МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.
Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №8
по дисциплине: Базы данных
тема: «Оптимизация sql-запросов в СУБД с использованием планировщика»

Выполнил: ст. группы ПВ-223
Дмитриев Андрей
Александрович

Проверил:
Панченко Максим Владимирович

Белгород 2024 г.

Вариант 2.

Цель работы: получение навыков повышения производительности работы СУБД с помощью оптимизации sql-запросов с использованием планировщика.

Задания:

1. Сгенерировать данные для таблиц, используемых в sql-запросах лабораторной работы №4 (1000 записей).
2. Построить планы исполнения запросов, созданных в процессе выполнения лабораторной работы №4.
3. Повлиять на производительность с использованием методов управления планировщиком. Предложить способы ускорения выполнения запросов. Продемонстрировать планы исполнения запросов.
4. Провести сравнительный анализ выполнения запросов при 100 / 1000 записей в таблицах.

Код генерации запросов:

```
fun testExecuteProcess() {
    connect()
        .onFailure { e -> println("Connection failed: $e") }
        .getOrNull()?.use { conn ->
            conn
                .executeDeleteDb()
                .executeInitDb()
                .insertBaseDataForTest()
                .insertRandomData(100)
        }
}

fun Connection.insertBaseDataForTest(): Connection {
    execute(
        "INSERT INTO ${Table.Store.T_NAME} " +
                "(id, address) VALUES " +
                "(1, 'LA, 5 Avenue'), " +
                "(2, 'LA, 11 Avenue')," +
                "(3, 'LA, 12 Avenue');"
    )

    return this
}

fun Connection.insertRandomData(rows: Int): Connection {
    val wordSet1 = setOf(
        "Колбаса",
        "Слойка",
        "Морковь",
        "Картошка",
        "Свекла",
        "Несъедобная"
    )
    val wordSet2 = setOf(
        "докторская",
        "студенческая",
        "польская",
        "французская",
        "русская",
        "испанская",
        "финская",
```

```
        "китайская",
        "корейская"
    )

    val categories = setOf(
        Category.BAKE.value, Category.MILK.value, Category.MEAT.value,
    )

    val quantityToAssesses = setOf(
        QuantityToAssess.PIECE.value, QuantityToAssess.GRAM_100.value,
QuantityToAssess.KILOGRAM_1.value,
    )

    val storeIdSet = setOf(
        1L, 2L
    )

    val random = Random()
    for (productArticle in 1..rows) {
        val productName = wordSet1.random() + " " + wordSet2.random()
        val productCategory = categories.random()
        val quantityToAssess = quantityToAssesses.random()
        execute(
            "insert into ${Table.Product.T_NAME} (article, name, category,
quantity_to_assess) " +
                    "values($productArticle, '$productName',
'$productCategory', '$quantityToAssess')"
        )

        val storeId = storeIdSet.random()
        val cost = random.nextDouble(60.0, 300.0)
        val amount = random.nextDouble(100.0, 500.0)
        execute(
            "insert into ${Table.Accounting.T_NAME} (store_id,
product_article, cost, amount) " +
                    "values($storeId, $productArticle, $cost, $amount)"
        )

        val intervalTime = random.nextInt(1, 10)
        execute(
            "insert into ${Table.CheckList.T_NAME} (store_id, time) " +
                    "values($storeId, now() + interval '$intervalTime
hour')"
        )
    }

    for (i in 1..5 * rows) {
        val checkListId = random.nextInt(1, rows/2)
        val article = random.nextInt(1, rows)
        val amount = random.nextDouble(10.0, 50.0)

        execute(
            "insert into ${Table.Purchase.T_NAME} (check_list_id,
product_article, amount)" +
                    "values($checkListId, $article, $amount)"
        )
    }

    return this
}
```

Запрос на получение чека:

```
explain
select  t2.name as prod_name,
```

```
        t1.amount,
        t2.quantity_to_assess,
        t1.amount*t3.cost as total_cost,
        (select time from check_list where id=2 limit 1) as time
    from purchase as t1
    inner join product as t2
        on t1.product_article=t2.article
    inner join accounting as t3
        on t1.product_article=t3.product_article
    where t1.check_list_id=2
    order by prod_name asc;
```

Замеры без ускорения (100 записей):
(cost=19.78..19.80 rows=11 width=84)

```
"Sort  (cost=19.78..19.80 rows=11 width=84)"
"  Sort Key: t2.name"
"  InitPlan 1 (returns $0)"
"    ->  Limit  (cost=0.00..2.25 rows=1 width=8)"
"          ->  Seq Scan on check_list  (cost=0.00..2.25 rows=1 width=8)"
"                Filter: (id = 4)"
"  ->  Nested Loop  (cost=3.39..17.34 rows=11 width=84)"
"        Join Filter: (t2.article = t1.product_article)"
"        ->  Hash Join  (cost=3.25..13.65 rows=11 width=28)"
"              Hash Cond: (t1.product_article = t3.product_article)"
"              ->  Seq Scan on purchase t1  (cost=0.00..10.25 rows=11 width=14)"
"                    Filter: (check_list_id = 4)"
"              ->  Hash  (cost=2.00..2.00 rows=100 width=14)"
"                    ->  Seq Scan on accounting t3  (cost=0.00..2.00 rows=100
width=14)"
"        ->  Index Scan using product_pkey on product t2  (cost=0.14..0.32
rows=1 width=46)"
"              Index Cond: (article = t3.product_article)"
```

Замеры без ускорения (5000 записей):
(cost=634.30..634.32 rows=10 width=84)

```
"Sort  (cost=634.30..634.32 rows=10 width=84)"
"  Sort Key: t2.name"
"  InitPlan 1 (returns $0)"
"    ->  Limit  (cost=0.28..8.30 rows=1 width=8)"
"          ->  Index Scan using check_list_pkey on check_list  (cost=0.28..8.30
rows=1 width=8)"
"                Index Cond: (id = 4)"
"  ->  Nested Loop  (cost=149.78..625.83 rows=10 width=84)"
"        Join Filter: (t2.article = t1.product_article)"
"        ->  Hash Join  (cost=149.50..622.14 rows=10 width=28)"
"              Hash Cond: (t1.product_article = t3.product_article)"
"              ->  Seq Scan on purchase t1  (cost=0.00..472.50 rows=10
width=14)"
"                    Filter: (check_list_id = 4)"
"              ->  Hash  (cost=87.00..87.00 rows=5000 width=14)"
"                    ->  Seq Scan on accounting t3  (cost=0.00..87.00 rows=5000
width=14)"
```

```
"        ->  Index Scan using product_pkey on product t2  (cost=0.28..0.35
rows=1 width=46)"
"              Index Cond: (article = t3.product_article)"
```

Замеры с ускорением (100 записей):
(cost=19.78..19.80 rows=11 width=84)

```
"Sort  (cost=19.78..19.80 rows=11 width=84)"
"  Sort Key: t2.name"
"  InitPlan 1 (returns $0)"
"    ->  Limit  (cost=0.00..2.25 rows=1 width=8)"
"          ->  Seq Scan on check_list  (cost=0.00..2.25 rows=1 width=8)"
"                Filter: (id = 4)"
"  ->  Nested Loop  (cost=3.39..17.34 rows=11 width=84)"
"        Join Filter: (t2.article = t1.product_article)"
"        ->  Hash Join  (cost=3.25..13.65 rows=11 width=28)"
"              Hash Cond: (t1.product_article = t3.product_article)"
"              ->  Seq Scan on purchase t1  (cost=0.00..10.25 rows=11 width=14)"
"                    Filter: (check_list_id = 4)"
"              ->  Hash  (cost=2.00..2.00 rows=100 width=14)"
"                    ->  Seq Scan on accounting t3  (cost=0.00..2.00 rows=100
width=14)"
"        ->  Index Scan using product_pkey on product t2  (cost=0.14..0.32
rows=1 width=46)"
"              Index Cond: (article = t3.product_article)"
```

Замеры с ускорением (5000 записей):
(cost=555.76..555.79 rows=10 width=84)

```
"Sort  (cost=555.76..555.79 rows=10 width=84)"
"  Sort Key: t2.name"
"  InitPlan 1 (returns $0)"
"    ->  Limit  (cost=0.28..8.30 rows=1 width=8)"
"          ->  Index Scan using check_list_pkey on check_list  (cost=0.28..8.30
rows=1 width=8)"
"                Index Cond: (id = 4)"
"  ->  Nested Loop  (cost=0.56..547.29 rows=10 width=84)"
"        Join Filter: (t2.article = t1.product_article)"
"        ->  Nested Loop  (cost=0.28..543.60 rows=10 width=28)"
"              ->  Seq Scan on purchase t1  (cost=0.00..472.50 rows=10
width=14)"
"                    Filter: (check_list_id = 4)"
"              ->  Index Scan using my_index_2 on accounting t3
(cost=0.28..7.10 rows=1 width=14)"
"                    Index Cond: (product_article = t1.product_article)"
"        ->  Index Scan using product_pkey on product t2  (cost=0.28..0.35
rows=1 width=46)"
"              Index Cond: (article = t3.product_article)"
```

Сумма среднего чека по определённому магазину за установленное время:

```
explain
```

```
select  t0.store_id,
        round(avg(t1.amount*t2.cost), 2) as avg_sum_of_checks
    from check_list as t0
    inner join purchase as t1
    on t0.id=t1.check_list_id
    inner join accounting as t2
    on t1.product_article=t2.product_article
    where t0.time<now()
    group by t0.store_id
    having t0.store_id=1;
```

Замеры без ускорения (100 записей):
(cost=13.19..15..66 rows=1 width=40)

```
"GroupAggregate  (cost=13.19..15.66 rows=1 width=40)"
"  ->  Hash Join  (cost=13.19..15.62 rows=5 width=20)"
"        Hash Cond: (t2.product_article = t1.product_article)"
"        ->  Seq Scan on accounting t2  (cost=0.00..2.00 rows=100 width=14)"
"        ->  Hash  (cost=13.13..13.13 rows=5 width=22)"
"              ->  Hash Join  (cost=2.76..13.13 rows=5 width=22)"
"                    Hash Cond: (t1.check_list_id = t0.id)"
"                    ->  Seq Scan on purchase t1  (cost=0.00..9.00 rows=500
width=22)"
"                    ->  Hash  (cost=2.75..2.75 rows=1 width=16)"
"                          ->  Seq Scan on check_list t0  (cost=0.00..2.75
rows=1 width=16)"
"                                Filter: ((store_id = 1) AND (""time"" <
now()))"
```

Замеры без ускорения (5000 записей):
(cost=595.26..701.10 rows=1 width=40)

```
"GroupAggregate  (cost=595.26..701.10 rows=1 width=40)"
"  ->  Hash Join  (cost=595.26..701.06 rows=5 width=20)"
"        Hash Cond: (t2.product_article = t1.product_article)"
"        ->  Seq Scan on accounting t2  (cost=0.00..87.00 rows=5000 width=14)"
"        ->  Hash  (cost=595.19..595.19 rows=5 width=22)"
"              ->  Hash Join  (cost=119.51..595.19 rows=5 width=22)"
"                    Hash Cond: (t1.check_list_id = t0.id)"
"                    ->  Seq Scan on purchase t1  (cost=0.00..410.00 rows=25000
width=22)"
"                    ->  Hash  (cost=119.50..119.50 rows=1 width=16)"
"                          ->  Seq Scan on check_list t0  (cost=0.00..119.50
rows=1 width=16)"
"                                Filter: ((store_id = 1) AND (""time"" <
now()))"
```

Замеры с ускорением (100 записей):
(cost=2.91..14.14 rows=1 width=40)

```
"GroupAggregate  (cost=2.91..14.14 rows=1 width=40)"
"  ->  Nested Loop  (cost=2.91..14.10 rows=5 width=20)"
"        ->  Hash Join  (cost=2.76..13.13 rows=5 width=22)"
"              Hash Cond: (t1.check_list_id = t0.id)"
"              ->  Seq Scan on purchase t1  (cost=0.00..9.00 rows=500 width=22)"
```

```
"              ->  Hash  (cost=2.75..2.75 rows=1 width=16)"
"                    ->  Seq Scan on check_list t0  (cost=0.00..2.75 rows=1
width=16)"
"                          Filter: ((store_id = 1) AND (""time"" < now()))"
"        ->  Index Scan using my_index_2 on accounting t2  (cost=0.14..0.18
rows=1 width=14)"
"              Index Cond: (product_article = t1.product_article)"
```

Замеры с ускорением (5000 записей):
(cost=119.80..596.83 rows=1 width=40)

```
"GroupAggregate  (cost=119.80..596.83 rows=1 width=40)"
"  ->  Nested Loop  (cost=119.80..596.79 rows=5 width=20)"
"        ->  Hash Join  (cost=119.51..595.19 rows=5 width=22)"
"              Hash Cond: (t1.check_list_id = t0.id)"
"              ->  Seq Scan on purchase t1  (cost=0.00..410.00 rows=25000
width=22)"
"              ->  Hash  (cost=119.50..119.50 rows=1 width=16)"
"                    ->  Seq Scan on check_list t0  (cost=0.00..119.50 rows=1
width=16)"
"                          Filter: ((store_id = 1) AND (""time"" < now()))"
"        ->  Index Scan using my_index_2 on accounting t2  (cost=0.28..0.31
rows=1 width=14)"
"              Index Cond: (product_article = t1.product_article)"
```

Применяемая оптимизация:

```
create index my_index_1 on accounting (store_id);
create index my_index_2 on accounting (product_article);
```

Вывод: в ходе работы получены навыки повышения производительности работы СУБД с помощью оптимизации sql-запросов с использованием планировщика.

В сравнение участвовали две выборки. После применения описанной оптимизации запросы начали работать быстрее, но в «запросе чека» для 100 записей примененные оптимизации были проигнорированы – план запроса идентичен плану запроса для неоптимизированной таблицы.