**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)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №8**

по дисциплине: Базы данных

тема: «Оптимизация SQL-запросов в СУБД с использованием планировщика»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Панченко Максим Владимирович

Белгород 2024 г.

**Лабораторная работа №8**

Оптимизация SQL-запросов в СУБД с использованием планировщика  
Вариант 8

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

Выполним генерацию данных при помощи faker, дополнив прошлую программу:

fake.py

import sys  
  
from PySide6.QtWidgets import QApplication  
from sqlalchemy.orm import Session  
  
from core.db import engine  
from models import Home, Contract, Resident, Payment, Task, Worker  
from faker import Faker  
from datetime import date, timedelta  
from widgets.main import MainDialog  
import random  
  
MAX\_COUNT = 100  
  
homes = []  
contracts = []  
residents = []  
payments = []  
tasks = []  
workers = []  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 fake = Faker()  
 with Session(engine) as session:  
 print("Generating homes...")  
 for i in range(0, MAX\_COUNT):  
 new\_home = Home(  
 address=fake.address(),  
 commisioning=fake.date\_between(  
 start\_date=(date.today() + timedelta(days=500)),  
 end\_date=(date.today() + timedelta(days=1000))),  
 floors=fake.random\_int(1, 50),  
 index=fake.random\_int(100000, 999999)  
 )  
  
 session.add(new\_home)  
 session.commit()  
 homes.append(new\_home)  
  
 print("Generating contracts...")  
 for i in range(0, MAX\_COUNT):  
 new\_contract = Contract(  
 transaction\_date=fake.date\_between(  
 start\_date=(date.today() + timedelta(days=500)),  
 end\_date=(date.today() + timedelta(days=1000))),  
 until\_date=fake.date\_between(  
 start\_date=(date.today() + timedelta(days=1001)),  
 end\_date=(date.today() + timedelta(days=1500))),  
 home\_address=random.choice(homes).address  
 )  
  
 session.add(new\_contract)  
 session.commit()  
 contracts.append(new\_contract)  
  
 prefix\_phone = [  
 "+7921",  
 "+7931",  
 "+7911",  
 "+7953",  
 "+7995",  
 "+7912",  
 "+7991",  
 "+7999",  
 "+7956",  
 "+7900",  
 ]  
  
 print("Generating residents...")  
 for i in range(0, MAX\_COUNT):  
 new\_resident = Resident(  
 passport\_data=str(fake.random\_int(1000\_000000, 9999\_999999)),  
 surname=fake.last\_name(),  
 name=fake.first\_name(),  
 patronymics=fake.first\_name() if random.random() > 0.5 else None,  
 email=fake.email(),  
 phone=f"{random.choice(prefix\_phone)}{random.randint(100, 999)}{random.randint(1000, 9999)}",  
 contracts=random.sample(contracts, fake.random\_int(0, 2))  
 )  
  
 session.add(new\_resident)  
 session.commit()  
 residents.append(new\_resident)  
  
 energy\_sources = [  
 "Ремонт",  
 "Интернет",  
 "Вода",  
 "Электричество",  
 "Отопление"  
 ]  
  
 print("Generating payments...")  
 for i in range(0, MAX\_COUNT):  
 new\_payment = Payment(  
 id=str(fake.random\_int(1000000000000, 10000000000000)),  
 until\_date=fake.date\_between(  
 start\_date=(date.today() - timedelta(days=300)),  
 end\_date=(date.today() - timedelta(days=200))),  
 paid\_date=fake.date\_between(  
 start\_date=(date.today() - timedelta(days=400)),  
 end\_date=(date.today() - timedelta(days=100))) if random.random() > 0.5 else None,  
 contract\_id=random.choice(contracts).id,  
 energy\_source=random.choice(energy\_sources),  
 payment=fake.random\_int(100, 10001)  
 )  
  
 session.add(new\_payment)  
 session.commit()  
 payments.append(new\_payment)  
  
 print("Generating tasks...")  
 for i in range(0, MAX\_COUNT):  
 new\_task = Task(  
 payment=fake.random\_int(100\_000, 10\_000\_000),  
 until\_date=fake.date\_between(  
 start\_date=(date.today() - timedelta(days=300)),  
 end\_date=(date.today() - timedelta(days=200))),  
 completed\_date=fake.date\_between(  
 start\_date=(date.today() - timedelta(days=400)),  
 end\_date=(date.today() - timedelta(days=100))) if random.random() > 0.5 else None,  
 home\_address=random.choice(homes).address  
 )  
  
 session.add(new\_task)  
 session.commit()  
 tasks.append(new\_task)  
  
 print("Generating tasks...")  
 for i in range(0, MAX\_COUNT):  
 new\_worker = Worker(  
 inn=str(fake.random\_int(100\_000\_000\_000, 1\_000\_000\_000\_000)),  
 email=fake.email() if random.random() < 0.5 else None,  
 phone=f"{random.choice(prefix\_phone)}{random.randint(100, 999)}{random.randint(1000, 9999)}" if random.random() < 0.5 else None,  
 tasks=random.sample(tasks, fake.random\_int(0, 10))  
 )  
  
 session.add(new\_worker)  
 session.commit()  
 workers.append(new\_worker)  
  
 print("Done!")

run\_fake.ps1

echo "> Read .env file"  
  
# Default location of venv file  
set-content env:\VENV\_DIR .venv  
  
get-content .env | foreach {  
 $name, $value = $\_.split('=')  
 set-content env:\$name $value  
}  
  
if (!(Test-Path "$env:VENV\_DIR/Scripts/python.exe")){  
 echo "> Venv is not found, let's install it. It'd take some time, grab some coffee!"  
 if (Test-Path "$env:VENV\_DIR") {  
 rm -r "$env:VENV\_DIR"  
 }  
 mkdir "$env:VENV\_DIR"  
 python -m venv "$env:VENV\_DIR"  
}  
  
echo "> Installing requirments"  
powershell "$env:VENV\_DIR/Scripts/pip.exe install -r ./requirments.txt"  
  
powershell "$env:VENV\_DIR/Scripts/alembic-autogen-check"  
if (!($?)) {  
 echo "> Database is not up-to-date"  
 echo "> Autogen new revision"  
 powershell "$env:VENV\_DIR/Scripts/alembic revision --autogenerate"  
 echo "> Upgrade to latest alembic version"  
 powershell "$env:VENV\_DIR/Scripts/alembic upgrade head"  
} else {  
 echo "> Database is up-to-date"  
}  
  
echo "> Start application"  
powershell "$env:VENV\_DIR/Scripts/python.exe fake.py"

До оптимизаций:

EXPLAIN ANALYZE of non payers

Sort (cost=171.20..172.58 rows=550 width=68) (actual time=1.112..1.125 rows=481 loops=1)

Sort Key: (sum(payment.payment))

Sort Method: quicksort Memory: 64kB

-> HashAggregate (cost=139.29..146.17 rows=550 width=68) (actual time=0.892..1.028 rows=481 loops=1)

Group Key: resident.passport\_data, payment.energy\_source

Batches: 1 Memory Usage: 105kB

-> Hash Join (cost=100.54..135.17 rows=550 width=52) (actual time=0.536..0.749 rows=522 loops=1)

Hash Cond: ((resident.passport\_data)::text = (residents\_contracts.resident\_passport\_data)::text)

-> Seq Scan on resident (cost=0.00..25.00 rows=1100 width=31) (actual time=0.003..0.109 rows=1100 loops=1)

-> Hash (cost=93.67..93.67 rows=550 width=32) (actual time=0.528..0.528 rows=522 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 42kB

-> Hash Join (cost=67.14..93.67 rows=550 width=32) (actual time=0.335..0.471 rows=522 loops=1)

Hash Cond: (residents\_contracts.contract\_id = contract.id)

-> Seq Scan on residents\_contracts (cost=0.00..16.93 rows=1093 width=15) (actual time=0.001..0.036 rows=1093 loops=1)

-> Hash (cost=60.22..60.22 rows=554 width=29) (actual time=0.332..0.332 rows=554 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 43kB

-> Hash Join (cost=36.75..60.22 rows=554 width=29) (actual time=0.130..0.265 rows=554 loops=1)

Hash Cond: (payment.contract\_id = contract.id)

-> Seq Scan on payment (cost=0.00..22.00 rows=554 width=25) (actual time=0.002..0.075 rows=554 loops=1)

Filter: (paid\_date IS NULL)

Rows Removed by Filter: 546

-> Hash (cost=23.00..23.00 rows=1100 width=4) (actual time=0.126..0.126 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 55kB

-> Seq Scan on contract (cost=0.00..23.00 rows=1100 width=4) (actual time=0.001..0.055 rows=1100 loops=1)

Planning Time: 0.259 ms

Execution Time: 1.164 ms

EXPLAIN ANALYZE of workers rating

Sort (cost=488.26..490.74 rows=992 width=53) (actual time=3.593..3.620 rows=992 loops=1)

Sort Key: (COALESCE(anon\_1.completed, '0'::bigint)) DESC

Sort Method: quicksort Memory: 81kB

-> Hash Left Join (cost=403.45..438.89 rows=992 width=53) (actual time=3.023..3.492 rows=992 loops=1)

Hash Cond: ((worker.inn)::text = (anon\_1.worker\_inn)::text)

-> Hash Join (cost=212.71..235.61 rows=992 width=21) (actual time=1.739..1.916 rows=992 loops=1)

Hash Cond: ((worker.inn)::text = (anon\_2.worker\_inn)::text)

-> Seq Scan on worker (cost=0.00..20.00 rows=1100 width=13) (actual time=0.002..0.041 rows=1100 loops=1)

-> Hash (cost=200.31..200.31 rows=992 width=21) (actual time=1.736..1.737 rows=992 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 60kB

-> Subquery Scan on anon\_2 (cost=180.47..200.31 rows=992 width=21) (actual time=1.523..1.654 rows=992 loops=1)

-> HashAggregate (cost=180.47..190.39 rows=992 width=21) (actual time=1.523..1.601 rows=992 loops=1)

Group Key: workers\_tasks.worker\_inn

Batches: 1 Memory Usage: 129kB

-> Hash Join (cost=43.25..152.05 rows=5683 width=13) (actual time=0.152..0.902 rows=5683 loops=1)

Hash Cond: (workers\_tasks.task\_id = task.id)

-> Seq Scan on workers\_tasks (cost=0.00..93.83 rows=5683 width=17) (actual time=0.001..0.192 rows=5683 loops=1)

-> Hash (cost=29.50..29.50 rows=1100 width=4) (actual time=0.149..0.149 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 55kB

-> Seq Scan on task (cost=0.00..29.50 rows=1100 width=4) (actual time=0.002..0.090 rows=1100 loops=1)

Filter: ((until\_date > '2004-01-01'::date) AND (until\_date < '2040-01-01'::date))

-> Hash (cost=178.35..178.35 rows=992 width=21) (actual time=1.279..1.280 rows=885 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 54kB

-> Subquery Scan on anon\_1 (cost=158.51..178.35 rows=992 width=21) (actual time=1.073..1.198 rows=885 loops=1)

-> HashAggregate (cost=158.51..168.43 rows=992 width=21) (actual time=1.073..1.144 rows=885 loops=1)

Group Key: workers\_tasks\_1.worker\_inn

Batches: 1 Memory Usage: 129kB

-> Hash Join (cost=36.09..144.89 rows=2723 width=13) (actual time=0.100..0.719 rows=2704 loops=1)

Hash Cond: (workers\_tasks\_1.task\_id = task\_1.id)

-> Seq Scan on workers\_tasks workers\_tasks\_1 (cost=0.00..93.83 rows=5683 width=17) (actual time=0.001..0.183 rows=5683 loops=1)

-> Hash (cost=29.50..29.50 rows=527 width=4) (actual time=0.098..0.098 rows=527 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 27kB

-> Seq Scan on task task\_1 (cost=0.00..29.50 rows=527 width=4) (actual time=0.001..0.069 rows=527 loops=1)

Filter: ((completed\_date IS NOT NULL) AND (until\_date > '2004-01-01'::date) AND (until\_date < '2040-01-01'::date))

Rows Removed by Filter: 573

Planning Time: 0.181 ms

Execution Time: 3.667 ms

EXPLAIN ANALYZE of house profit

Sort (cost=228.87..231.62 rows=1100 width=52) (actual time=1.546..1.576 rows=1100 loops=1)

Sort Key: ((COALESCE(anon\_1.plus, '0'::bigint) - COALESCE(anon\_2.minus, '0'::bigint)))

Sort Method: quicksort Memory: 134kB

-> Hash Left Join (cost=141.76..173.31 rows=1100 width=52) (actual time=1.067..1.417 rows=1100 loops=1)

Hash Cond: ((home.address)::text = (anon\_2.home)::text)

-> Hash Left Join (cost=89.93..115.83 rows=1100 width=52) (actual time=0.675..0.862 rows=1100 loops=1)

Hash Cond: ((home.address)::text = (anon\_1.home)::text)

-> Seq Scan on home (cost=0.00..23.00 rows=1100 width=44) (actual time=0.002..0.041 rows=1100 loops=1)

-> Hash (cost=81.17..81.17 rows=701 width=52) (actual time=0.672..0.673 rows=519 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 51kB

-> Subquery Scan on anon\_1 (cost=67.15..81.17 rows=701 width=52) (actual time=0.545..0.619 rows=519 loops=1)

-> HashAggregate (cost=67.15..74.16 rows=701 width=52) (actual time=0.545..0.589 rows=519 loops=1)

Group Key: contract.home\_address

Batches: 1 Memory Usage: 105kB

-> Hash Left Join (cost=36.75..61.65 rows=1100 width=48) (actual time=0.184..0.385 rows=1100 loops=1)

Hash Cond: (payment.contract\_id = contract.id)

-> Seq Scan on payment (cost=0.00..22.00 rows=1100 width=8) (actual time=0.002..0.084 rows=1100 loops=1)

-> Hash (cost=23.00..23.00 rows=1100 width=48) (actual time=0.180..0.180 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 104kB

-> Seq Scan on contract (cost=0.00..23.00 rows=1100 width=48) (actual time=0.002..0.072 rows=1100 loops=1)

-> Hash (cost=43.24..43.24 rows=687 width=52) (actual time=0.389..0.390 rows=687 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 65kB

-> Subquery Scan on anon\_2 (cost=29.50..43.24 rows=687 width=52) (actual time=0.230..0.322 rows=687 loops=1)

-> HashAggregate (cost=29.50..36.37 rows=687 width=52) (actual time=0.229..0.283 rows=687 loops=1)

Group Key: task.home\_address

Batches: 1 Memory Usage: 169kB

-> Seq Scan on task (cost=0.00..24.00 rows=1100 width=48) (actual time=0.001..0.041 rows=1100 loops=1)

Planning Time: 0.129 ms

Execution Time: 1.626 ms

1. Рассмотрим запрос на получение прибыли домов. Можно попробовать оптимизировать запрос путём введения индексов на home\_address:

EXPLAIN ANALYZE of house profit

Sort (cost=228.87..231.62 rows=1100 width=52) (actual time=1.667..1.698 rows=1100 loops=1)

Sort Key: ((COALESCE(anon\_1.plus, '0'::bigint) - COALESCE(anon\_2.minus, '0'::bigint)))

Sort Method: quicksort Memory: 134kB

-> Hash Left Join (cost=141.76..173.31 rows=1100 width=52) (actual time=1.079..1.519 rows=1100 loops=1)

Hash Cond: ((home.address)::text = (anon\_2.home)::text)

-> Hash Left Join (cost=89.93..115.83 rows=1100 width=52) (actual time=0.674..0.876 rows=1100 loops=1)

Hash Cond: ((home.address)::text = (anon\_1.home)::text)

-> Seq Scan on home (cost=0.00..23.00 rows=1100 width=44) (actual time=0.002..0.043 rows=1100 loops=1)

-> Hash (cost=81.17..81.17 rows=701 width=52) (actual time=0.670..0.671 rows=519 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 51kB

-> Subquery Scan on anon\_1 (cost=67.15..81.17 rows=701 width=52) (actual time=0.544..0.619 rows=519 loops=1)

-> HashAggregate (cost=67.15..74.16 rows=701 width=52) (actual time=0.544..0.589 rows=519 loops=1)

Group Key: contract.home\_address

Batches: 1 Memory Usage: 105kB

-> Hash Left Join (cost=36.75..61.65 rows=1100 width=48) (actual time=0.180..0.381 rows=1100 loops=1)

Hash Cond: (payment.contract\_id = contract.id)

-> Seq Scan on payment (cost=0.00..22.00 rows=1100 width=8) (actual time=0.002..0.082 rows=1100 loops=1)

-> Hash (cost=23.00..23.00 rows=1100 width=48) (actual time=0.176..0.177 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 104kB

-> Seq Scan on contract (cost=0.00..23.00 rows=1100 width=48) (actual time=0.001..0.071 rows=1100 loops=1)

-> Hash (cost=43.24..43.24 rows=687 width=52) (actual time=0.402..0.403 rows=687 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 65kB

-> Subquery Scan on anon\_2 (cost=29.50..43.24 rows=687 width=52) (actual time=0.231..0.324 rows=687 loops=1)

-> HashAggregate (cost=29.50..36.37 rows=687 width=52) (actual time=0.230..0.284 rows=687 loops=1)

Group Key: task.home\_address

Batches: 1 Memory Usage: 169kB

-> Seq Scan on task (cost=0.00..24.00 rows=1100 width=48) (actual time=0.001..0.040 rows=1100 loops=1)

Planning Time: 0.126 ms

Execution Time: 1.761 ms

Добавленный индекс игнорируется планировщиком и только будет занимать время работы. Оптимизировать данный запрос не получится. Индекс не помог при оптимизации.

1. Попробуем оптимизировать запрос на получение рейтинга рабочих.

Основной проблемой этого запроса является то, что в нём используются два практически одинаковых подзапроса для вычисления completed и total.

completed = (  
 select(  
 WorkerTask.worker\_inn,  
 func.count().label("completed")  
 )  
 .select\_from(WorkerTask)  
 .join(Task)  
 .where(  
 and\_(  
 Task.completed\_date.isnot(None),  
 Task.until\_date > begin\_date,  
 Task.until\_date < end\_date  
 )  
 )  
 .group\_by(WorkerTask.worker\_inn)  
).subquery()  
  
total = (  
 select(  
 WorkerTask.worker\_inn,  
 func.count().label("total")  
 )  
 .select\_from(WorkerTask)  
 .join(Task)  
 .where(  
 and\_(  
 Task.until\_date > begin\_date,  
 Task.until\_date < end\_date  
 )  
 )  
 .group\_by(WorkerTask.worker\_inn)  
).subquery()

Возможно, получится обойтись одним подзапросом:

completed\_total = (  
 select(  
 WorkerTask.worker\_inn,  
 func.count().label("total"),  
 func.count(Task.completed\_date).label("completed")  
 )  
 .select\_from(WorkerTask)  
 .join(Task)  
 .where(  
 Task.until\_date.between(begin\_date, end\_date)  
 )  
 .group\_by(WorkerTask.worker\_inn)  
).subquery()  
  
main\_query = (select(  
 Worker.inn.label("worker\_inn"),  
 func.coalesce(completed\_total.c.completed, 0).label("completed"),  
 (1.0 \* func.coalesce(completed\_total.c.completed, 0) / func.coalesce(completed\_total.c.total, 1)).label("rating")  
 )  
 .select\_from(Worker)  
 .join(completed\_total, completed\_total.c.worker\_inn == Worker.inn, isouter=True)  
 .order\_by(desc("completed")))

EXPLAIN ANALYZE of workers rating

Sort (cost=316.38..319.13 rows=1100 width=53) (actual time=2.426..2.454 rows=1100 loops=1)

Sort Key: (COALESCE(anon\_1.completed, '0'::bigint)) DESC

Sort Method: quicksort Memory: 111kB

-> Hash Left Join (cost=226.92..260.81 rows=1100 width=53) (actual time=1.944..2.323 rows=1100 loops=1)

Hash Cond: ((worker.inn)::text = (anon\_1.worker\_inn)::text)

-> Seq Scan on worker (cost=0.00..20.00 rows=1100 width=13) (actual time=0.002..0.039 rows=1100 loops=1)

-> Hash (cost=214.52..214.52 rows=992 width=29) (actual time=1.938..1.939 rows=992 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 68kB

-> Subquery Scan on anon\_1 (cost=194.68..214.52 rows=992 width=29) (actual time=1.706..1.852 rows=992 loops=1)

-> HashAggregate (cost=194.68..204.60 rows=992 width=29) (actual time=1.706..1.795 rows=992 loops=1)

Group Key: workers\_tasks.worker\_inn

Batches: 1 Memory Usage: 193kB

-> Hash Join (cost=43.25..152.05 rows=5683 width=17) (actual time=0.171..1.008 rows=5683 loops=1)

Hash Cond: (workers\_tasks.task\_id = task.id)

-> Seq Scan on workers\_tasks (cost=0.00..93.83 rows=5683 width=17) (actual time=0.001..0.190 rows=5683 loops=1)

-> Hash (cost=29.50..29.50 rows=1100 width=8) (actual time=0.169..0.169 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 57kB

-> Seq Scan on task (cost=0.00..29.50 rows=1100 width=8) (actual time=0.002..0.097 rows=1100 loops=1)

Filter: ((until\_date > '2004-01-01'::date) AND (until\_date < '2040-01-01'::date))

Planning Time: 0.111 ms

Execution Time: 2.497 ms

Время подготовки и выполнения снизилось с

Planning Time: 0.181 ms

Execution Time: 3.667 ms

до

Planning Time: 0.111 ms

Execution Time: 2.497 ms

(практически в 1.5 раза)

А значит оптимизация прошла успешно. Можно также попробовать добавить индекс на поле until\_date, чтобы сравнение с переданной датой выполнялось по индексу:

EXPLAIN ANALYZE of workers rating

Sort (cost=316.38..319.13 rows=1100 width=53) (actual time=2.376..2.404 rows=1100 loops=1)

Sort Key: (COALESCE(anon\_1.completed, '0'::bigint)) DESC

Sort Method: quicksort Memory: 111kB

-> Hash Left Join (cost=226.92..260.81 rows=1100 width=53) (actual time=1.894..2.272 rows=1100 loops=1)

Hash Cond: ((worker.inn)::text = (anon\_1.worker\_inn)::text)

-> Seq Scan on worker (cost=0.00..20.00 rows=1100 width=13) (actual time=0.002..0.038 rows=1100 loops=1)

-> Hash (cost=214.52..214.52 rows=992 width=29) (actual time=1.889..1.890 rows=992 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 68kB

-> Subquery Scan on anon\_1 (cost=194.68..214.52 rows=992 width=29) (actual time=1.658..1.803 rows=992 loops=1)

-> HashAggregate (cost=194.68..204.60 rows=992 width=29) (actual time=1.657..1.746 rows=992 loops=1)

Group Key: workers\_tasks.worker\_inn

Batches: 1 Memory Usage: 193kB

-> Hash Join (cost=43.25..152.05 rows=5683 width=17) (actual time=0.181..1.010 rows=5683 loops=1)

Hash Cond: (workers\_tasks.task\_id = task.id)

-> Seq Scan on workers\_tasks (cost=0.00..93.83 rows=5683 width=17) (actual time=0.001..0.189 rows=5683 loops=1)

-> Hash (cost=29.50..29.50 rows=1100 width=8) (actual time=0.178..0.178 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 57kB

-> Seq Scan on task (cost=0.00..29.50 rows=1100 width=8) (actual time=0.002..0.097 rows=1100 loops=1)

Filter: ((until\_date >= '2004-01-01'::date) AND (until\_date <= '2040-01-01'::date))

Planning Time: 0.123 ms

Execution Time: 2.448 ms

Однако индекс был проигнорирован планировщиком, время незначительно уменьшилось:

Planning Time: 0.123 ms

Execution Time: 2.448 ms

Оптимизация с применением индекса не принесла пользы.

1. Попробуем оптимизировать запрос на получение неплательщиков

Попробуем добавить индекс на Payment.paid\_date, чтобы фильтр по запросу IS NULL выполнялся по индексированной таблице:

EXPLAIN ANALYZE of non payers

Sort (cost=171.20..172.58 rows=550 width=68) (actual time=1.099..1.112 rows=481 loops=1)

Sort Key: (sum(payment.payment))

Sort Method: quicksort Memory: 64kB

-> HashAggregate (cost=139.29..146.17 rows=550 width=68) (actual time=0.880..1.016 rows=481 loops=1)

Group Key: resident.passport\_data, payment.energy\_source

Batches: 1 Memory Usage: 105kB

-> Hash Join (cost=100.54..135.17 rows=550 width=52) (actual time=0.539..0.738 rows=522 loops=1)

Hash Cond: ((resident.passport\_data)::text = (residents\_contracts.resident\_passport\_data)::text)

-> Seq Scan on resident (cost=0.00..25.00 rows=1100 width=31) (actual time=0.003..0.097 rows=1100 loops=1)

-> Hash (cost=93.67..93.67 rows=550 width=32) (actual time=0.530..0.530 rows=522 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 42kB

-> Hash Join (cost=67.14..93.67 rows=550 width=32) (actual time=0.335..0.471 rows=522 loops=1)

Hash Cond: (residents\_contracts.contract\_id = contract.id)

-> Seq Scan on residents\_contracts (cost=0.00..16.93 rows=1093 width=15) (actual time=0.001..0.036 rows=1093 loops=1)

-> Hash (cost=60.22..60.22 rows=554 width=29) (actual time=0.332..0.332 rows=554 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 43kB

-> Hash Join (cost=36.75..60.22 rows=554 width=29) (actual time=0.130..0.274 rows=554 loops=1)

Hash Cond: (payment.contract\_id = contract.id)

-> Seq Scan on payment (cost=0.00..22.00 rows=554 width=25) (actual time=0.002..0.074 rows=554 loops=1)

Filter: (paid\_date IS NULL)

Rows Removed by Filter: 546

-> Hash (cost=23.00..23.00 rows=1100 width=4) (actual time=0.127..0.127 rows=1100 loops=1)

Buckets: 2048 Batches: 1 Memory Usage: 55kB

-> Seq Scan on contract (cost=0.00..23.00 rows=1100 width=4) (actual time=0.001..0.055 rows=1100 loops=1)

Planning Time: 0.268 ms

Execution Time: 1.152 ms

Индекс был проигнорирован, время изменилось незначительно

Planning Time: 0.268 ms

Execution Time: 1.152 ms

Оптимизация с применением индекса не принесла пользы.

Ссылка на репозиторий: <https://github.com/IAmProgrammist/database/tree/main/lab8>

**Вывод:** в ходе лабораторной работы получили навыки повышения производительности работы СУБД с помощью оптимизации sql-запросов с использованием планировщика. Индексы, хоть и являются очень полезными, однако часто игнорируются планировщиками, основным инструментом при оптимизации запросов являются оптимальные запросы: использование как можно меньшего кол-ва вложенных запросов и джойнов, знание работы функций.