_tables ON tablename = matviewname
8  WHERE schemaname = 'public';

```

8.3 Виявлення повільних запитів

Лістинг 15: Аналіз продуктивності запиту

```

1 EXPLAIN (ANALYZE, BUFFERS, TIMING)
2 SELECT ... FROM ...;

```

На що звертати увагу:

- **Seq Scan** на великих таблицях → потрібен індекс
- **Hash Join з великими таблицями** → можливо, потрібна денормалізація
- **Sort operations** → індекс може допомогти з ORDER BY
- **Buffers: shared read** → багато дискових операцій, погане кешування

9 Висновки

У ході виконання лабораторної роботи №6 було проведено комплексну оптимізацію аналітичного сховища даних TechMarket. Застосовано систематичний підхід: аналіз продуктивності, виявлення вузьких місць, реалізація оптимізацій та верифікація результатів.

9.1 Досягнуті результати

1. Створено 25 індексів:
 - 14 індексів на `fact_sales` (композитні, покриваючі, BRIN)
 - 11 індексів на dimension-таблицях
2. Реалізовано 4 матеріалізовані представлення:
 - Місячна статистика продажів (113 рядків)
 - Аналітика продуктів (25 рядків)

- Регіональна статистика (5 рядків)
- Клієнтська аналітика (50 рядків)

3. Покращення продуктивності:

- Середнє покращення: **198x швидше**
- Найкраще: Revenue by Month — **308x швидше**
- Усі запити тепер виконуються < 1 мс

4. Автоматизація:

- Python-скрипт для оновлення матеріалізованих views
- Bash-скрипт для тестування оптимізацій
- Інтеграція з Airflow для автоматичного refresh

Оптимізація бази даних — це ітеративний процес, що вимагає постійного моніторингу та адаптації до змін у навантаженні та обсягах даних. Створена в цій роботі основа забезпечує ефективну аналітичну систему TechMarket з покращенням продуктивності у середньому в 198 разів.