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”

Щеникова Снежана

Асташенкова Лада

Ли Цзин

Преподаватель

Севрюков Сергей Юрьевич

Дата: 27.06.2020**Начало работы**

Перед началом работы в нашей команде были распределены задачи (Таблица 1).

|  |  |
| --- | --- |
| **Участник** | **Обязанности** |
| Щеникова Снежана | Генерация данных, создание теста, Task 1 |
| Асташенкова Лада | Task 1, Task 2 |
| Li Jing | Task 2 |

Таблица 1. Распределение нагрузки на участников

**Этап 1. Выбор СУБД**

В связи с тем, что у участников нет опыта работы с Microsoft SQL Server, но есть знания в PostgreSQL, было решено перевести исходный скрипт PaymentData.sql на диалект PostgreSQL. Для этого сначала была попытка использовать существующие инструменты (встроенный в PostgreSQL, online conversion SQLines), однако не получилось сделать это корректно. В связи с этим, перевод происходил вручную. Это стало причиной возникновения проблем при тестировании в расчете баланса (Рис 1).

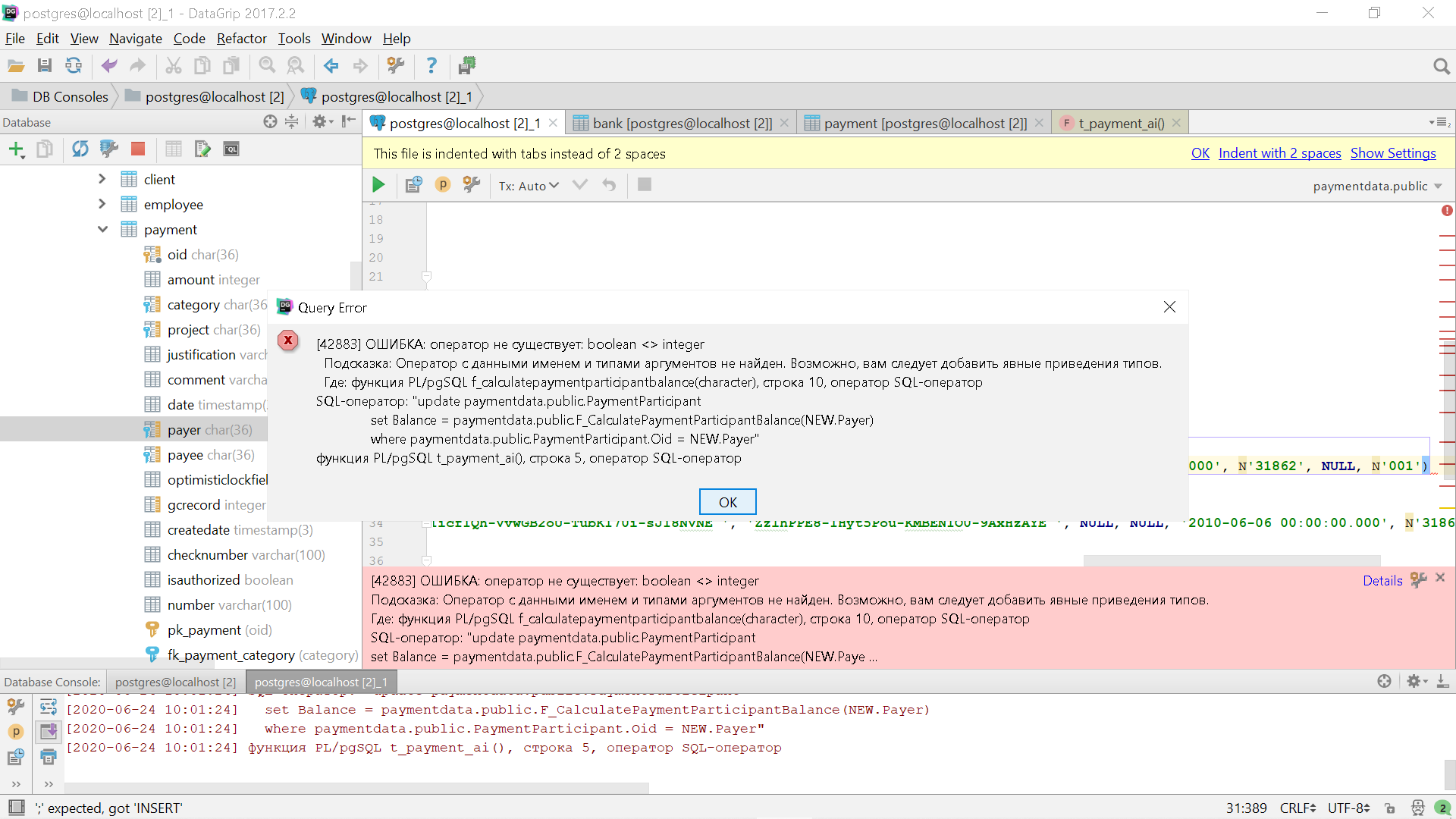


Рис.1. Ошибка при выполнении тестового скрипта

Из-за ограниченного времени работы над проектом и отсутствия знаний в pl/pgsql было принято решение осваивать Microsoft SQL Server и проводить анализ на нем, используя как встроенные инструменты, так и внешние. В ходе запуска скрипта проблем не возникло (Рис. 2).

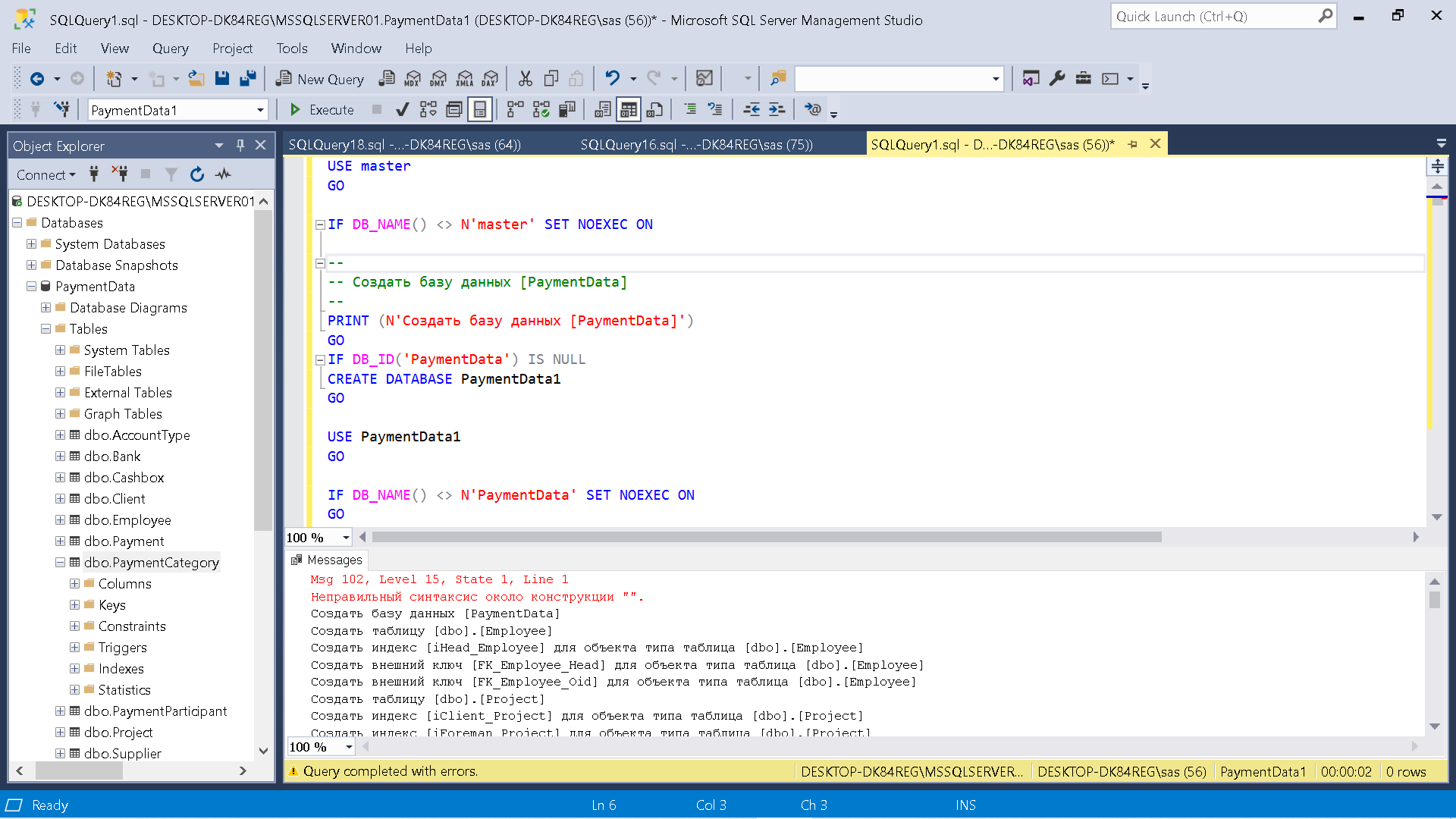


Рис. 2. Запуск скрипта

**Этап 2. Генерация данных**

Для генерации данных использовался инструмент **dbForge**, однако там были ограничения на то, что максимально можно было добавить 50 строк. В связи с этим для дальнейшего тестирования на производительность было принято решение генерации данных с помощью инструмента **Red Gate**. В ходе генерации данных с помощью dbForge возникали проблемы в понимании, как не заносить данные в таблицы AccountType и PaymentCategory. Однако получилось понять, что это сделать возможно с помощью создания запроса на выбор произвольного ключа из этих таблиц.

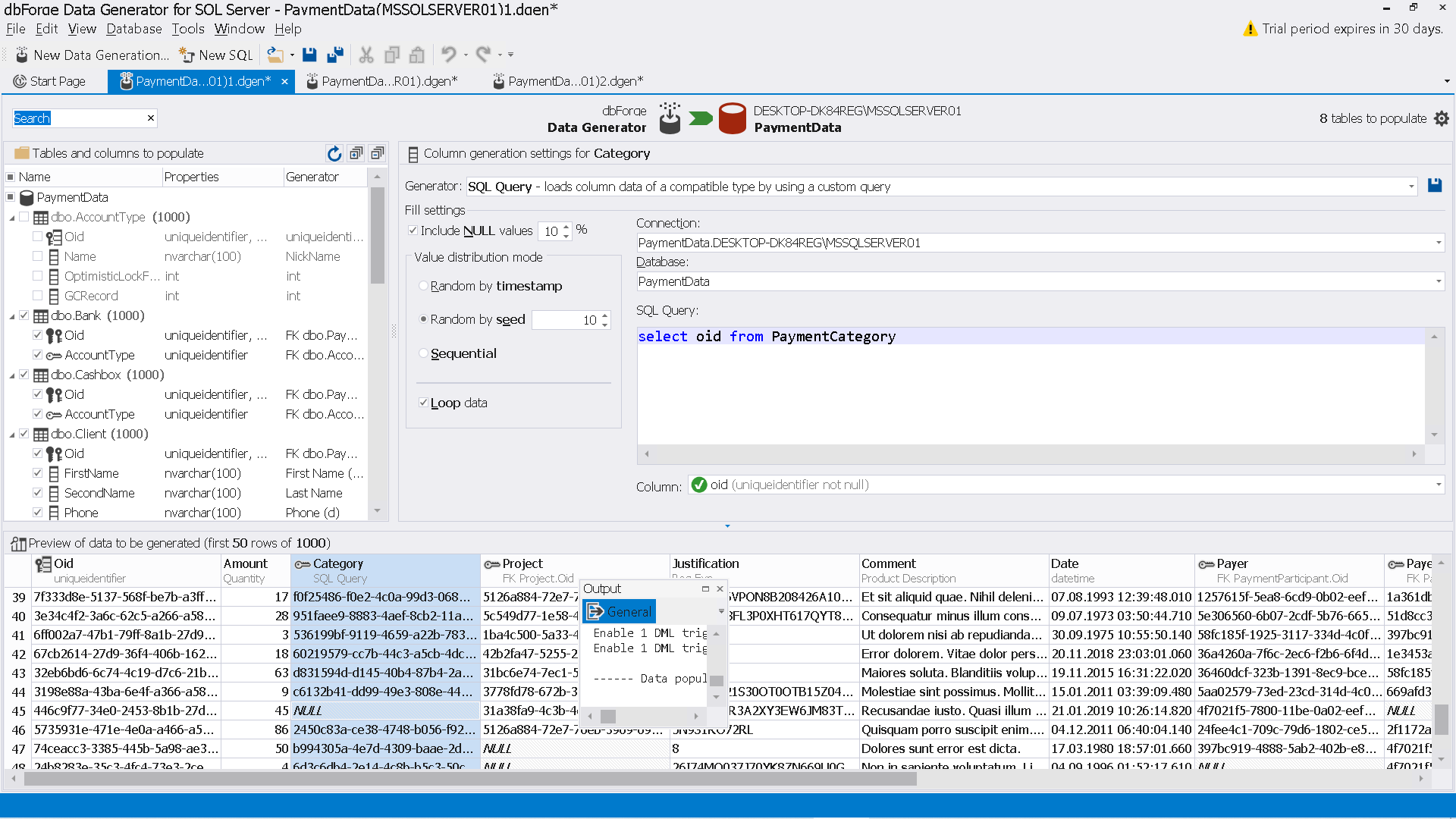


Рис. 3. Применение генератора данных dbForge

**Этап 3. Создание тестового скрипта**

Перед составлением тестового скрипта необходимо было ознакомиться с тем, что собой представляет база данных, каким образом связаны таблицы и их предназначение.

**Сущности**:

• accounttype = тип счета

• bank = безналичные счета

• cashbox = наличные счета

• client = клиент

• employee = наемный рабочий

• payment = Платежи. Кто, кому, сколько, когда, зачем (в какой категории) и в контексте какого объекта

• paymentcategory = категории платежа. В зависимости от категории, платёж участвует в том или ином балансе

• paymentparticipant = базовые данные всех участников платежей

• project = сведения об объектах работ (объектах недвижимости)

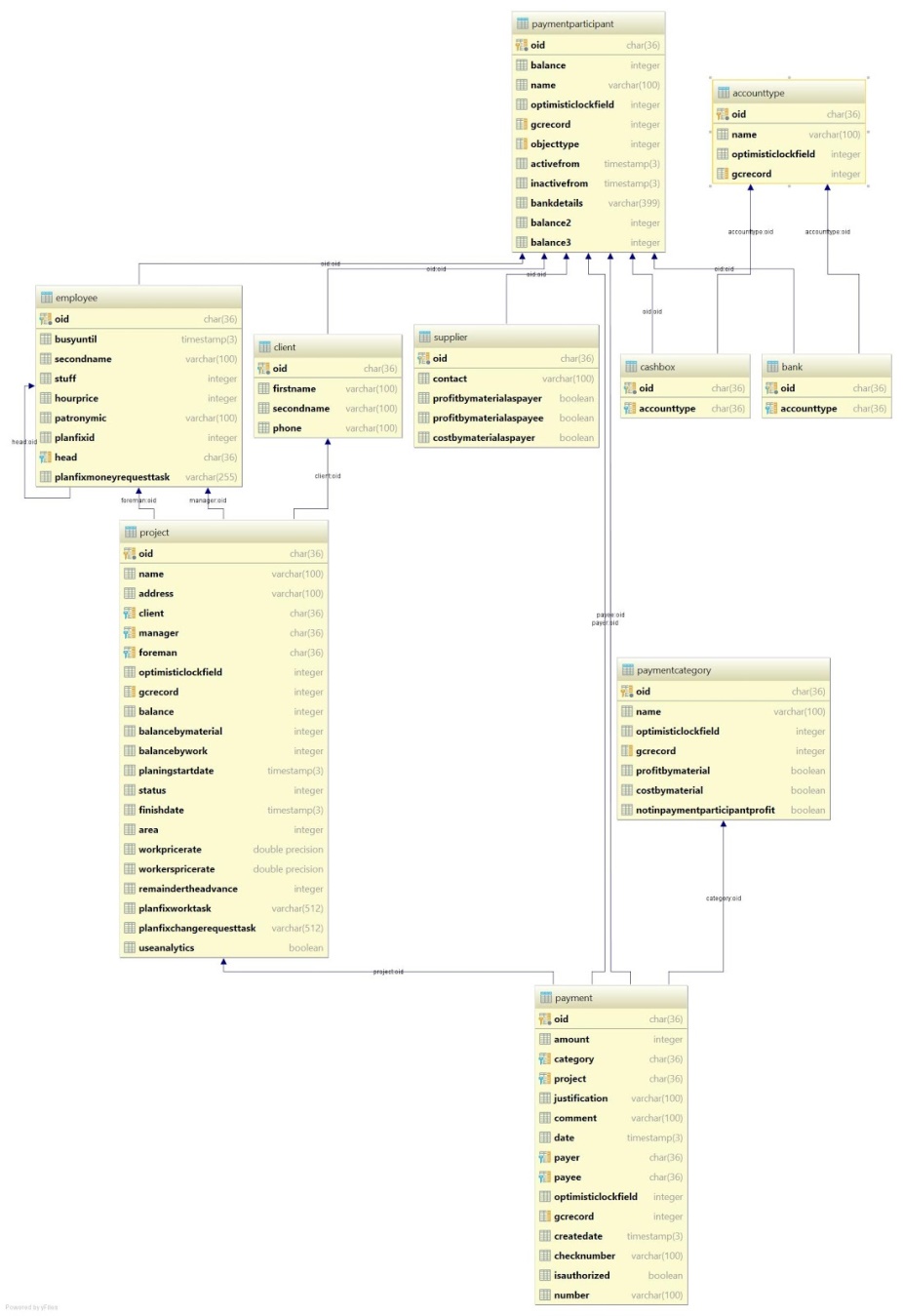
• supplier = поставщики

Схема 1. Диаграмма таблиц текущей базы данных

Хранимые процедуры и триггеры

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Где используется |
| f\_calculatebalancebymaterial | Расчет баланса по колонке «materal» | **update** paymentdata.public.Project  **BalanceByMaterial** |
| f\_calculatebalancebywork | Расчет баланса по работе | **update** paymentdata.public.Project  **BalanceByWork** |
| f\_calculatepaymentparticipantbalance | Расчет баланса при сущности paymentparticipant | **update** paymentdata.public.PaymentParticipant  **Balance** |
| f\_calculateprojectbalance | Расчет баланса по проекту | **update** paymentdata.public.Project  **Balance** |
| f\_calculateremaindertheadvance | Расчет остатка аванса | **UPDATE** paymentdata.public.Project  **RemainderTheAdvance** |

Таблица 2. Хранимые процедуры

**T\_Payment\_AI** = триггер, что запускает функцию расчета баланса после обновления для таблицы PaymentParticipant

**T\_PaymentParticipant\_BU** = триггер, что запускает функцию расчета баланса в своей таблице перед обновлением

**T\_PROJECT\_BU** = триггер, что запускает функции расчета

[Balance] = dbo.F\_CalculateProjectBalance(Project.Oid)

,[BalanceByMaterial] = dbo.F\_CalculateBalanceByMaterial(Project.Oid)

,[BalanceByWork] = dbo.F\_CalculateBalanceByWork(Project.Oid)

,[RemainderTheAdvance] = dbo.F\_CalculateRemainderTheAdvance(Project.Oid)

перед обновлением таблицы Project.

Схема расчета баланса:

1.Insert Payment: заносятся данные в таблицу

2.Вызывается триггер **t\_payment\_ai**

- Вызывается триггер **t\_paymentparticipant\_bu**;

- Обновляется баланс в PaymentParticipant для новых получателя и плательщика и двух старых (UPDATE paymentparticipant);

3.Вызывается триггер **t\_project\_bu**

- Вносятся изменения в таблицу Project

Для формирования теста было проведено ознакомление с документом по расчету баланса (Balance description.pdf). Тестовый скрипт был создан по тем данным, что приведены в таблице.

Алгоритм тестирования:

1.Заносим нулевые значения в PaymentPartition (INSERT): balance = 0

2.Создаем проведенные платежи (их у нас 4)

- после вставки вызывается расчет F\_CalculatePaymentParticipantBalance

для таблицы PaymentParticipant столбца Balance для 4х строк с разными плательщиками и получателями

1: баланс банка, баланс поставщика меняются

2: баланс поставщика, баланс клиента меняются

3: баланс кассы и клиента меняются

4: баланс кассы и банка меняются

расчет F\_CalculateBalanceByMaterial для таблицы Project

В результате теста получаются значения баланса, которые соответствуют последней строке таблицы из примера.

Таким образом, были проведены начальные действия по работе над проектом.

**Часть 1**

**Задача**: Реализовать индексы, повышающие производительность операций вставки и изменения платежей без модификации программных компонент

**Выполнила**: Щеникова Снежана

**Инструменты:** SQL Server Profiler, Database Engine Tuning Advisor, Activity Monitor, Client Statistics, SQL Query Stress, запросы.

Перед началом реализации индексов необходимо было ознакомиться с существующими индексами (Таблица 3) и тем, как происходит расчет баланса.

|  |  |
| --- | --- |
| **Сущность** | **Индексы** |
| **accounttype** | id |
| **bank** | Id,Accounttype\_id |
| **cashbox** | Id,Accounttype\_id |
| **Client** | id |
| **employee** | id |
| **payment** | 1.id  2.category\_id  3.project\_id  4.payer\_id  5.payee\_id  6.gcrecord |
| **Paymentcategory** | Id, gcrecord |
| **Paymentparticipant** | id, gcrecord, objecttype |
| **Project** | id, client\_id, manager\_id, foreman\_id, gcrecord |
| **supplier** | Id |

Таблица 3. Существующие индексы

Ради интереса было проведено тестирование на изменение производительности за счет удаления каких-либо индексов из данного списка. Но избавляться от индексов нужно с логической точки зрения, а не как попало. В связи с этим была рассмотрена статистика, которая показывает плотность индекса. Чем меньше плотность, тем лучше – это увеличивает избирательность, а, следовательно, и ценность построенного индекса. Плотность считается по формуле: .

Делается это с помощью команды:

DBCC SHOW\_STATISTICS ([<Сущность>], <Название индекса>)

**Результаты**

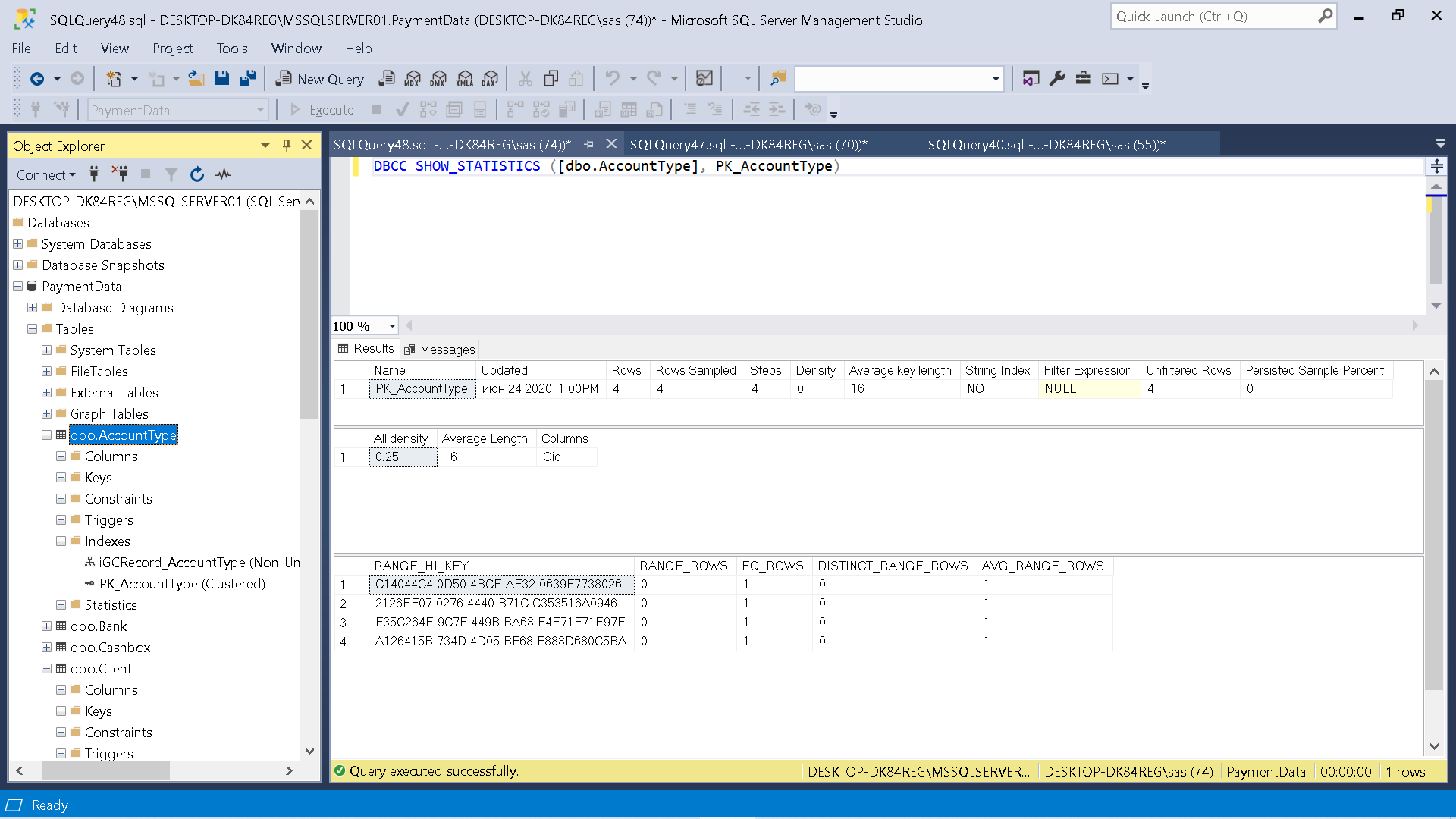


Рис. 4. Анализ индекса таблицы AccountType

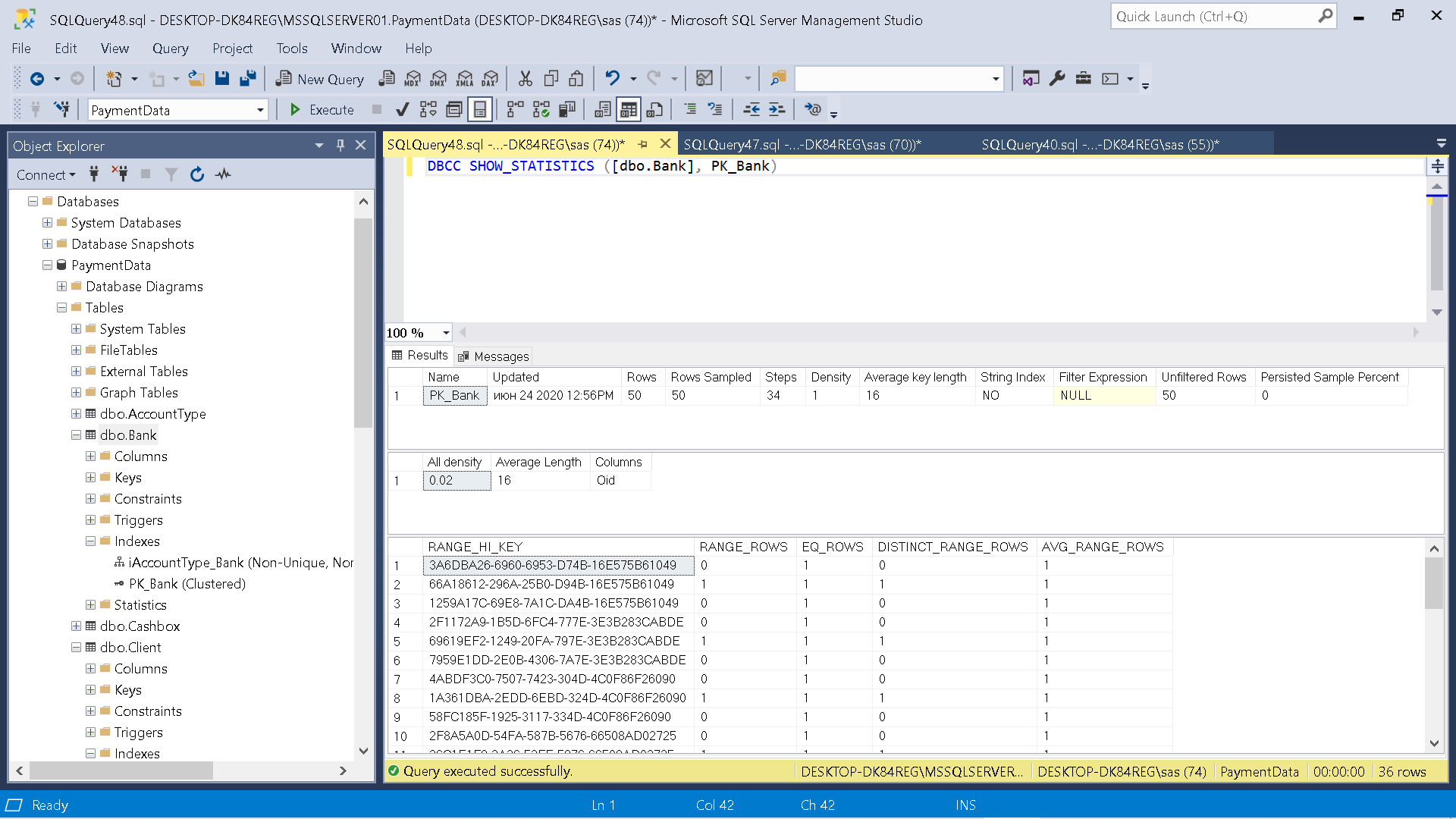


Рис. 5. Анализ индекса таблицы Bank

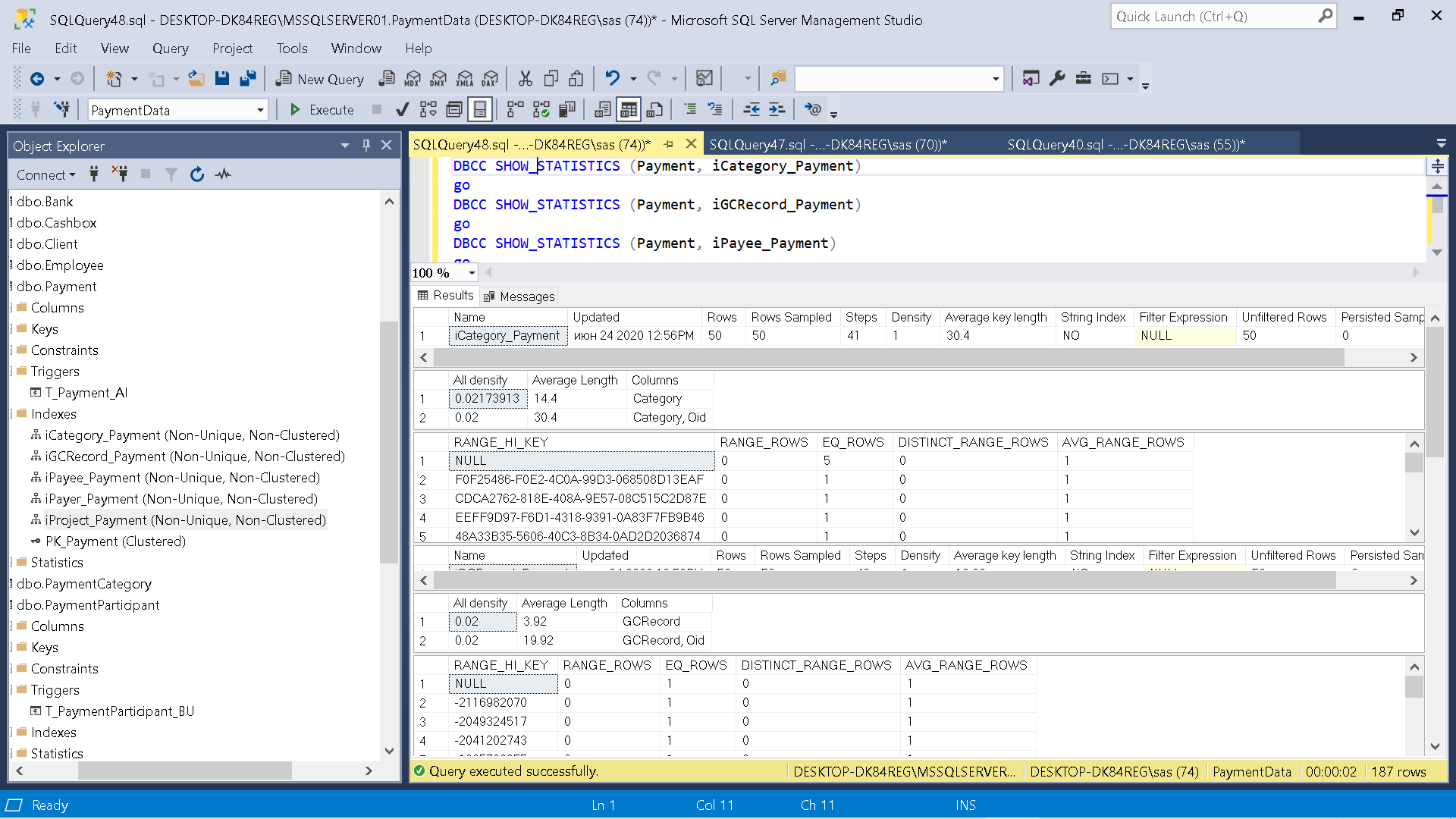


Рис. 6. Анализ индексов таблицы Payment

Такая работа была проделана для всех таблиц. В ходе анализа не было обнаружено плохих показателей, тем самым подтвердился тот факт, что существующие индексы являются полезными.

Далее была поставлена задача: определить, какие запросы к бд являются наиболее дорогими. Для этого использовался Activity Monitor, где можно увидеть запросы, скорость выполнения которых является замедленной. На Рис. 7 отображается результат анализа.

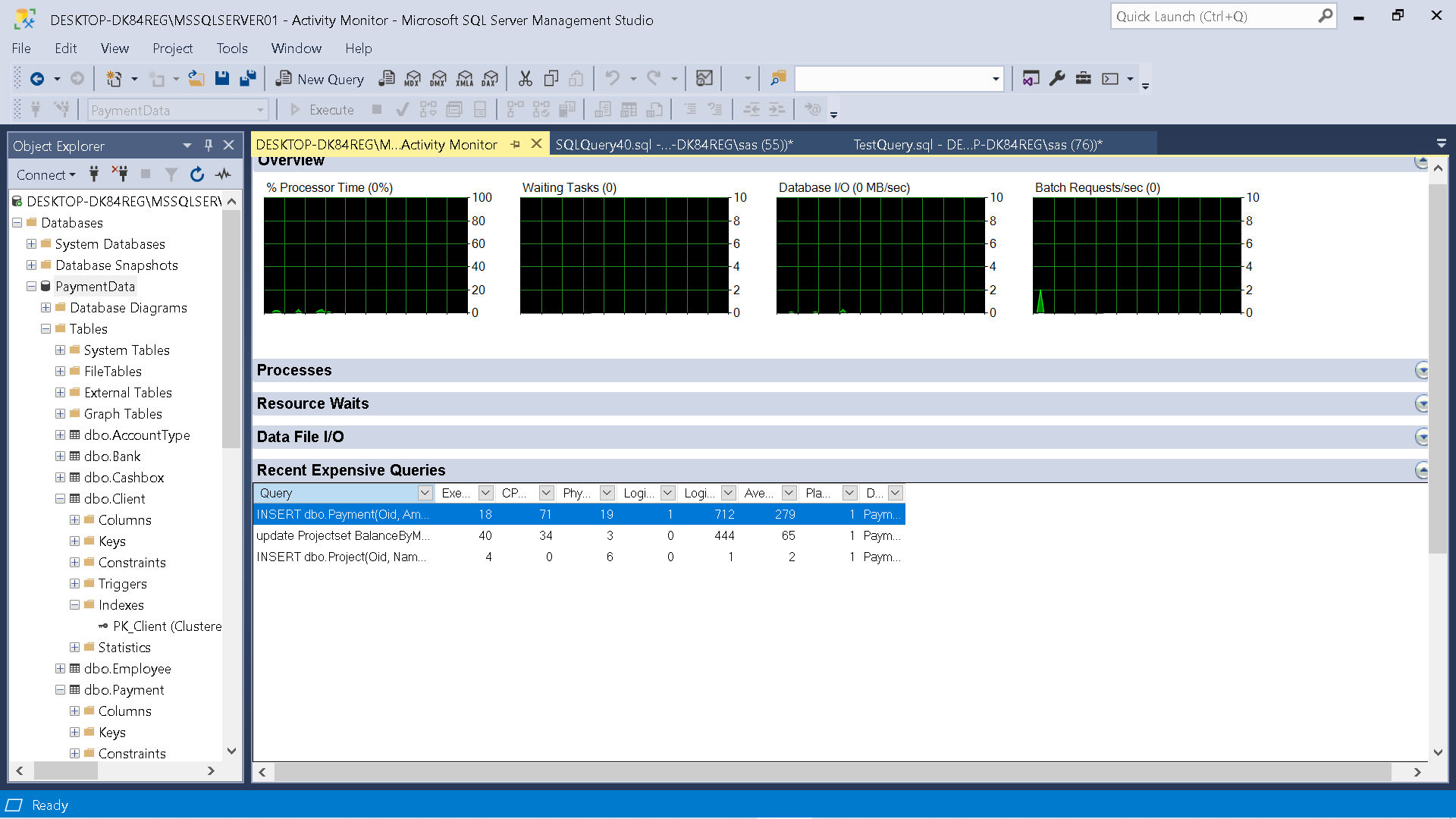


Рис.7. Результаты монитора активности

Данный анализ был проведен на данных размером 50 строк в таблице, однако далее, при работе с 5000 строками запрос по вычислению “BalanceByMaterial” также оказался дорогим (Рис. 8).

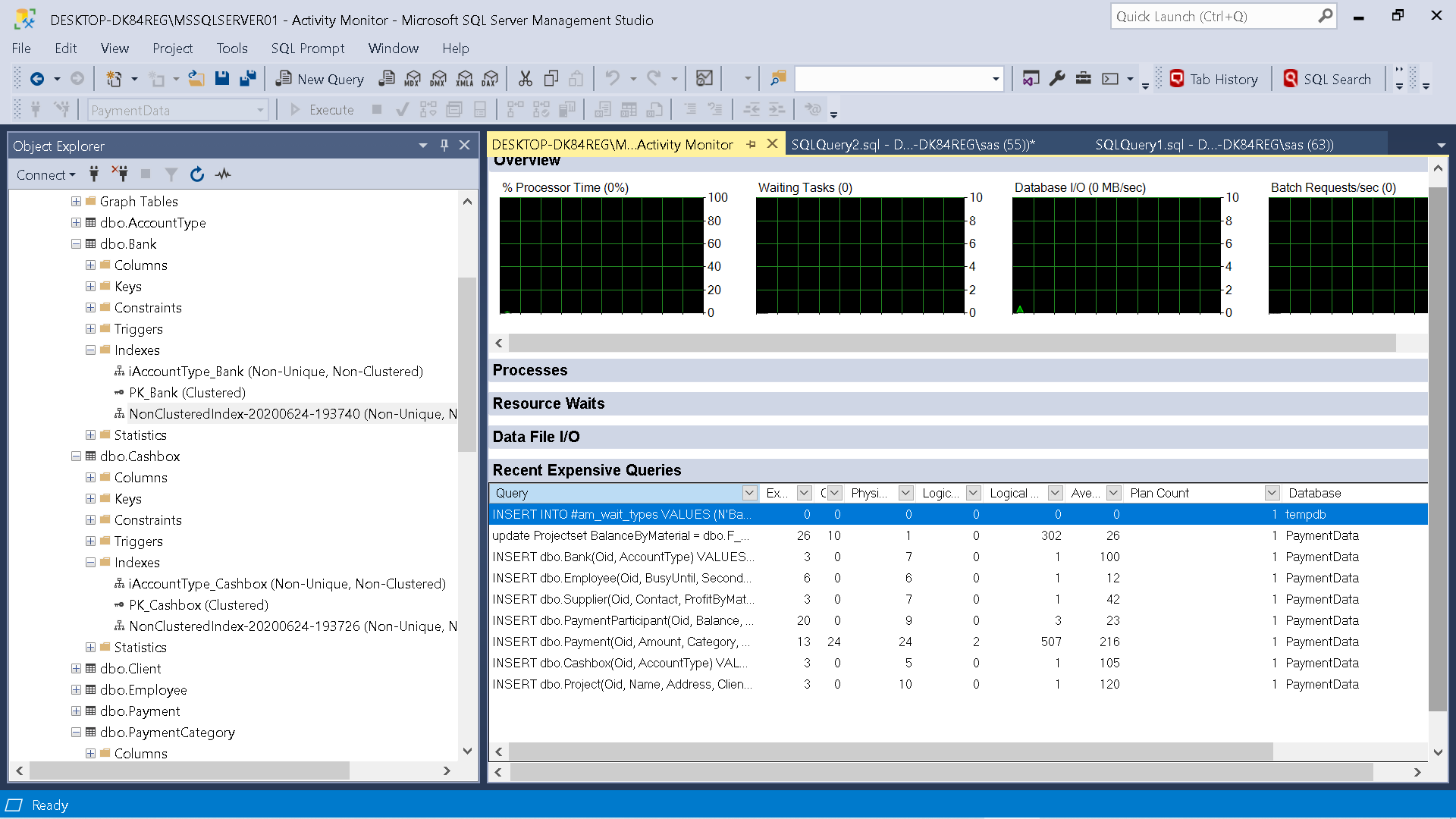


Рис. 8. Результат анализа производительности на 5000 строках

Можно заметить, что запросы по вставке также являются дорогими. Обусловлено это сложностью запросов при вычислении балансов.

Еще одним подходом для определения недостающих индексов была возможность отобразить их с помощью запроса

SET STATISTICS XML ON

Это было произведено для каждого запроса при расчете баланса. В результате в полученных xml файлах не было найдено тега <MissingIndexes/>.

Далее производился анализ с помощью SQL Server Profiler. Данный интерфейс позволяет создавать трассировки, управлять ими и получать результаты, которые можно в дальнейшем анализировать. Запрос был взят из данной статьи (<https://docs.microsoft.com/ru-ru/sql/tools/sql-server-profiler/view-and-analyze-traces-with-sql-server-profiler?view=sql-server-ver15>). В результате получилось подтвердить, что время исполнения скрипта превысило ожидаемое время (Рис. 9).

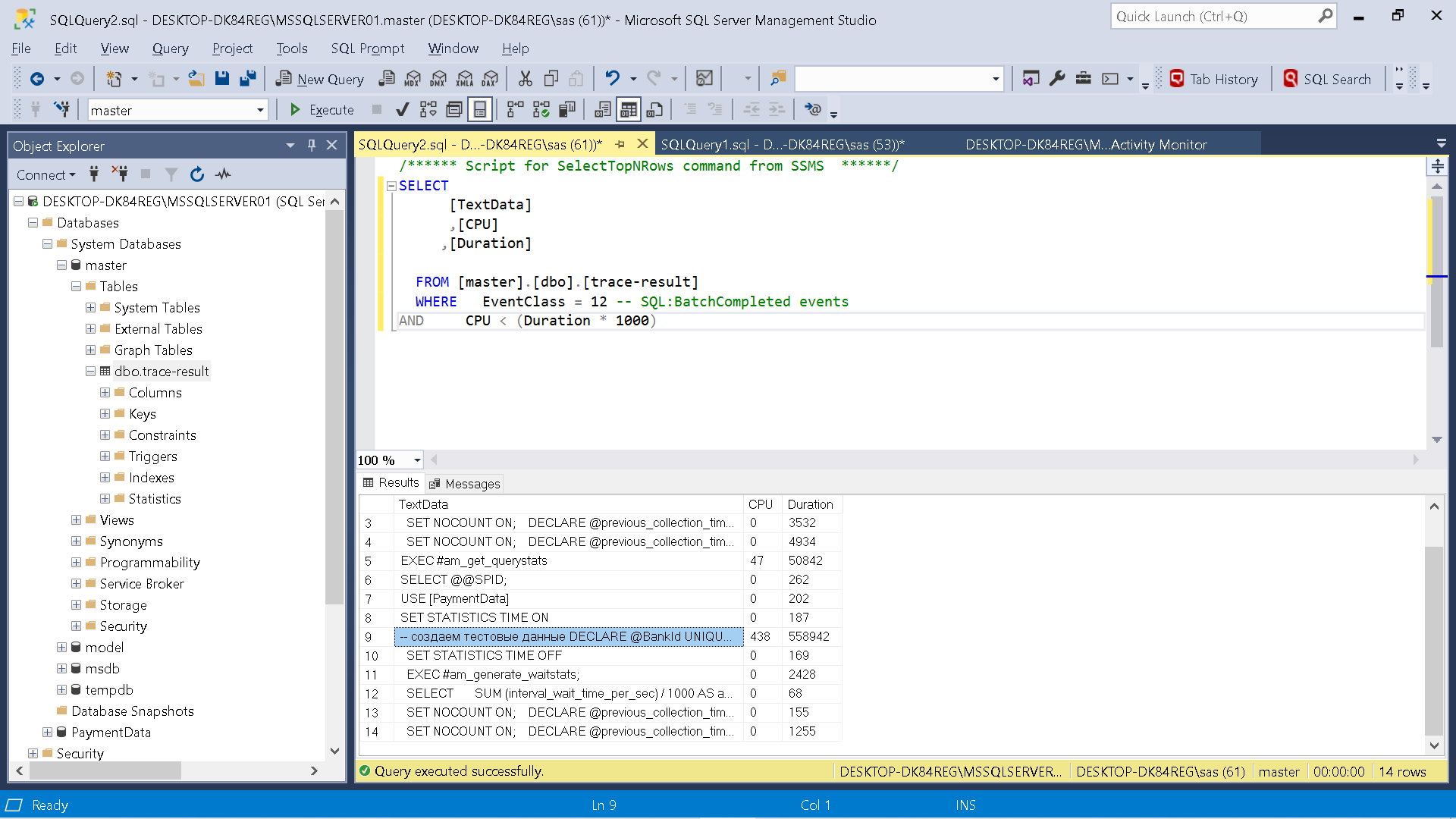


Рис 9. Результаты анализа работы SQL Server Profiler.

Помимо этого, был использован такой показатель, как “Object Execution Statistics”, что отображает время выполнения всех хранимых процедур и триггеров (Рис. 10).

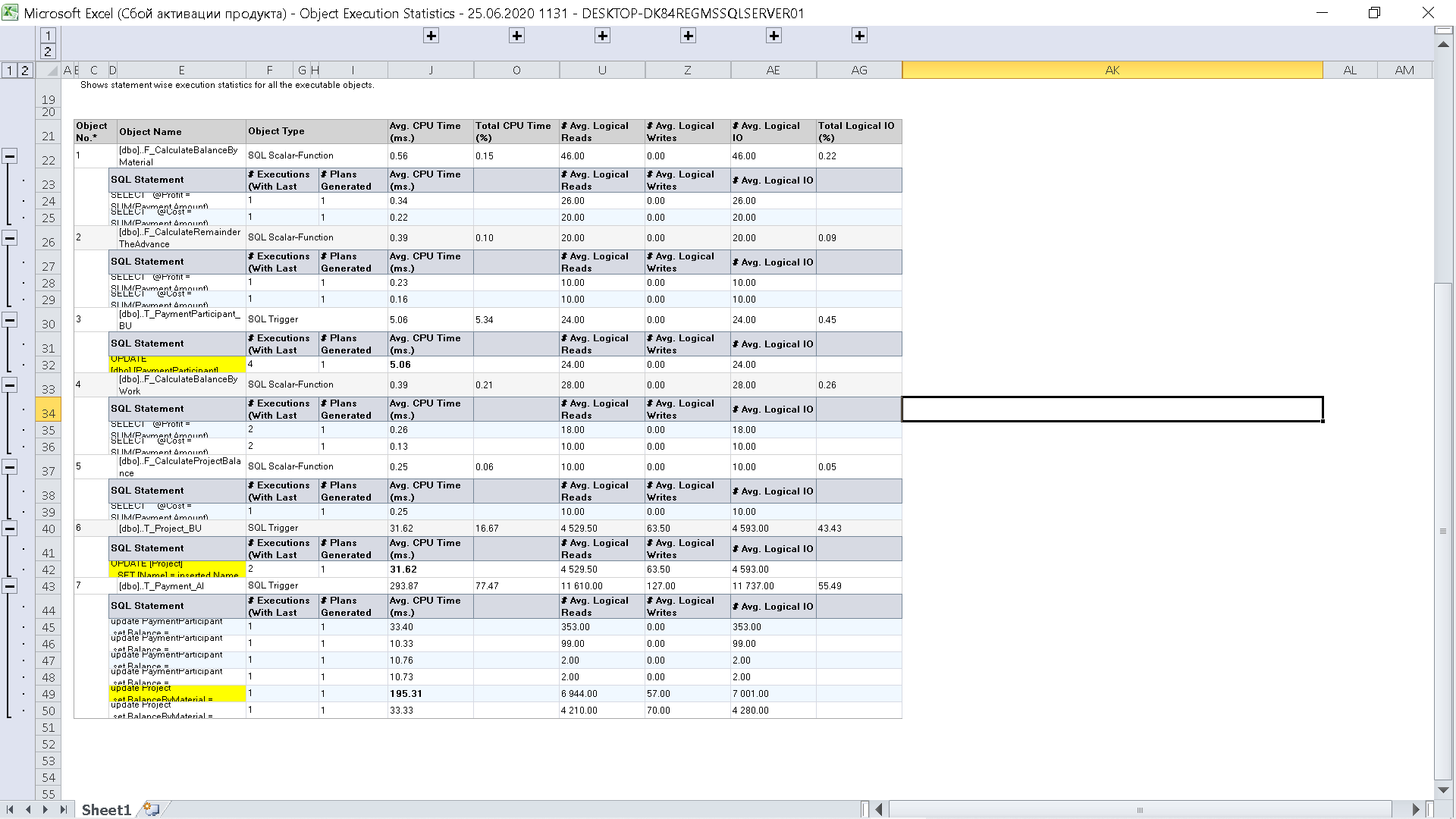


Рис. 10. Статистические данные по триггерам и процедурам

Отсюда были выделены запросы, на которые тратится больше CPU Time. Данная таблица оказалась полезна и для выполнения задания 1 части 2.

Инструмент Database Engine Tuning Advisor позволил изучить, какие поля из таблиц используются чаще всего в вычислениях, чтобы в дальнейшем на основе этого выбирать, что стоит использовать в качестве индексов (Рис.11).

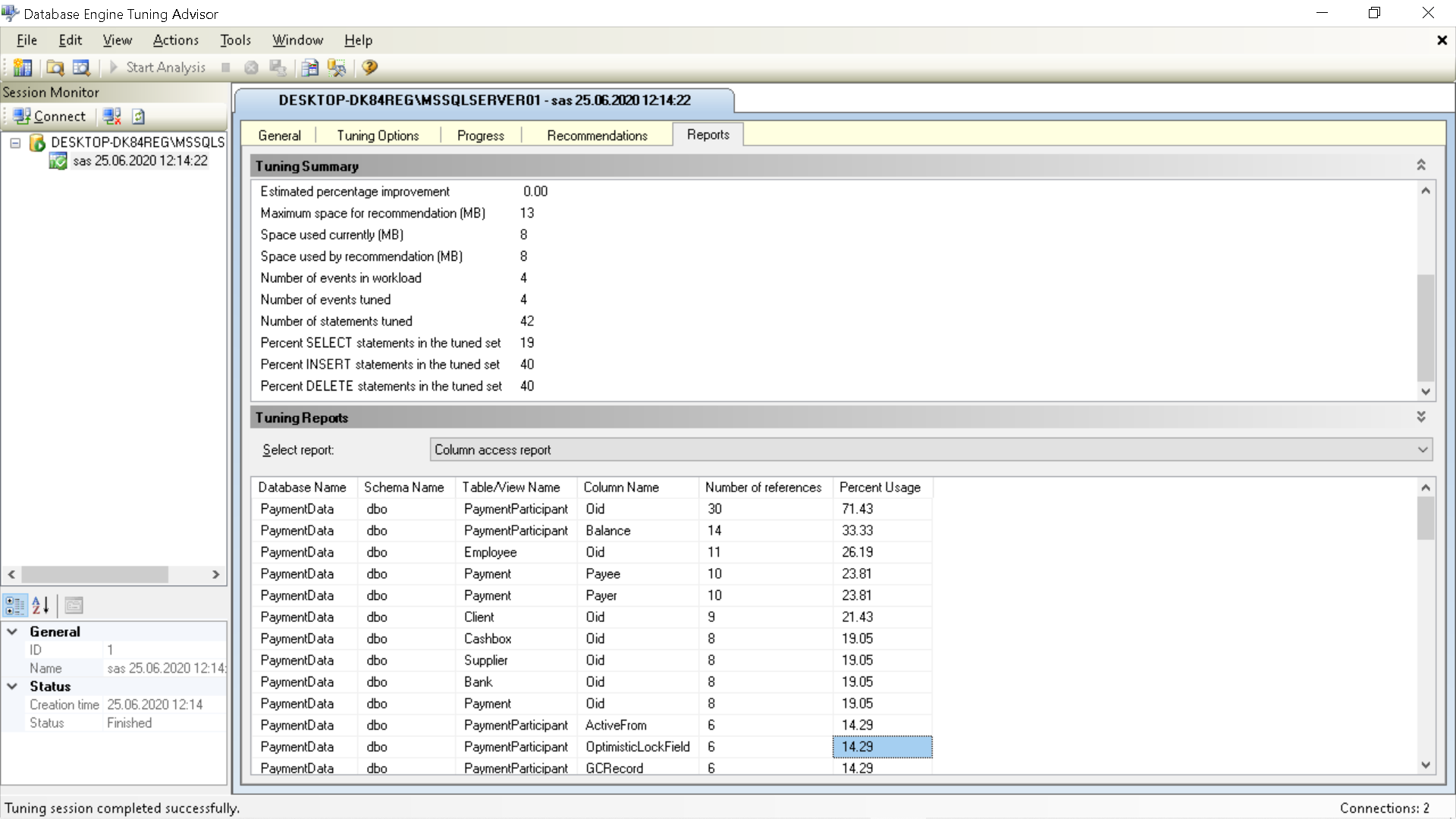


Рис. 11. Таблица частоты использования полей в запросах

Далее был исследован вручную запрос «INSERT dbo.Payment …», в котором производились вычисления баланса. Во внутренних запросах были определены поля, которые используются в части «WHERE». В запросе по обновлению таблицы “PaymentParticipant” были выделены следующие поля:

**SupplierPayer.ProfitByMaterialAsPayer, CashboxAccountType.Name, BankAccountType.Name, PaymentCategory.ProfitByMaterial**, **SupplierPayer.CostByMaterialAsPayer, PaymentCategory.CostByMaterial**

К этим полям были добавлены некластерные индексы. Был проведен анализ на Monitor Activity. На Рис. 12 представлен результат ДО добавления новых индексов, на Рис.13 – после добавления.

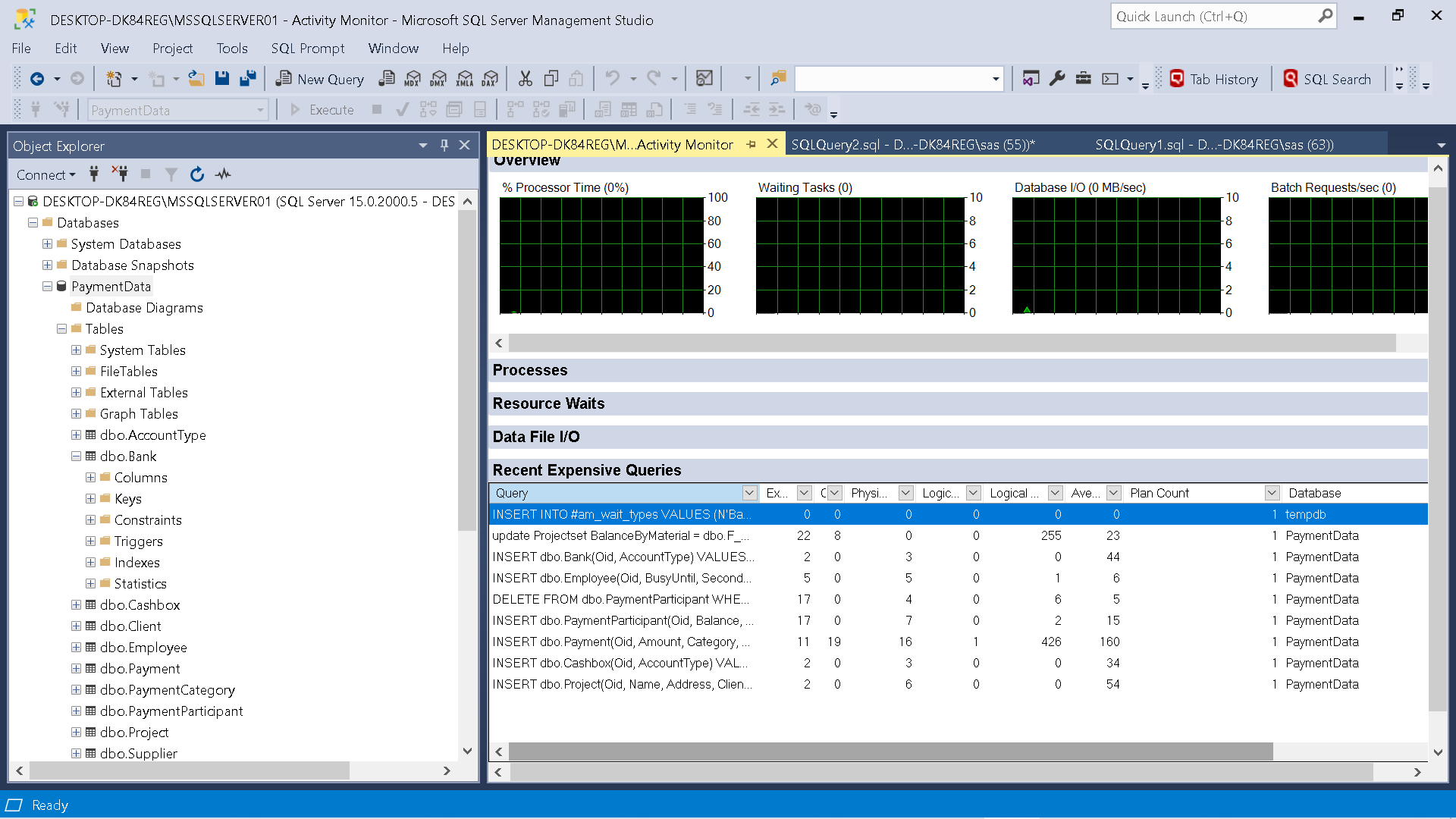


Рис. 12. Дорогие запросы до добавления новых индексов

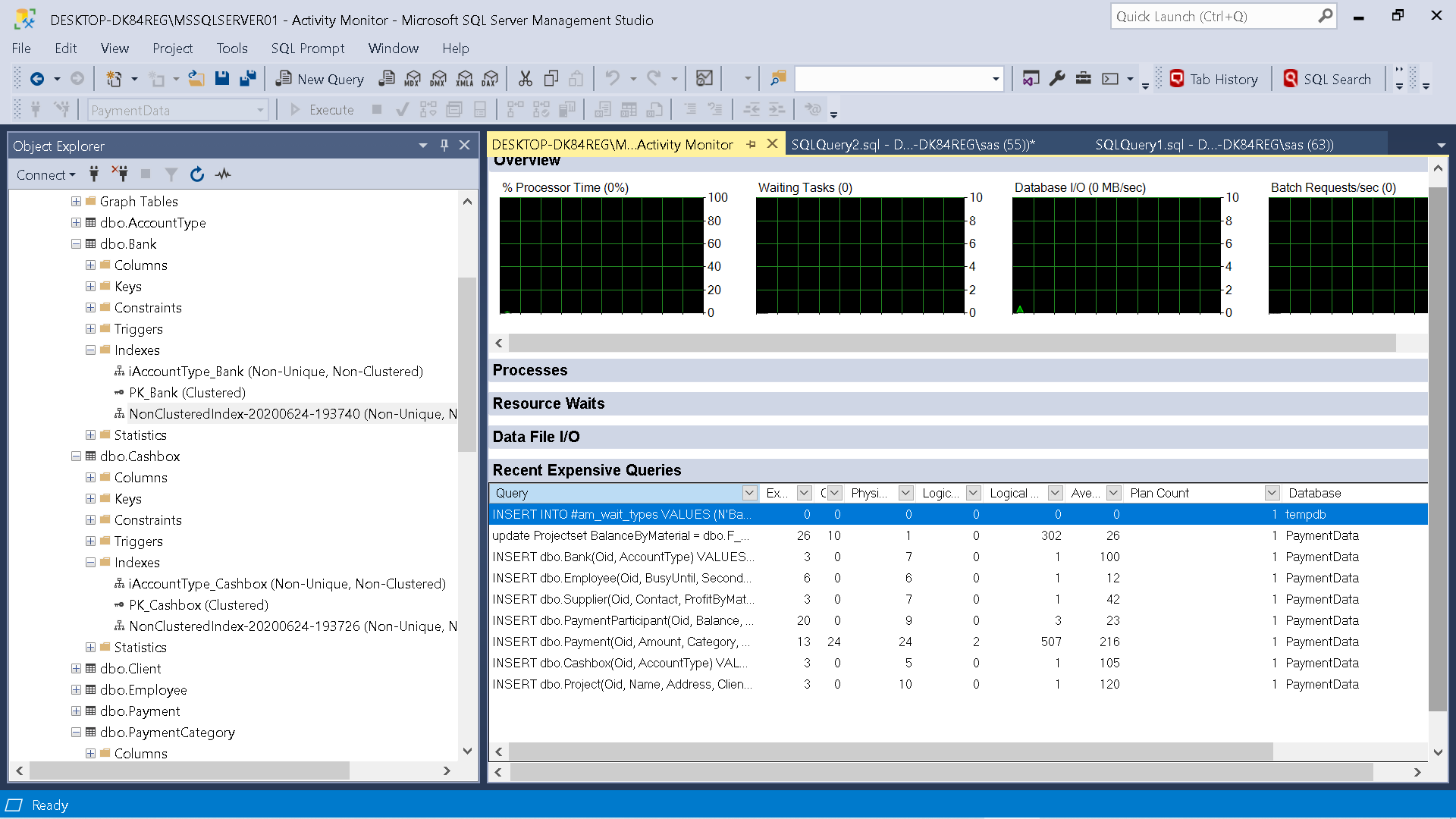


Рис. 13. Дорогие запросы после добавления индексов

По рисункам видно, что производительность ухудшилась (было 22мс, стало 26 мс).

Далее был проведен эксперимент, по добавлению лишь части из перечисленных индексов. Здесь для анализа использовалось расширение Client Statistics (Рис. 14).

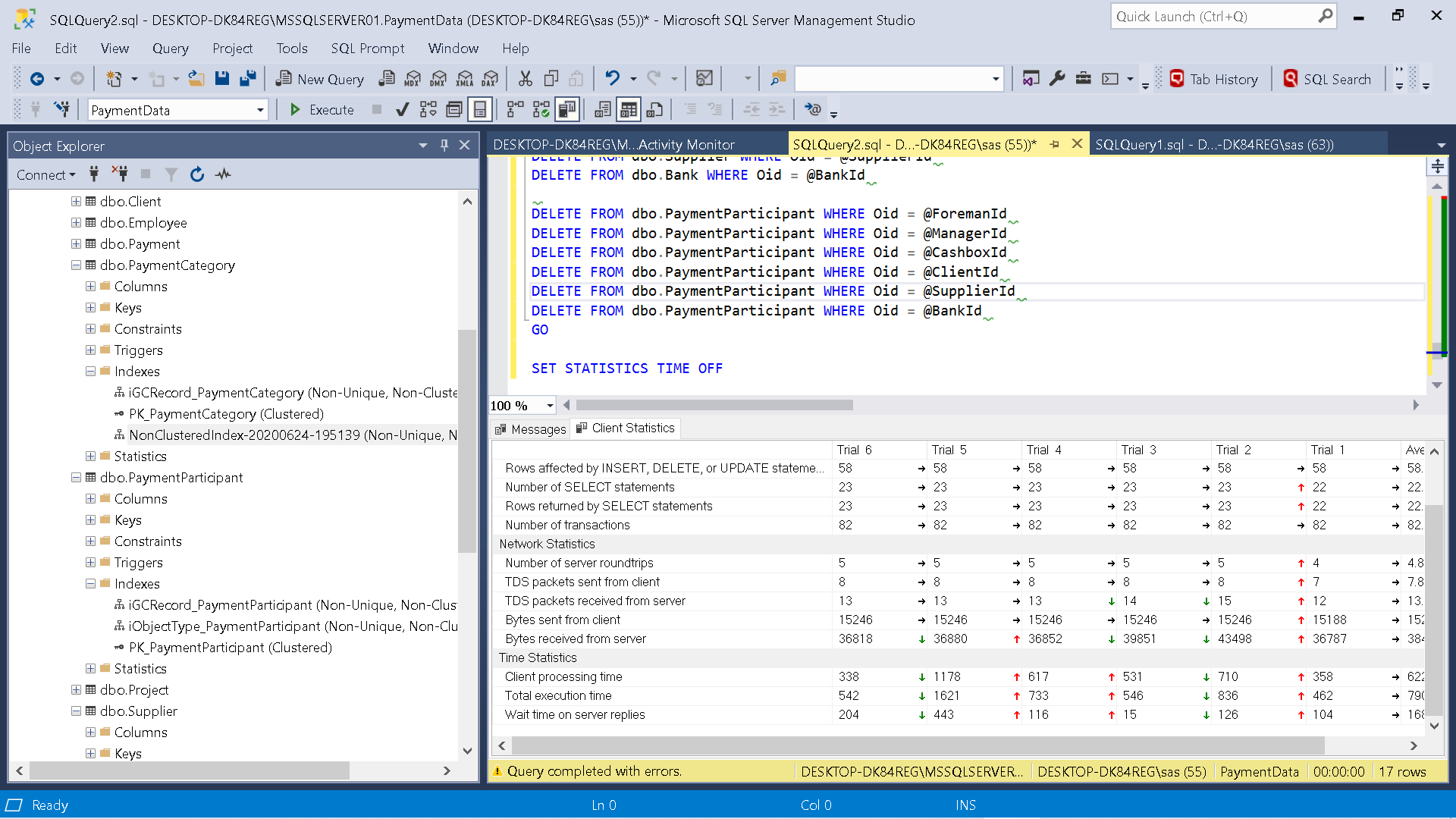


Рис. 14. Эксперимент по добавлению части индексов

Описание:

Trial 1: нет новых индексов

Trial 2: индексы в PaymentParticipant, Bank, Cashbox, Supplier

Trial 3: убран индекс из Supplier

Trial 4: убран индекс из PaymentCategory

Trial 6: возвращен индекс в PaymentCategory

Вывод: добавление индексов даже таким способом не улучшает производительность.

Далее был изучен запрос по обновлению таблицы Project (так как тоже занимает много времени). Был добавлен индекс к полю PaymentCategory.Name. По результатам (Рис.15) видно, что ничего не улучшилось.

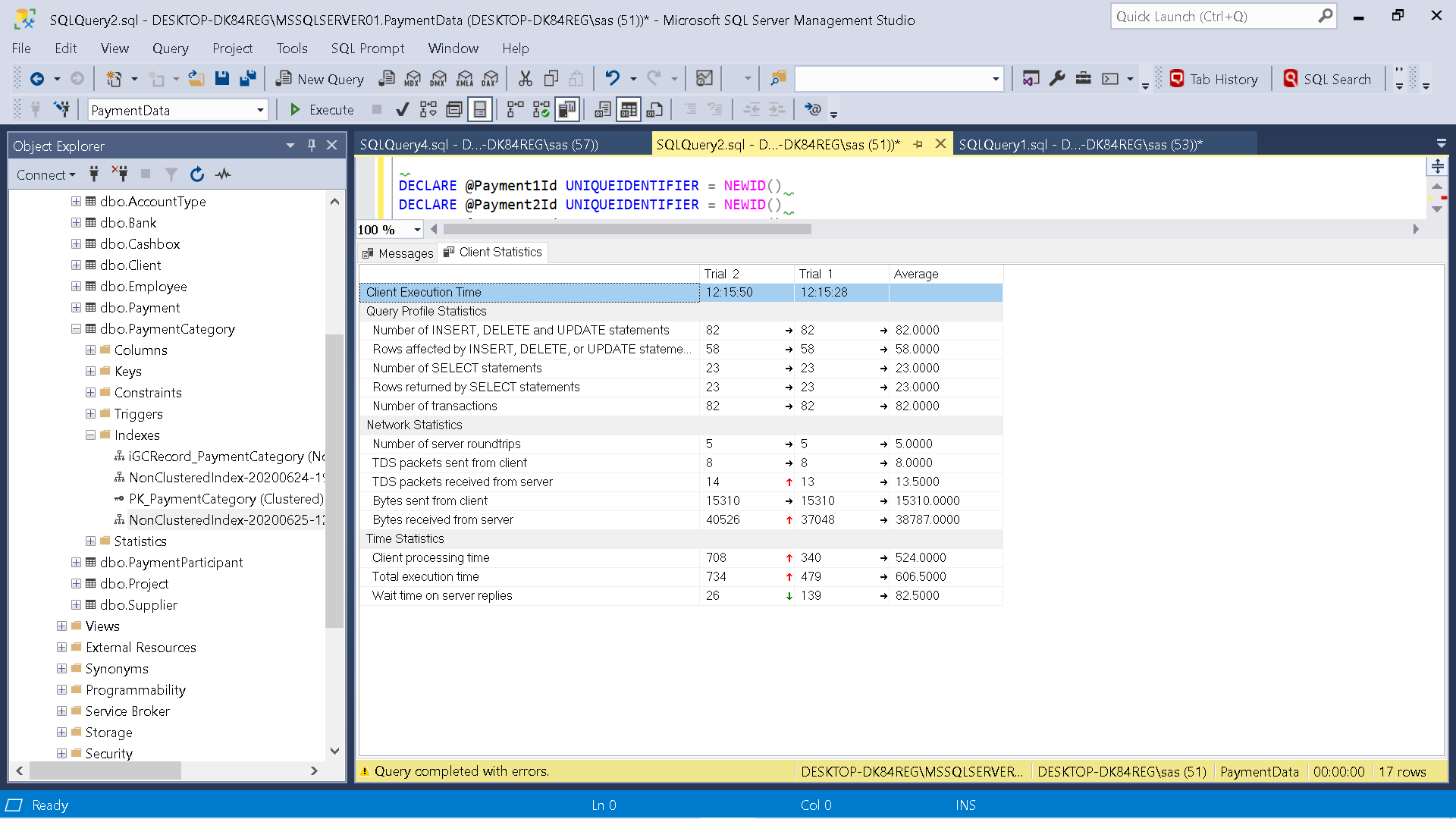


Рис. 15. Результаты добавления индекса по Payment.Name

В ходе работы я ознакомилась с тем, какие индексы вводились коллегами за предыдущие года. С использованием нагрузочного тестирования (инструмент SQL Query Stress) была проведена следующая работа:

1.Нагрузочного тестирование по 1000 итераций на 5 потоков без использования новых индексов (Рис. 16).

2.Нагрузочное тестирование с использованием индексов по dbo.Supplier (CostByMaterialAsPayer), dbo.Supplier (ProfitByMaterialAsPayee), dbo.Supplier (ProfitByMaterialAsPayer), dbo.PaymentCategory (NotInPaymentParticipantProfit), dbo.PaymentCategory (CostByMaterial), dbo.PaymentCategory (ProfitByMaterial), dbo.PaymentCategory (Name), dbo.AccountType (Name) (Рис. 17).

3.Нагрузочное тестирование с использованием составных индексов (Рис. 18):

dbo.Supplier (CostByMaterialAsPayer) + dbo.Supplier (ProfitByMaterialAsPayee) + dbo.Supplier (ProfitByMaterialAsPayer);

dbo.PaymentCategory (NotInPaymentParticipantProfit) + dbo.PaymentCategory (CostByMaterial) + dbo.PaymentCategory (ProfitByMaterial) + dbo.PaymentCategory (Name);

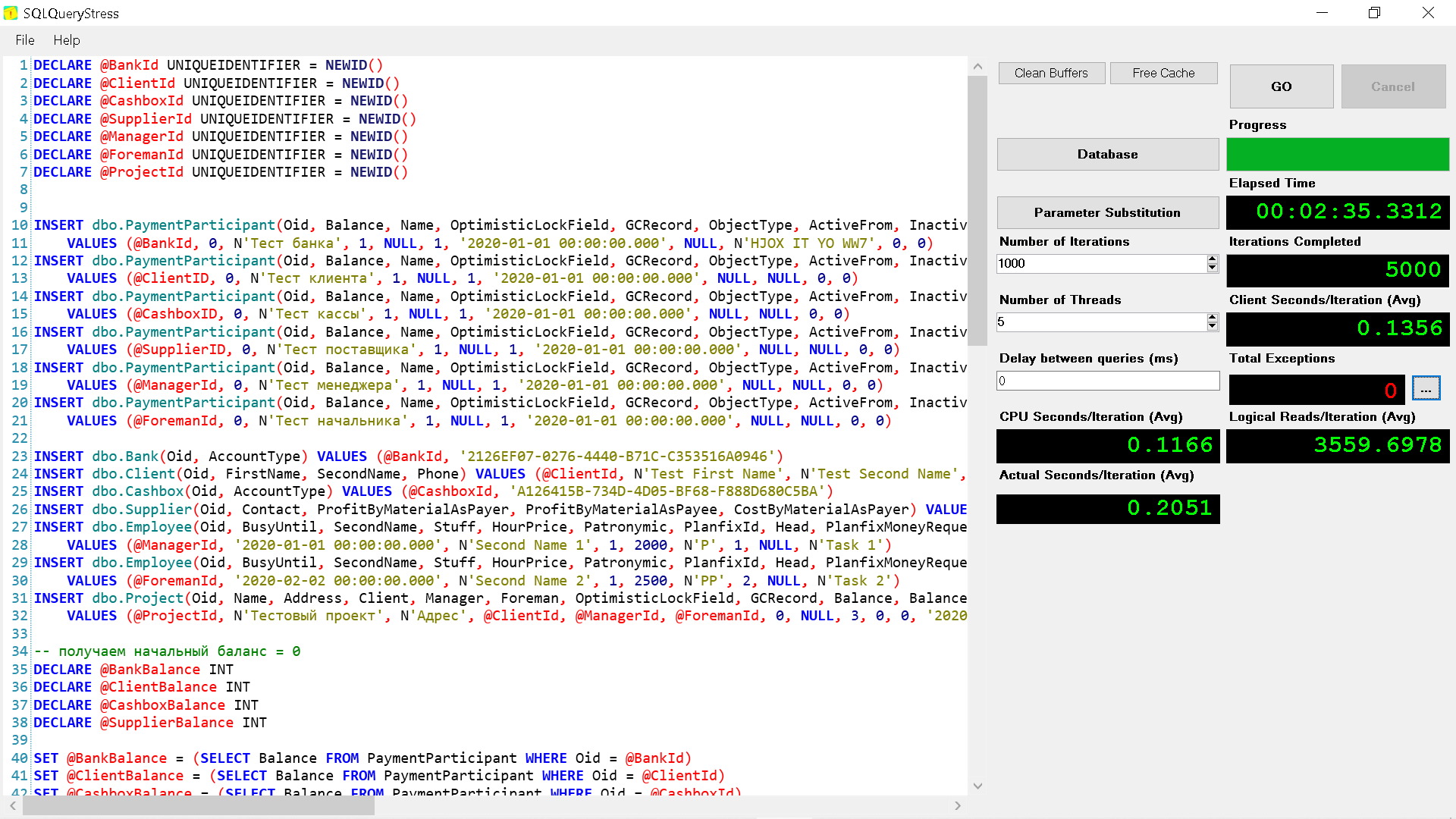


Рис. 16. Результаты нагрузочного тестирования без новых индексов

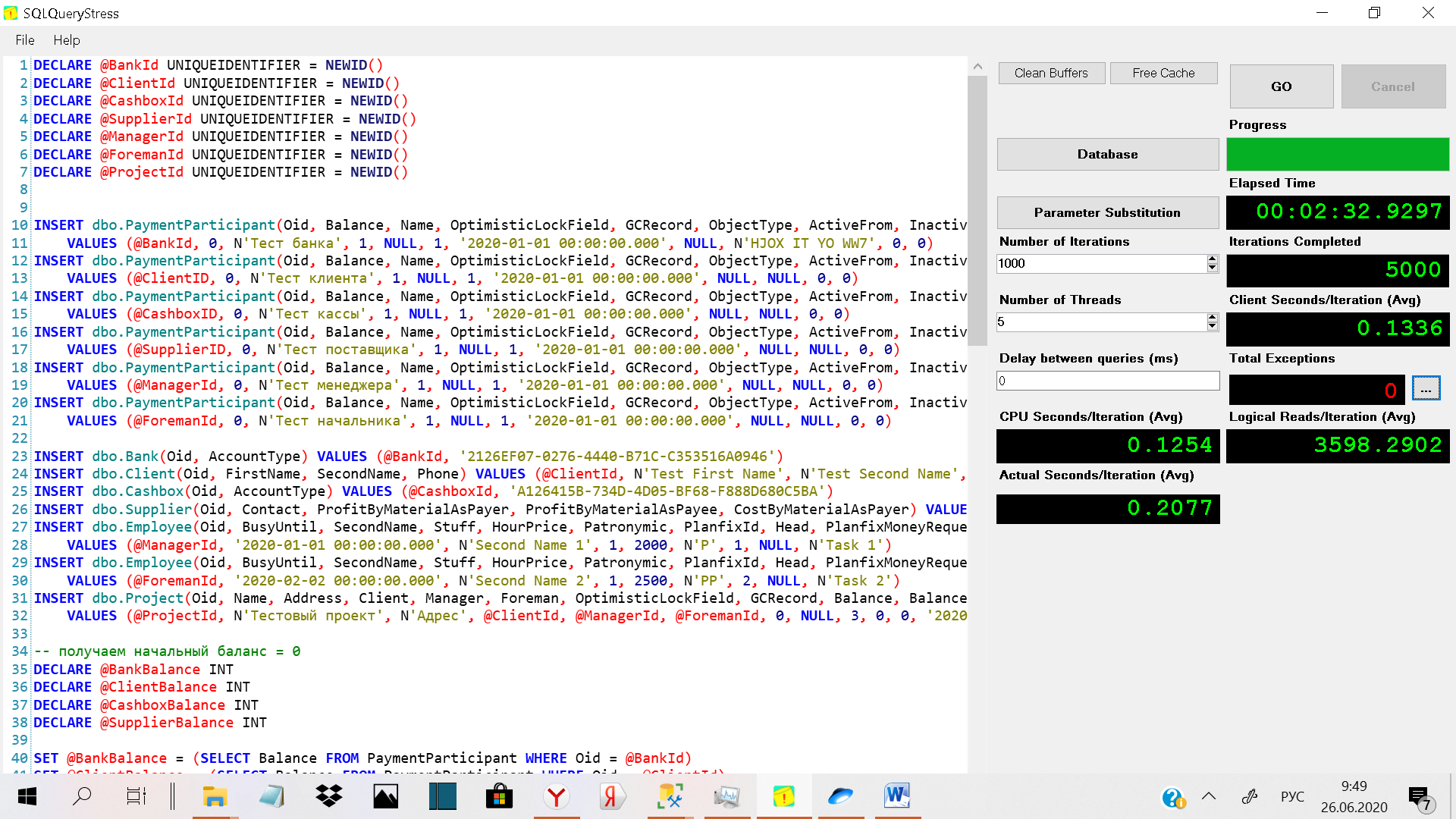


Рис. 17. Результаты нагрузочного тестирования с новыми индексами

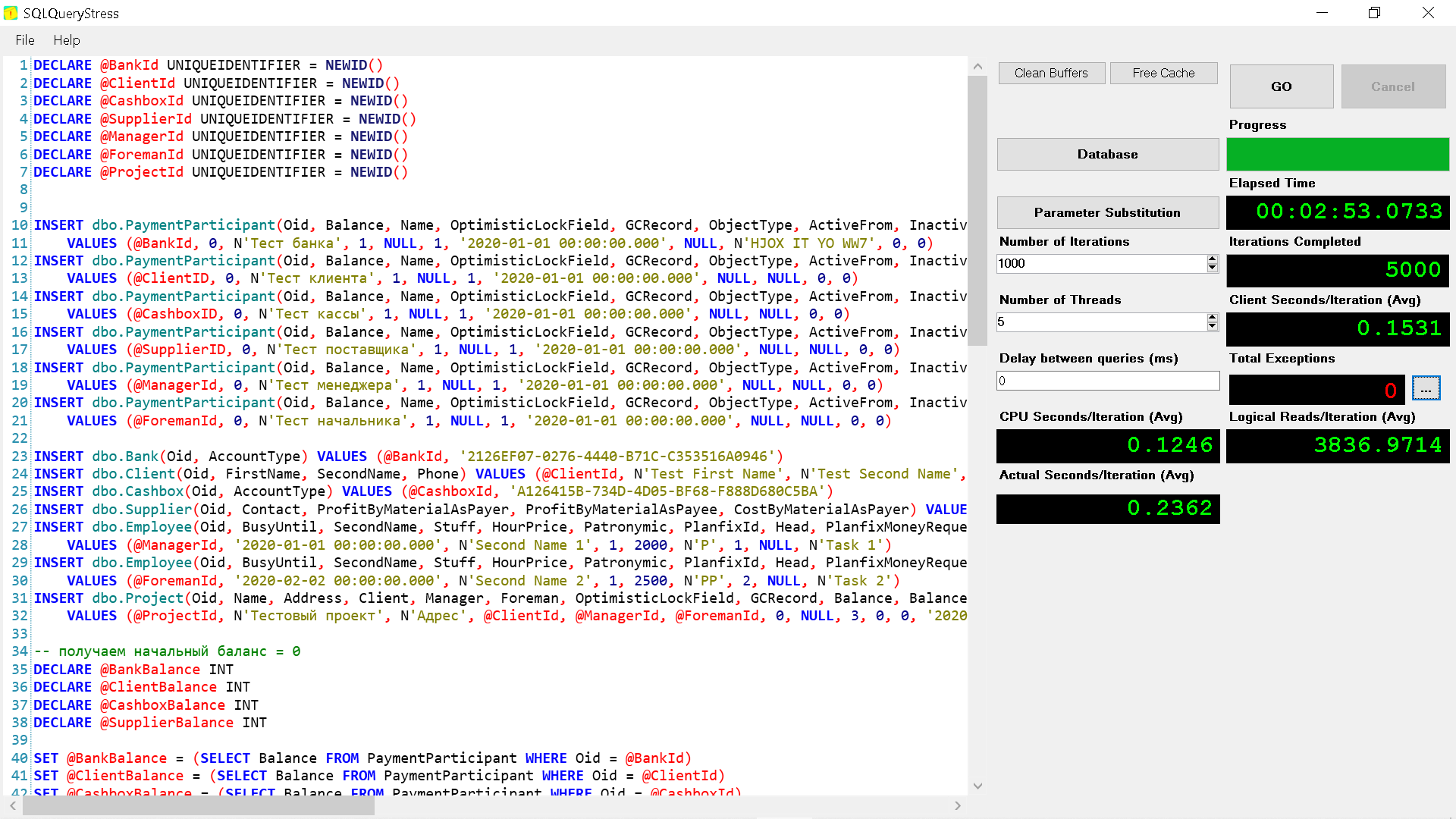


Рис. 18. Результаты нагрузочного тестирования с составными индексами

Результаты:

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **CPU (Avg)** | **Elapsed time** | **Actual query time** |
| Нет индексов новых | 0.1166 | 2:35.332 минуты | 0.2051 |
| Новые индексы (7 шт.) | 0.1254 | 2:32.9297 минуты | 0.2077 |
| Составные индексы | 0.1246 | 2:53.0733 минуты | 0.2362 |

Таблица 4. Сравнение результатов нагрузочного тестирования

Можно отметить, что общее время тестирования при добавлении новых индексов уменьшилось, однако среднее время обращения к процессору и среднее время выполнения всего скрипта увеличилось.

В ходе работы была попытка анализировать изменение плана выполнения запроса в зависимости от добавления индексов, но для каждого запроса в транзакции планы оказались очень большими, что вызвало сложности в анализе.

**Вывод:** В ходе анализа было принято решение, что добавление индексов не приводит к оптимизации производительности. Обусловлено это тем, что при INSERT происходит изменение индексов, их реорганизация, что приводит к плохому результату при работе с большими данными. Здесь чем больше индексов, тем хуже. Поэтому логичнее оставить существующие индексы по первичным и внешним ключам.

Статьи, что помогли мне делать анализ:

1. <https://habr.com/ru/post/336586/>
2. <https://habr.com/ru/post/310328/>
3. <http://wikie.lexema.ru/index.php/Анализ_запросов_с_помощью_SQL_Profiler>
4. <http://ts-soft.ru/blog/sql-optimization-1>

**Часть 2**

**Выполнили**: Асташенкова Лада, Ли Цзин.

**Задача 1. Оценка затрат на выполнение операций расчета балансов**

При внесении информации о платеже в БД возникает необходимость перерасчета балансов. На основании статистики выполнения объектов (рис. 10) можно привести следующие показатели затраченного времени ЦП на выполнение различных типов объектов:

1. 77% - триггер T\_Payment\_AI, который запускает функцию расчета баланса после обновления для таблицы PaymentParticipant;
2. 17% - триггер T\_Project\_BU, который запускает функции расчета балансов;
3. 5% - триггер T\_PaymentParticipant\_BU, который запускает функцию расчета баланса в своей таблице перед обновлением;
4. 0.21% - функция F\_CalculateBalanceByWork, проводящая расчет баланса по работе;
5. 0.15% - функция F\_CalculateBalanceByMaterial, проводящая расчет баланса по материалам;
6. 0.1% - функция F\_CalculateRemainderTheAdvance, проводящая расчет остатка аванса;
7. 0.05% - функция F\_CalculateProjectBalance, проводящая расчет баланса по проекту.

**Задача 2. Сценарий оптимизации механизмов расчета**

Пусть введены две роли пользователей:

* Оператор, который вводит и обновляет информацию о платежах;
* Бухгалтер-аналитик, который отслеживает информацию о балансах и заведенных платежах.

На этапе работы оператора необходимо минимизировать его время работы, тогда расчет балансов можно отложить. Расчет балансов может запускаться в момент запуска процедуры вывода баланса бухгалтером-аналитиком, либо до этого момента, чтобы уменьшить время ожидания пользователя. Т.е. на этапе работы бухгалтера-аналитика расчет балансов уже должен быть выполнен, если не полностью, то хотя бы частично.

Тогда оптимизировать работу БД можно следующим образом:

* каждую операцию вставки информации о платежах не производить по отдельности, вместо этого данные операции записываются в файл;
* при достижении заданного количества N операций в файле запускается механизм расчета балансов и файл очищается;
* количество операций N, запускающих процесс, вычисляется на основе статистических данных о работе оператора и бухгалтера-аналитика.

**Задача 3. Оценка недостатков с точки зрения пользователей**

1. Информация о платежах обновляется с задержкой по отношению к действиям оператора, на его работу это влияет только в случае необходимости использовать информации о предыдущих платежах, но в описание его действий это не входит;
2. Бухгалтер-аналитик работает не всегда с актуальной информацией о платежах и балансах, но если расчет производится регулярно, то с такой же регулярностью и обновляется информация у бухгалтера-аналитика. Таким образом, актуальностью информации можно управлять путем изменения показателя N, запускающего очередной процесс расчета балансов. Расчет должен производиться чаще (N должно быть меньше), если бухгалтер-аналитик нуждается в более актуальной информации;
3. Показатель N, запускающий процесс расчета, нуждается в настройке. Если оператор вводит недостаточное количество запросов на вставку или обновление, то данные запросы, будучи отложенными, будут дольше ожидать своего исполнения. Таким образом, показатель N может менять в течение суток или разных периодов работы пользователей;
4. Дополнительные расходы памяти на файл с отложенными операциями.

**Заключение**

В ходе выполнения работы команда ознакомилась с Microsoft SQL Server, его возможностями и предоставляемыми утилитами, научились генерировать данные, писать небольшие тесты, анализировать зависимость производительности от изменений в индексах. Были оценены затраты на выполнение операций расчета баланса, предложен сценарий оптимизации и рассмотрены его основные недостатки.