Санкт-Петербургский государственный университет
Факультет Прикладной математики - Процессов управления

Отчет по дисциплине
“Оптимизация баз данных”

Работу выполнили
1-курс магистратуры по направлению подготовки: ВМ.5503
“Технологии баз данных”

Тарасов Никита
Кукаркин Михаил
Гао Тянь

Преподаватель
Севрюков Сергей Юрьевич

Дата: 27.06.2020

Начальные действия

Описание архитектуры базы данных

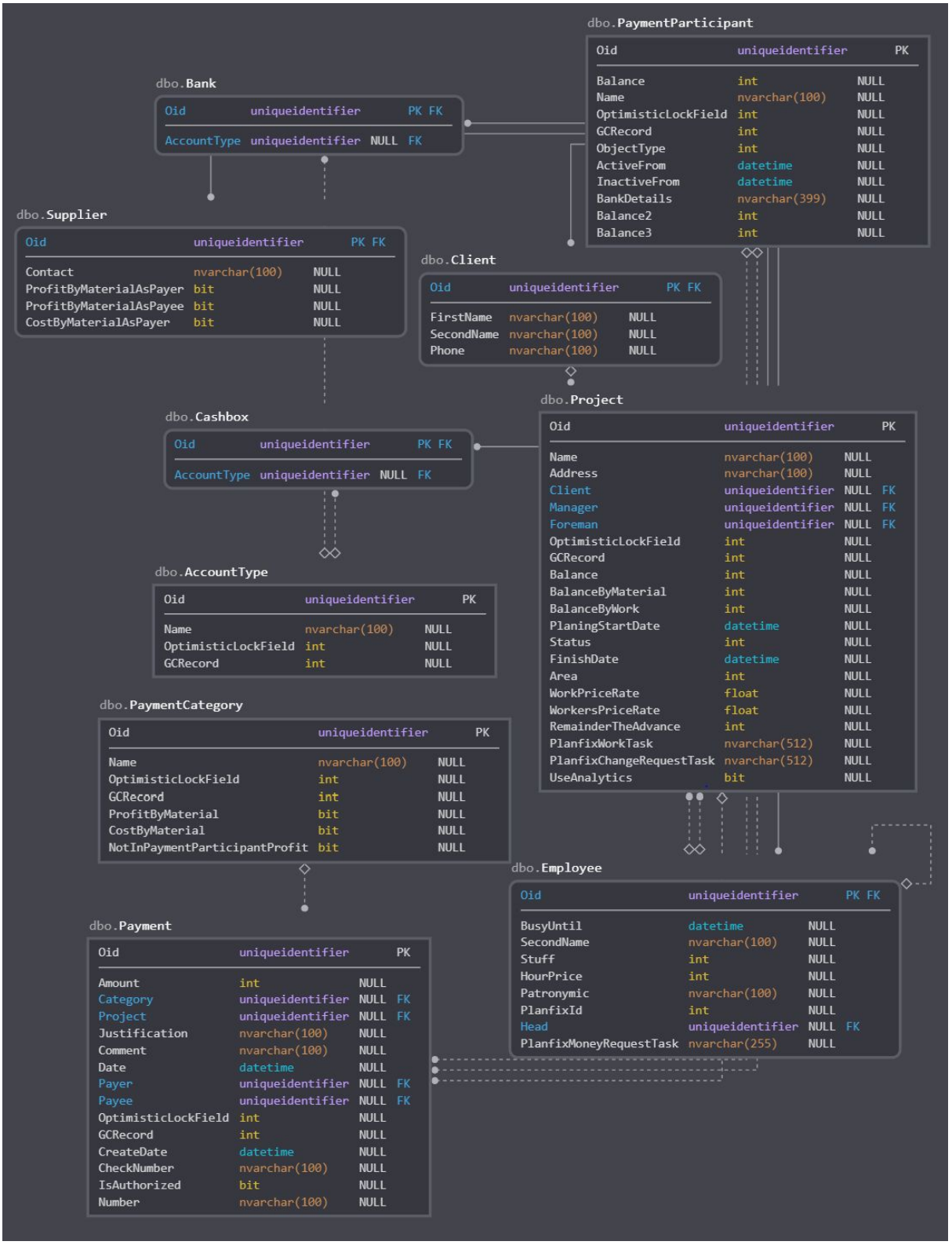

Рис 1. Схема базы данных до заполнения

Функции создаваемые sql скриптом

CalculateBalanceByMaterial	Функция расчета баланса по материалам
CalculateRemainderTheAdvance	Функция расчета остатка аванса
CalculatePaymentParticipantBalance	Функция расчета баланса участника платежа
CalculateBalanceByWork	Функция расчета баланса по работам
CalculateProjectBalance	Функция расчета баланса объекта

Триггеры создаваемые sql скриптом

PaymentParticipant_BU	Срабатывает до внесения изменений в таблицу
Project_BU	Срабатывает до внесения правок в объект
Payment_AI	Срабатывает после вставки или изменения платежа

Генерация данных для заполнения

Рассматривались различные способы заполнения данными:

- SQL запросами
- Python для генерации и последующей записи данных
- Использование специальных инструментов, автоматизирующих процесс

Пример - Redgate SQL Data Generator дает простой и удобный интерфейс заполнения нужными данными, при этом данные подбираются автоматически по типам данных и заголовкам, тем не менее необходимо ручное исправление диапазонов для некоторых столбцов и процентного соотношения NULL элементов (очевидно что поле GCRecord в большинстве случаев будет иметь значение NULL)

Есть возможность указать число создаваемых элементов (и на них нет ограничений по лицензии как у многих других проприетарных решений), что полезно в процессе тестирования нагрузки

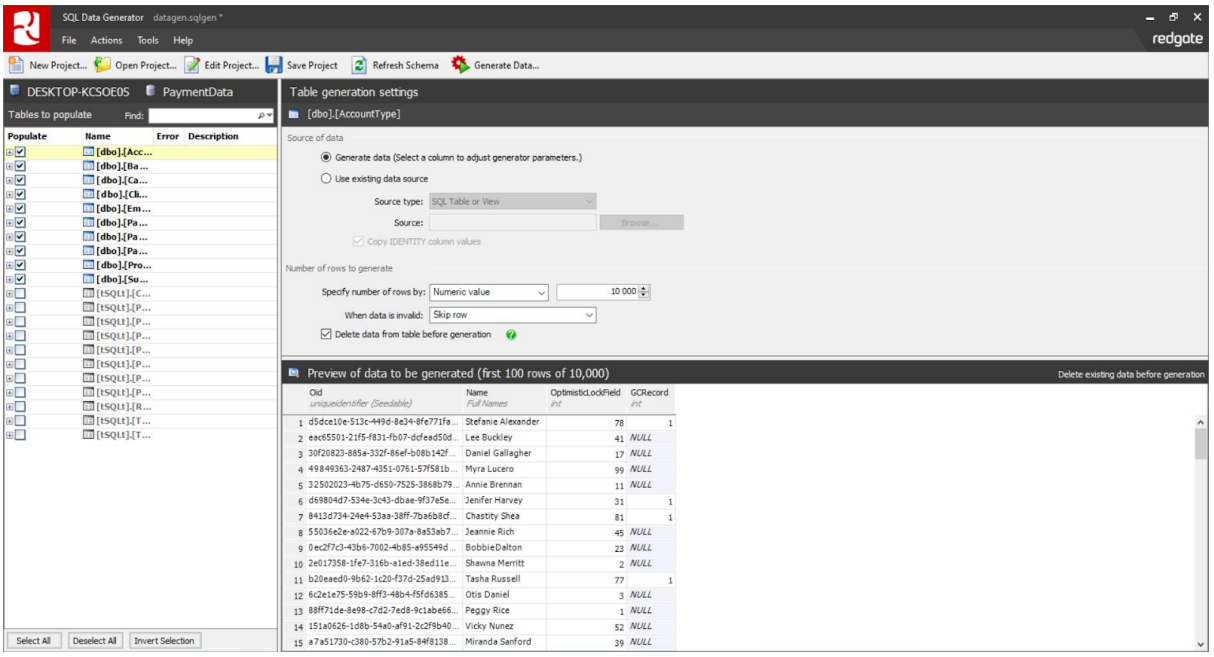

Рис 2. Пример интерфейса и итоговая версия для таблицы PaymentParticipant
Поля, заполняемые триггерами и функциями остаются пустыми.

Тест на корректность расчета

Нужна проверка 4 балансов по строкам в таблице: банка, поставщика, клиента и кассы (по транзакциям соответствующим таблице из https://github.com/sevryukov/DBOpt_Course_PaymentData/blob/master/Balance%20description.pdf)
Тест проводится путем создания транзакций (записей в таблицу Payment) и проверки соответствия значений записанных триггером в таблицу после выполнения функции, значениям рассчитанным вручную.
SQL запрос выполняющий проверку приведен в репозитории.

Задача 1

Тестирование скорости добавления и чтения транзакций с помощью SQLQueryStress

Первоначальные тесты производились на базе с 1000 записями
Тест операции чтения (в данном случае случайной записи), на чтение одной записи требуется 0.004 сек

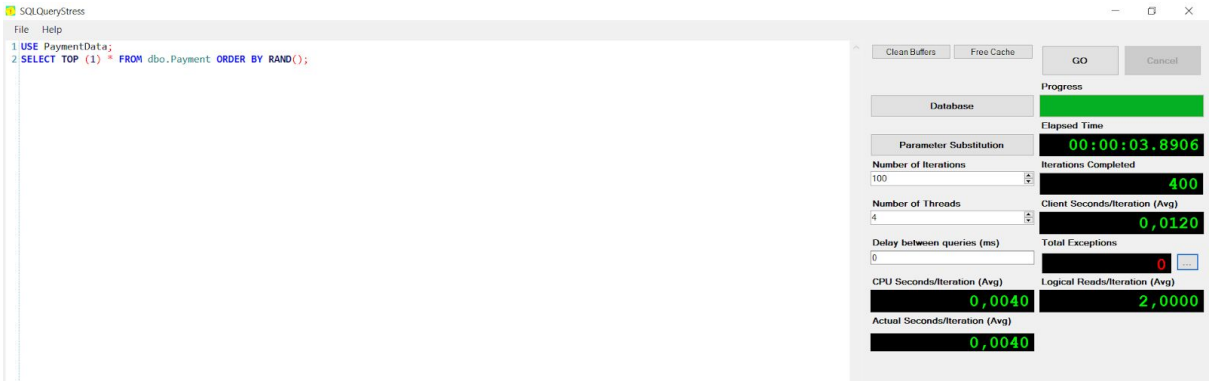

Рис 3. Тест чтения для базы из 1000 элементов

Была протестирована операция добавления новой записи 400 раз на 1 потоке (400 суммарных записей)
В результате среднее время добавления платежа составило 0.6501 сек

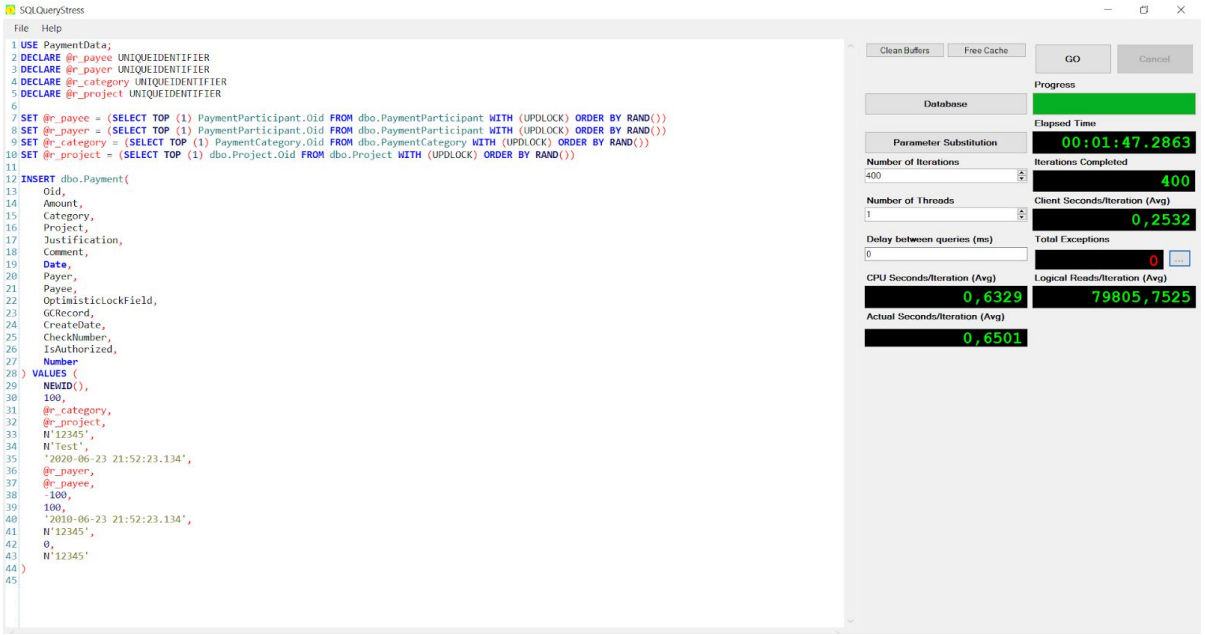

Рис 4. Тест записи для базы из 1000 элементов

Такие же тесты с базой в которой 10 000 записей показывают следующие результаты

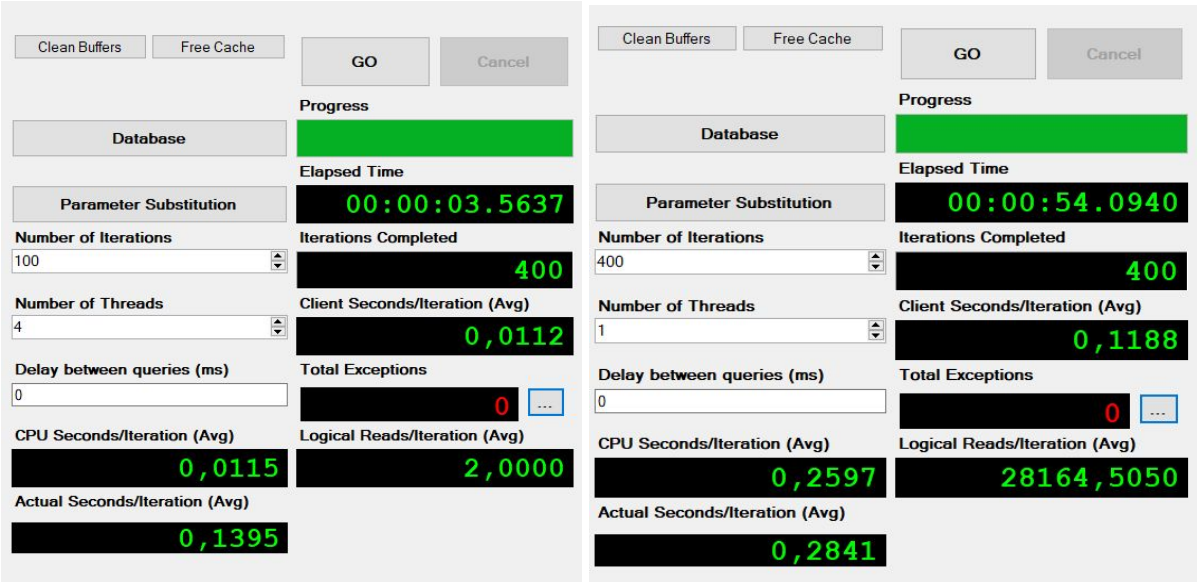

Рис 5. Слева: Тест чтения для базы из 10 00 элементов
Справа: Тест записи для базы из 10 00 элементов

Чтение заметно замедлилось, но скорость записи транзакции увеличилась???

Повышение производительности операций вставки и изменения платежей

Несмотря на то что Execution plan показывает, что при добавлении транзакции в таблицу Payment использует операцию Index Scan (NonClustered), а не Index Seek, на операцию сканирования тратится незначительное время (>1%) от общего времени выполнения запроса.

Большая часть нагрузки при этом происходит в момент выполнения функций расчета балансов (от 5% до 19%) и обновления таблиц, т.е. можно оптимизировать запрос на добавление платежа с помощью создания некластеризованных индексов для параметров, используемых в функциях.

```
USE PaymentData
-- все функции
CREATE NONCLUSTERED INDEX indexedNamePaymentCategory ON dbo.PaymentCategory (Name)

--все функции кроме CalculateProjectBalance
CREATE NONCLUSTERED INDEX indexedNameAccountType ON dbo.AccountType (Name)

-- функция CalculateBalanceByMaterial
CREATE NONCLUSTERED INDEX indexedProfitByMaterialAsPayer ON dbo.Supplier (ProfitByMaterialAsPayer)
CREATE NONCLUSTERED INDEX indexedProfitByMaterialAsPayee ON dbo.Supplier (ProfitByMaterialAsPayee)
CREATE NONCLUSTERED INDEX indexedProfitByMaterial ON dbo.PaymentCategory (ProfitByMaterial)
CREATE NONCLUSTERED INDEX indexedCostByMaterial ON dbo.PaymentCategory (CostByMaterial)

--функция CalculatePaymentParticipantBalance
CREATE NONCLUSTERED INDEX indexedNotInPaymentParticipantProfit ON dbo.PaymentCategory (NotInPaymentParticipantProfit)
```

Рис 6. Создание индексов для различных параметров функций

После создания индексов SQLQueryStress показывает следующий результат

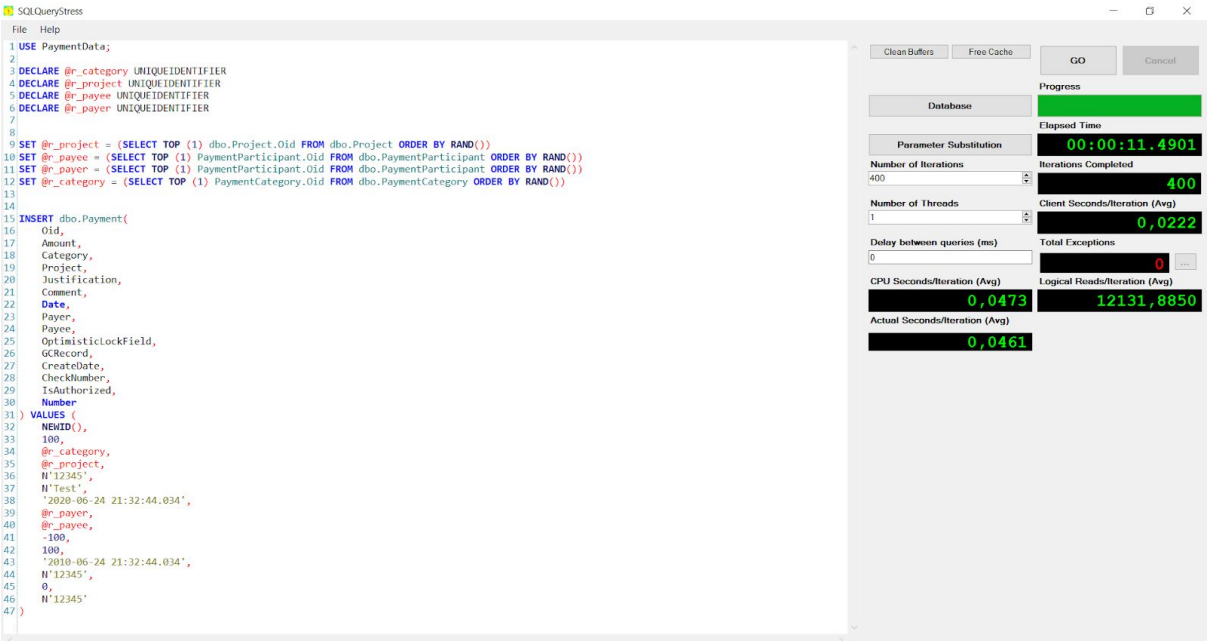

Рис 7. Результат операции вставки платежа после создания индексов

Как можно увидеть, скорость операции вставки многократно возросла, прирост скорее всего слишком большой, возможно из-за неточностей измерения, однако похожий результат сохраняется на нескольких прогонах.

Задача сделана по материалам: https://habr.com/ru/post/310328/

Задача 2

Оценка затрат на выполнения операций

Профайлинг показал, что в среднем на вычисление баланса затрачивается 3.4% ресурсов, а на обновление баланса 24.7%. SELECT запросы исходного когда в среднем выполняются за 9ms. При вставке нескольких строк за раз больше всего времени тратится на поиск по индексу и его обновление.

Сценарий оптимизации механизмов расчета

Исходя из результатов профайлинга предлагается следующий сценарий оптимизации механизмов расчета. Основная идея заключается в производстве расчетов над батчем данных, а не поштучно после каждого добавления. При заполнении батча заданным количеством данных должен срабатывать триггер для запуска расчета баланса. Однако, если данные обновляются недостаточно

часто перерасчет может быть произведен с слишком большой задержкой. Для устранения этой проблемы следует реализовать сторонний программный компонент, выполняющий расчет баланса с заданным интервалом.

Недостатки сценария

Недостатки данного сценария оптимизации механизмов расчета можно выявить следующие:

пользователь может получать неактуальные данные
расходы по вычислительным ресурсам

Заключение

- В результате выполнения работы, была создана и наполнена данными база, проверена корректность функций расчета баланса.
- В первой задаче было протестировано добавление и чтение записей в таблицу Payment, созданы индексы, повышающие производительность данных операций.
- Во второй задаче была выполнена оценка затрат, предложен сценарий оптимизации, описаны его потенциальные недостатки.