Итоговое задание по курсу Оптимизация БД

Итоговый отчет по заданию.

Задание было выполнено в команде, состоящей из нескольких человек:
Ян Тяньюй ;Шэнь Жуйцзи ;Фу Цзухэ ;Ху Юйхан(в алфавитном порядке)

Ян Тяньюй: Разработка кода в заданиях второго уровня, введение ролей, назначение прав доступа, разработка кода в заданиях третьего уровня для реализации отложенной функциональности, реализация оптимизированного решения.

Шэнь Жуйцзи: Оценка производительности после введения ролей пользователей во вторичных задачах, введения тестовых данных, оптимизации дизайна решения и оценки недостатков оптимизационного решения.

Фу Цзухэ: Оптимизация кода индексации и вставка тестовых данных в задачу первого уровня; тестирование производительности нового и оригинального индексов и получение результатов производительности. В заданиях уровня 3 проверяется производительность сценариев с задержкой и реального времени и получаются результаты производительности

Ху Юйхан: Проверка результатов тестирования в задаче первого уровня и в задаче третьего уровня. Оценка новой схемы индексирования цитат на основе полученных результатов, оценка схемы с задержкой и заключение. Объедините все элементы для написания окончательного отчета по заданию.

P.S. Данный документ является отчетом по одной задаче, кодовая часть представлена в отдельном документе.

Задачи первого уровня

Описание задачи: внедрение индекса для повышения производительности операций вставки и модификации платежа без модификации компонентов программного обеспечения

Выполнение задачи:

В исходном коде показано, как создать таблицу с именем [dbo]. [Payment] и создать несколько одноколоночных индексов для этой таблицы.

Для одностолбцовых индексов можно выполнять быстрые одностолбцовые запросы: производительность хороша для одностолбцовых запросов. Но в многостолбцовых запросах производительность низкая: для запросов, включающих несколько столбцов, одностолбцовые индексы имеют ограниченный эффект. На практике запросы часто включают несколько столбцов (например, Payer, Payee и Project). Например: найти все записи о транзакциях плательщика и получателя; найти все записи о платежах, связанных с проектом. В этом случае использование объединенного индекса может значительно повысить производительность запроса. Объединенные индексы могут повысить эффективность сложных запросов, уменьшить объем хранилища и лучше соответствовать реальным требованиям запроса. Поэтому мы создаем индекс объединения и удаляем исходный одноколоночный индекс. Оптимизация многостолбцовых запросов для союзных индексов: повышение производительности запросов с несколькими столбцами (например, Payer, Payee и Project). Меньше места в хранилище: объединенный индекс может заменить несколько одноколоночных индексов, что позволяет сократить объем занимаемого пространства.

Результаты выполнения:

Тест производительности сырого индекса:

------ вставка 10 000 строк ---------
Время выполнения SQL Server.
Процессорное время = 3797 мс, время занятости = 5174 мс.

(затронуто 10000 строк)

Вставка занимает (прошедшее время вставки): 5,176000 с

----- тест обновления 1-й --------
Время выполнения SQL Server.
Процессорное время = 9609 мс, время занятости = 14087 мс.

(затронуто 40000 строк)

Операция обновления занимает (затраченное время на обновление): 14,093000 с

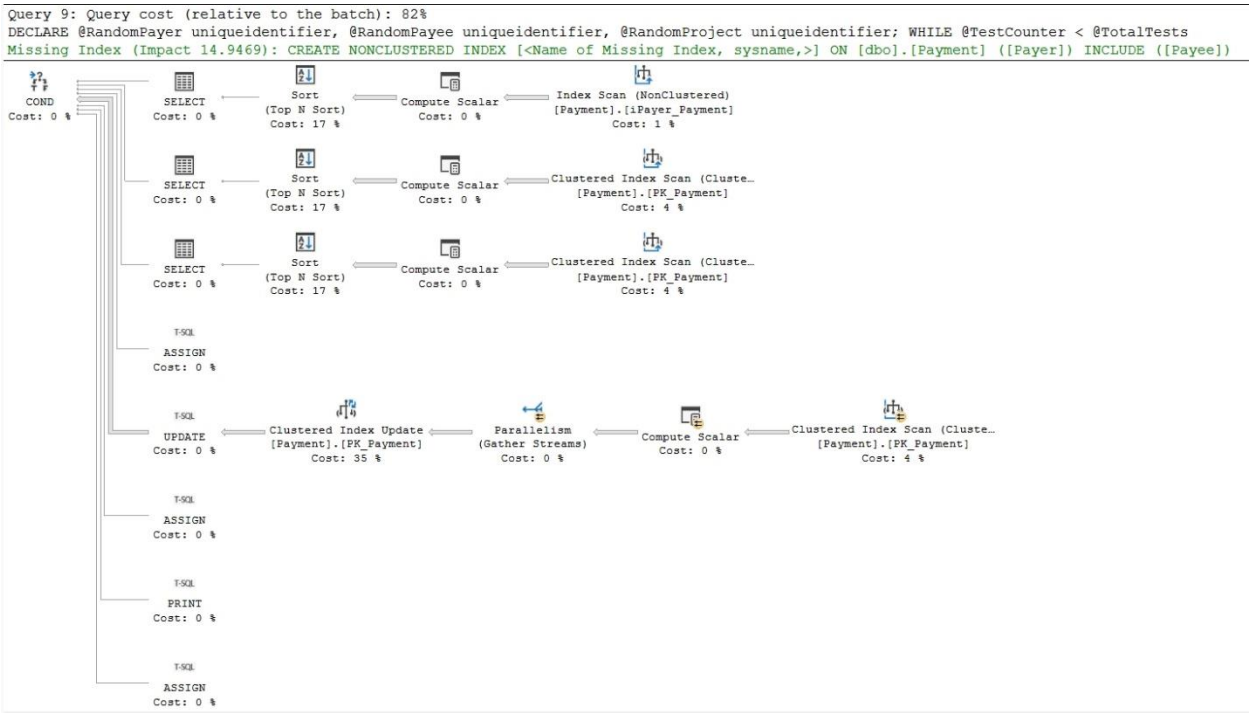
----- тест обновления 2-й --------
Время выполнения SQL Server.

Процессорное время = 2781 мс, время занятости = 4192 мс.

(затронуто 10000 строк)

Операция обновления занимает (затраченное время операции обновления): 4,190000 с

Свойства таблицы платежей после ----- тест оригинального индекса -----
Прилагается следующий снимок экрана



Тестирование производительности сырьевого индекса

```
 SQL Server 执行时间:
   CPU 时间 = 3797 毫秒，占用时间 = 5174 毫秒。

(10000 rows affected)
test insert data

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Вставка занимает(插入耗时): 5.176000 s
```

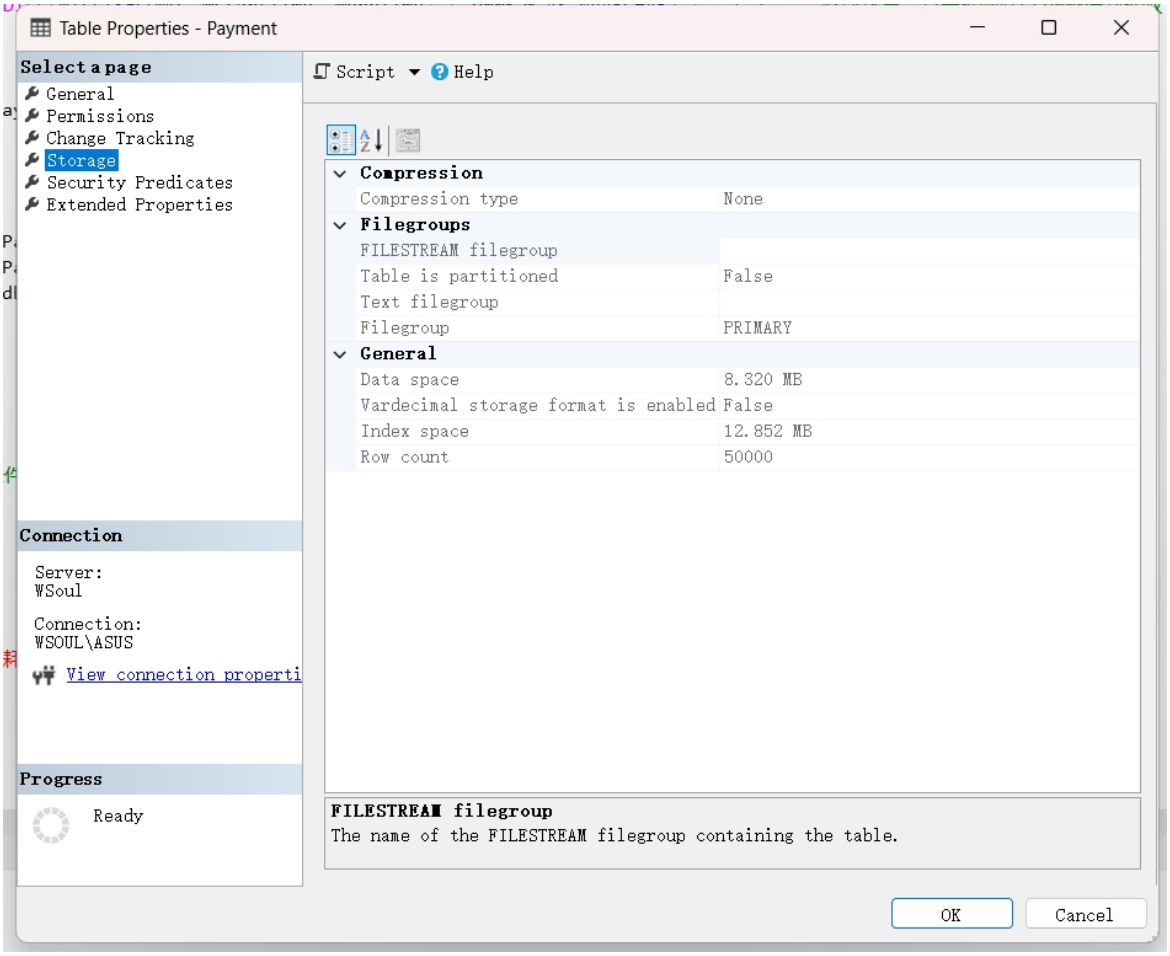
Потребление времени на вставку сырья

```
 SQL Server 执行时间:
   CPU 时间 = 9609 毫秒，占用时间 = 14087 毫秒。

(40000 rows affected)

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Операция обновления занимает(更新操作耗时): 14.093000 s
```

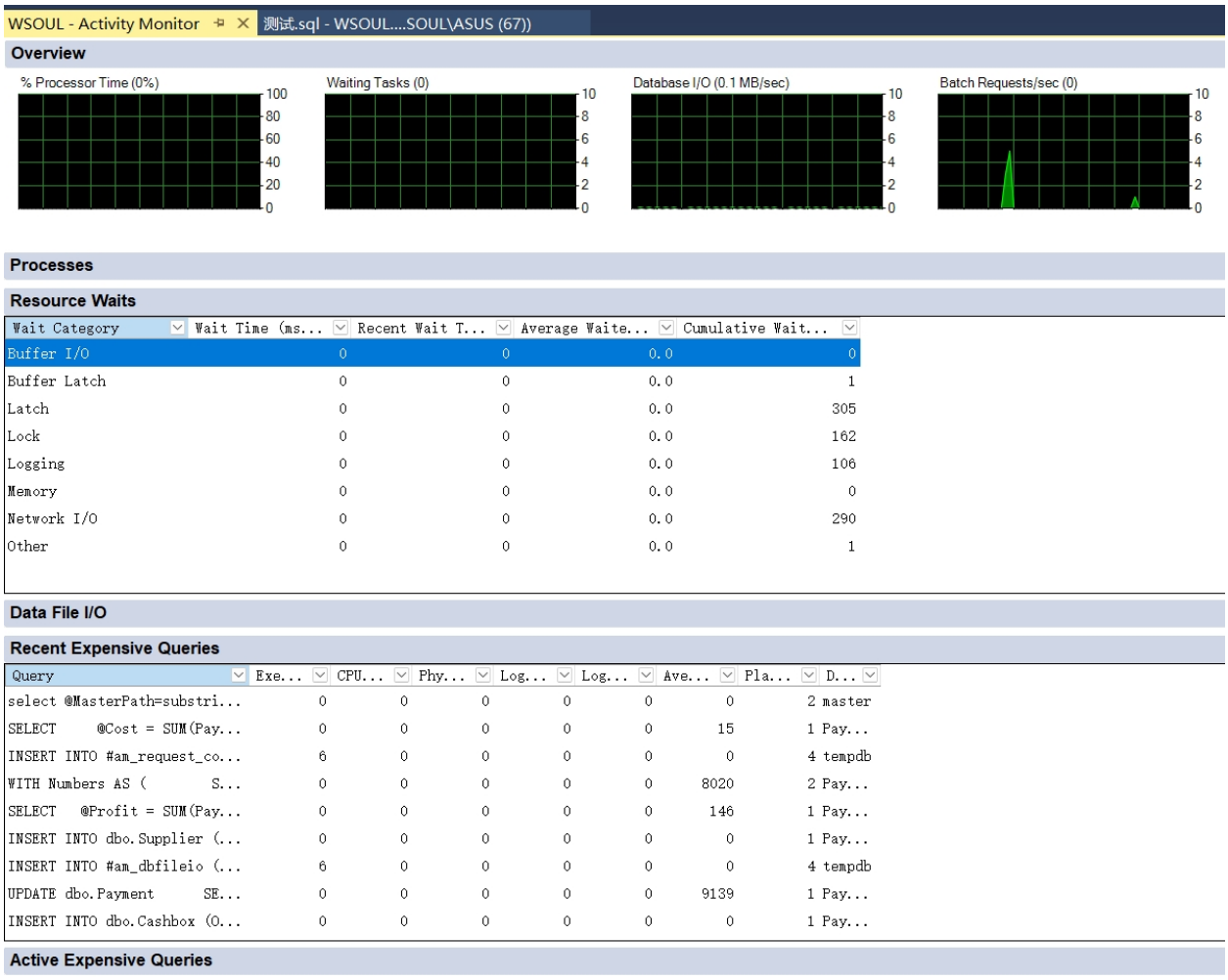
Оригинальный тест обновления занял 1

```
 SQL Server 执行时间:
   CPU 时间 = 2781 毫秒，占用时间 = 4192 毫秒。

(10000 rows affected)

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Операция обновления занимает(更新操作耗时): 4.190000 s
```

Первоначальный тест обновления занял 2



Свойства таблицы платежей после проверки сырого индекса



Активное слушание под индексом Raw

Новый тест производительности индекса:

------ Вставка 10 000 строк ---------

Время выполнения SQL Server.

Процессорное время = 1750 мс, время занятости = 2196 мс.

(затронуто 10000 строк)

Вставка занимает (время выполнения вставки): 2,213000 с

----- тест обновления 1-й --------

Время выполнения SQL Server.

Процессорное время = 8266 мс, время занятости = 9215 мс.

(затронуто 40000 строк)

Операция обновления занимает (затраченное время на обновление): 9,220000 с

----- тест обновления 2-й --------

Время выполнения SQL Server.

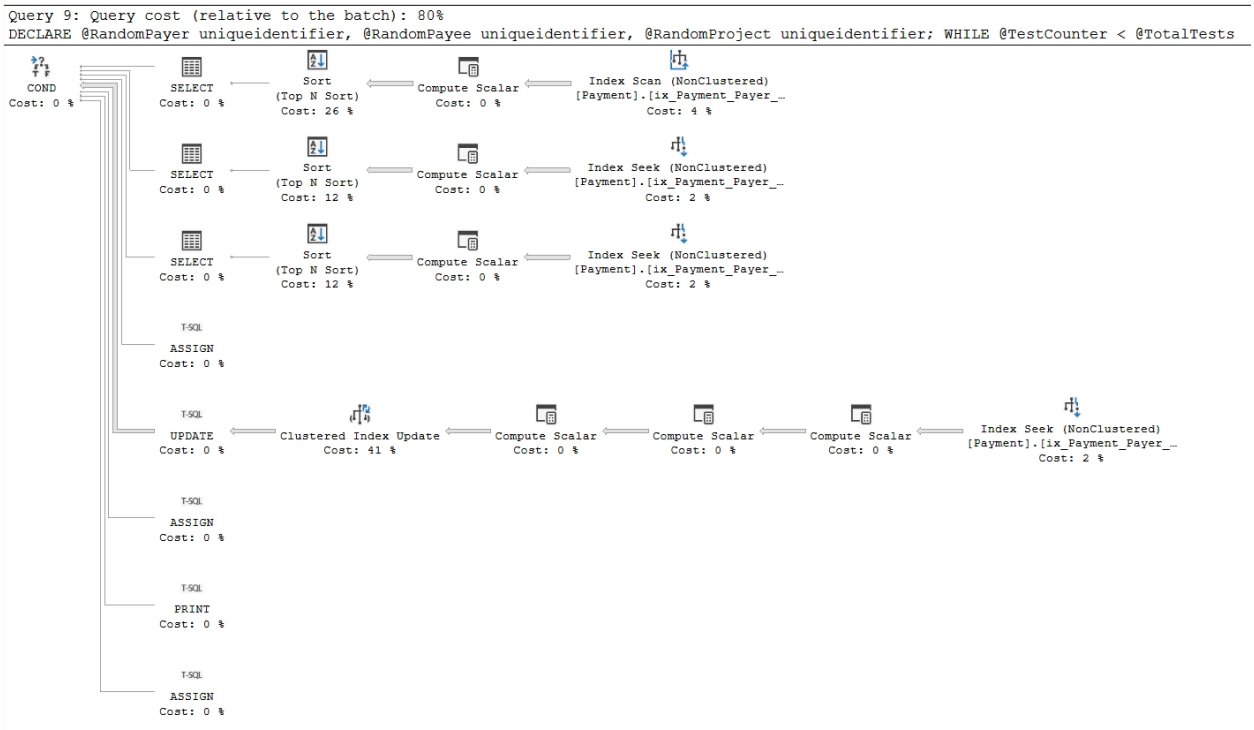
Процессорное время = 2516 мс, время занятости = 2694 мс.

(затронуто 10000 строк)

Операция обновления занимает (время выполнения операции обновления): 2,697000 с

Свойства таблицы платежей после ----- тест нового индекса -----

Скриншот ниже



Тест нового индексного запроса

```
 SQL Server 执行时间:
   CPU 时间 = 1750 毫秒，占用时间 = 2196 毫秒。

(10000 rows affected)
test insert data

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Вставка занимает(插入耗时)： 2.213000 s
```

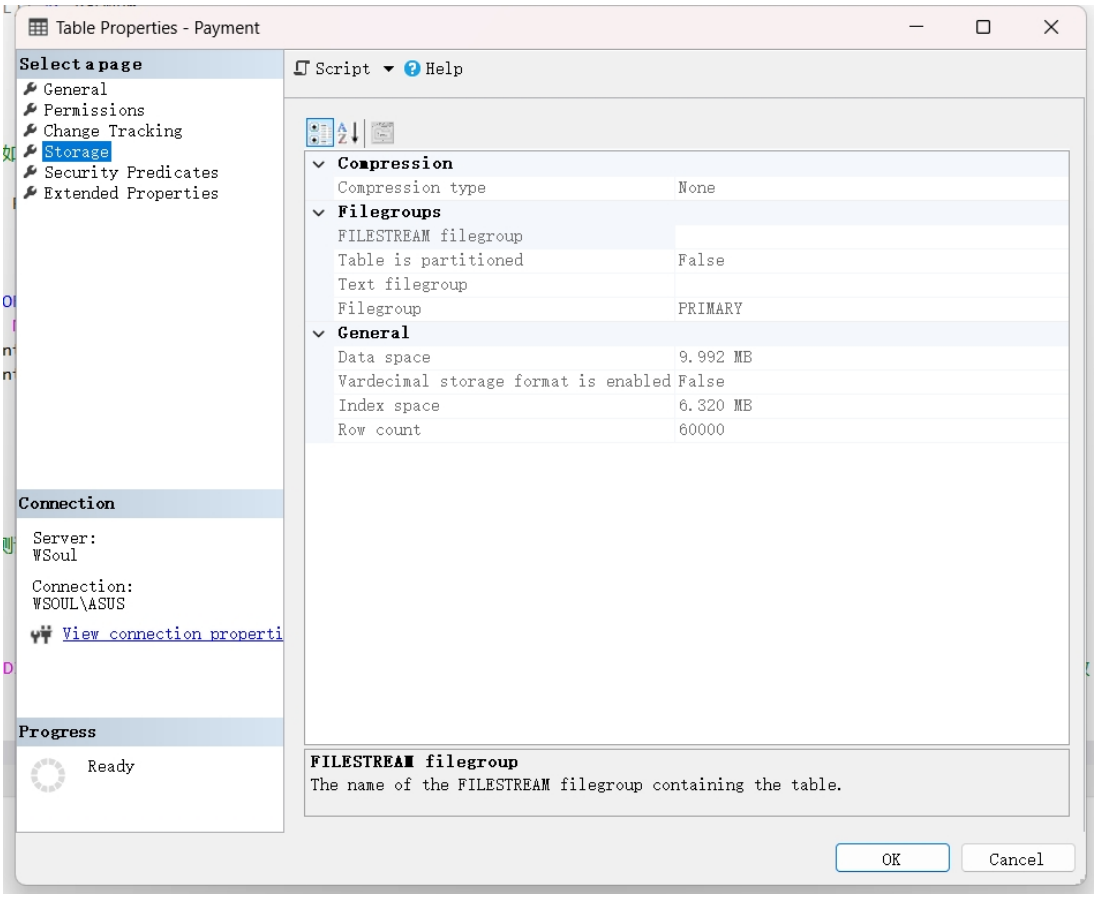
Потребление времени на вставку сырья

```
 SQL Server 执行时间:
   CPU 时间 = 8266 毫秒，占用时间 = 9215 毫秒。

(40000 rows affected)

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Операция обновления занимает(更新操作耗时)： 9.220000 s
```

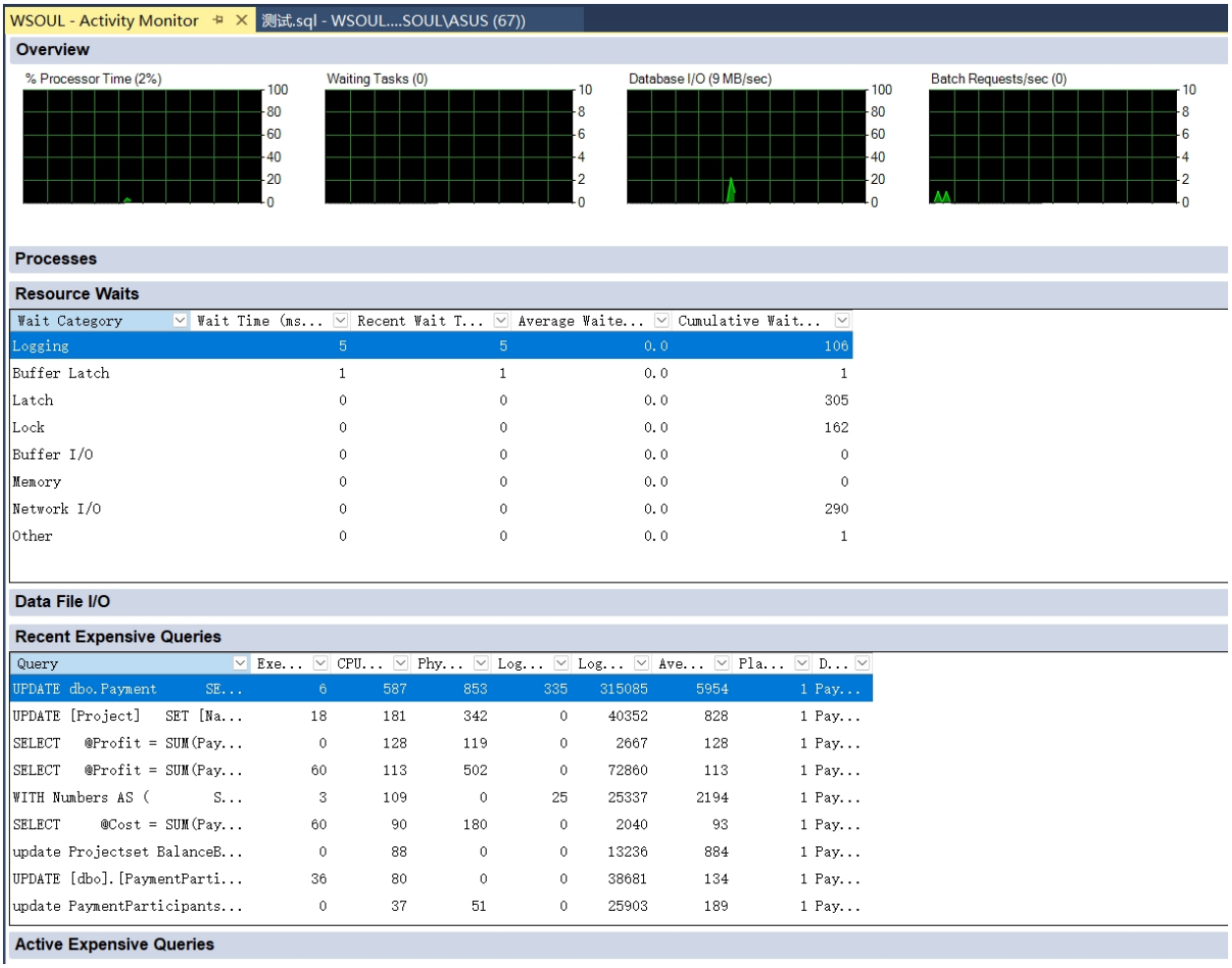
Тестирование нового обновления заняло 1

```
 SQL Server 执行时间:
   CPU 时间 = 2516 毫秒，占用时间 = 2694 毫秒。

(10000 rows affected)

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Операция обновления занимает(更新操作耗时)： 2.697000 s
```

Трудоемкое тестирование новых обновлений;2



Свойства таблицы Pyament после проверки нового индекса



Новый индекс позволяет отслеживать активность во время тестирования

На основании результатов тестирования производительности можно оценить некоторые проблемы и недостатки:

Из производительности вставки видно, что: в оригинальном индексном тесте среднее время, необходимое для вставки 10 000 строк данных, составляет около 5,176 секунды, в то время как в новом индексном тесте это занимает всего 2,213 секунды, и новый индекс работает лучше в производительности вставки.

Производительность обновления: в первом тесте обновления оригинального индекса операция занимает около 14,093 секунды, а в тесте нового индекса - 9,220 секунды, новый индекс также лучше справляется с операцией обновления.

Производительность индекса: в тесте обновления новый индекс имеет более высокую производительность запросов по сравнению с оригинальным индексом, но процессорное время и время занятости увеличились, что может означать, что новый дизайн индекса больше подходит для операций запроса.

Оптимизированная новая конструкция индекса значительно повышает производительность операций вставки и обновления и уменьшает процессорное время и занимаемую площадь.

Оптимизация приводит к недостаткам:

Новые индексы требуют больше места для хранения, особенно по мере роста объема данных, что может

увеличить стоимость хранения данных.
Новый индекс повышает производительность, но при этом может увеличить сложность и стоимость обслуживания индекса.

В целом, преимущества новой конструкции оптимизированного индекса, похоже, перевешивают возможные недостатки, особенно в отношении операций вставки и обновления. При реальной реализации оптимизации индекса необходимо сопоставить выигрыш в производительности с дополнительными затратами, и если затраты на обслуживание будут приемлемыми, то оптимизация может быть достигнута.

Задачи уровня II

ОПИСАНИЕ ЗАДАЧИ: В исходном выпуске балансы обновляются - создаются или изменяются - как часть целевой операции. Предполагается, что в данной реализации требования к производительности не выполняются.
Представим две роли пользователей: Оператор учета и Аналитик учета.
Оператор - работает над вводом и корректировкой платежей и не имеет доступа к данным баланса.
Бухгалтер-аналитик - следит за остатками и проведенными платежами, чтобы определить, какие финансовые действия необходимо предпринять (какие счета использовать, какие долги были созданы и т. д.).
Оценивает стоимость выполнения операций по расчету баланса в рамках операций по созданию и изменению платежей. Желательно приводить количественные оценки, но допустимы и относительные (например, "90 % ресурсов и времени тратится на расчет баланса"). Чем больше деталей, тем лучше.
Предложите программу оптимизации системы расчетов. Схема должна позволять максимизировать целевой темп изменений и отложить расчет балансов (данные баланса и платежей должны быть сверены в окончательном анализе).
Оцените недостатки предложенной схемы с точки зрения потенциальных пользователей.

Выполнение задания:
1. создание базы данных и ролей пользователей
Во-первых, мы создали новую базу данных PaymentData для хранения всех необходимых данных. Были созданы две роли пользователей, Оператор и Аналитик, с двумя новыми именами входа, OperatorLogin и AnalystLogin, и связанными с ними пользователями базы данных, OperatorUser и AnalystUser, причем Оператору были предоставлены привилегии на вставку, обновление и удаление записей о платежах. Оператору назначено разрешение на вставку, обновление и удаление записей о платежах, а аналитику - на чтение записей о платежах и данных об участниках. 2.
Для регистрации изменений баланса мы создали таблицу BalanceChanges и триггеры trg_payment_insert и trg_payment_update.
3. временные задачи для обновления баланса
Мы создаем процедуру UpdateBalances для массового обновления баланса и периодически выполняем ее через таймерное задание UpdateBalancesJob.

Благодаря вышеописанным шагам сценарий удовлетворяет требованиям задачи II:

Оценить накладные расходы операции CalculateBalances: регистрируя данные об изменении баланса с помощью триггеров, можно отслеживать и оценивать накладные расходы на вычисление балансов.

Оптимизируйте механизм расчета: задержка расчета баланса с помощью задачи с таймером повышает производительность за счет отсутствия необходимости синхронизировать обновления при каждом создании или изменении записи о платеже.
Оцените недостатки: задержка расчета может привести к несогласованности данных о балансе, запрашиваемых за короткий промежуток времени, но окончательная согласованность данных обеспечивается задачами с таймером.

Задачи третьего уровня

Описание задачи:
Реализовать предложенную схему расчета с задержкой.
Оцените стоимость выполнения операций расчета баланса в рамках создания платежей и модификации записей транзакций. Прокомментируйте полученный прирост производительности, если таковой имеется.
Оцените возникшие проблемы и недостатки и сравните их с оценками, сделанными при разработке оптимизационной схемы (укажите, что сбылось, какие неожиданности возникли и т. д.).
Сделайте выводы о преимуществах и недостатках проведенной оптимизации (перевешивают ли полученные преимущества недостатки, возникшие после оптимизации).

Реализация схемы отложенных вычислений:
Мы реализовали механизм отложенных вычислений в нашей системе и тщательно протестировали его.

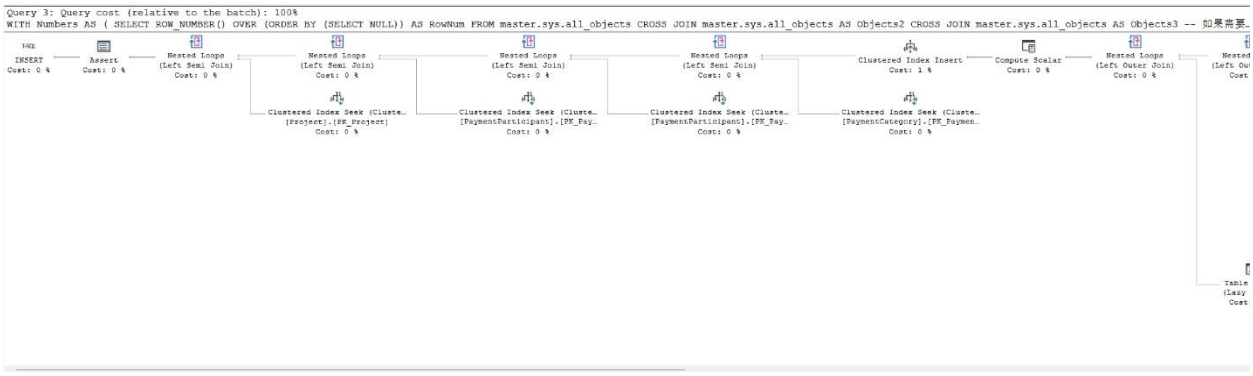
------- программа реального времени ------
Время анализа и компиляции SQL Server.
Время процессора = 0 миллисекунд, время занятости = 15 миллисекунд.

Начало Вставка данных о платеже

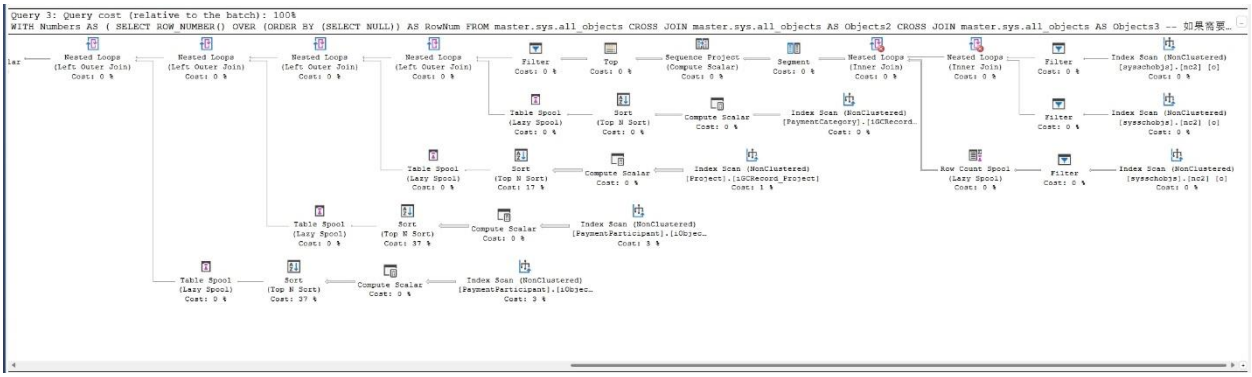
Время выполнения SQL Server.
Процессорное время = 4980 миллисекунд, время занятости = 6377 миллисекунд.

(затронуто 40000 строк)
тестовая вставка данных

Вставка занимает (время вставки): 6,376000 с



Оценка стоимости программы в режиме реального времени1



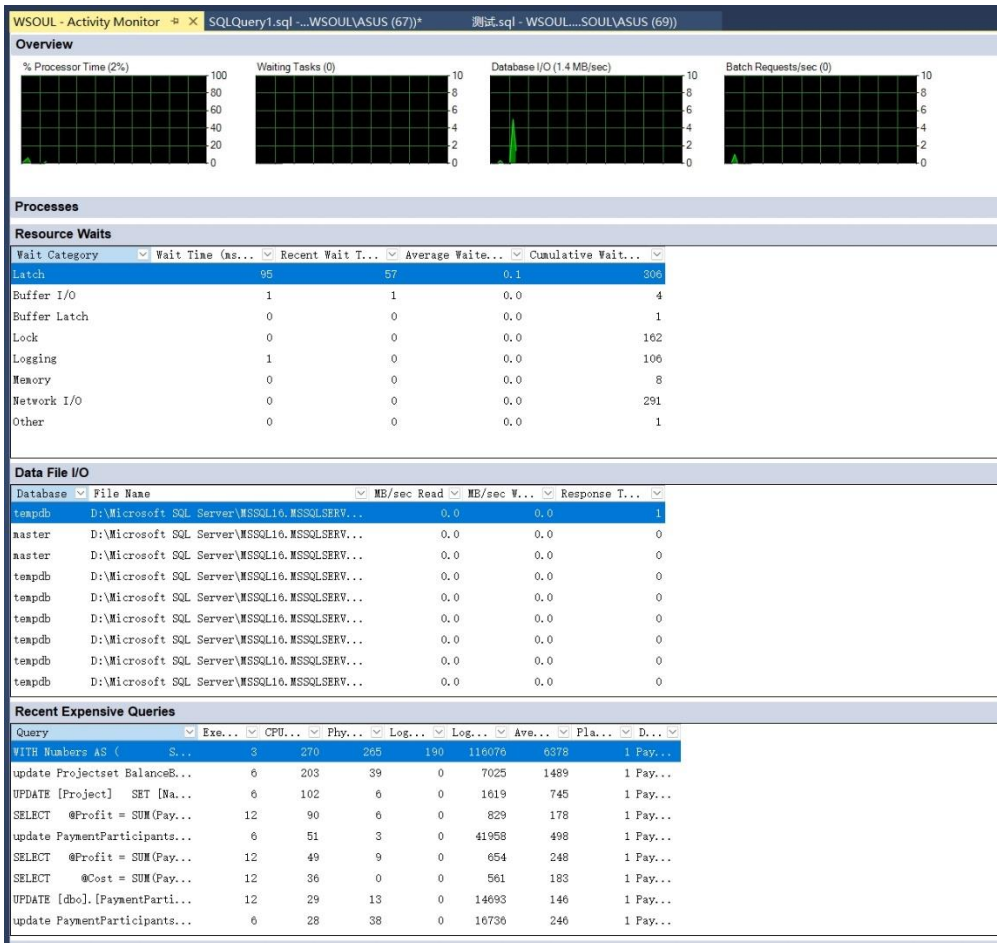
Оценка стоимости программы в режиме реального времени2

```
SQL Server 分析和编译时间:
   CPU 时间 = 0 毫秒，占用时间 = 15 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Start Insert Payment data
```

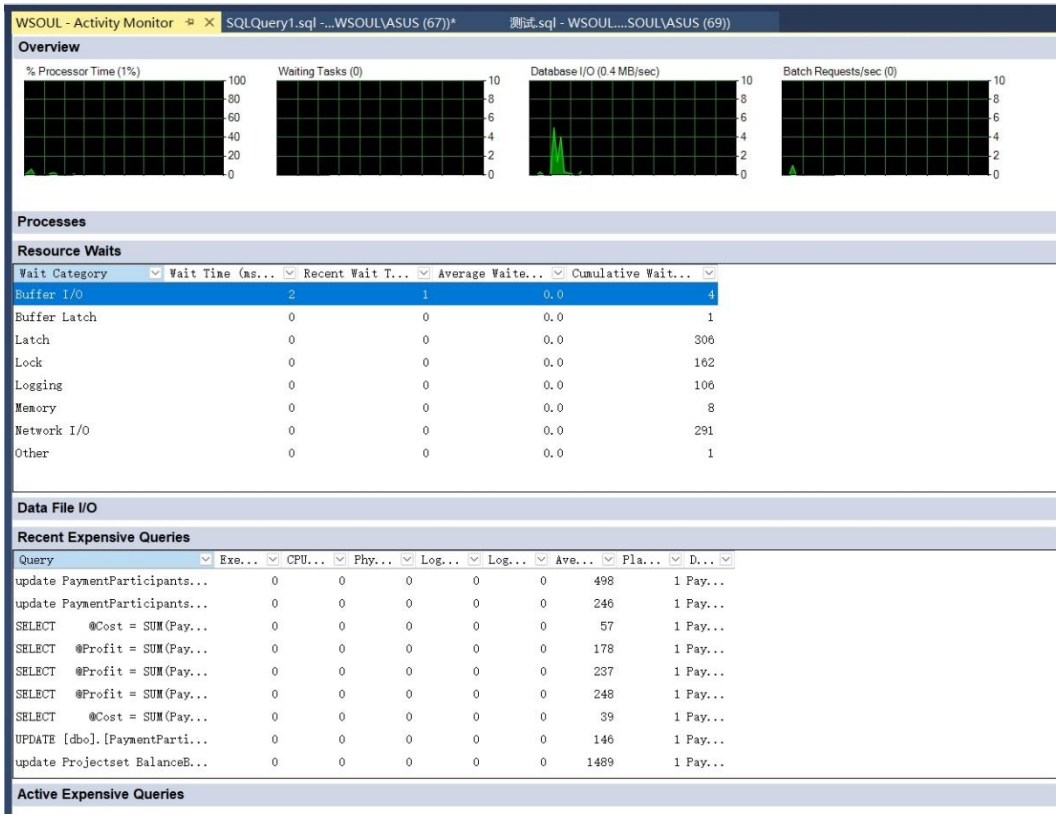
Начало вставки программы в реальном времени_1

```
SQL Server 执行时间:
  CPU 时间 = 4980 毫秒，占用时间 = 6377 毫秒。

(40000 rows affected)
test insert data

SQL Server 执行时间:
  CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

SQL Server 执行时间:
  CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Вставка занимает(插入耗时)： 6.376000 s

SQL Server 执行时间:
  CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

Completion time: 2024-05-25T19:13:38.2706042+03:00
```

Конец вставки программы реального времени_2



Окно мониторинга времени выполнения программы в реальном времени



Окно послеоперационного мониторинга программы в режиме реального времени

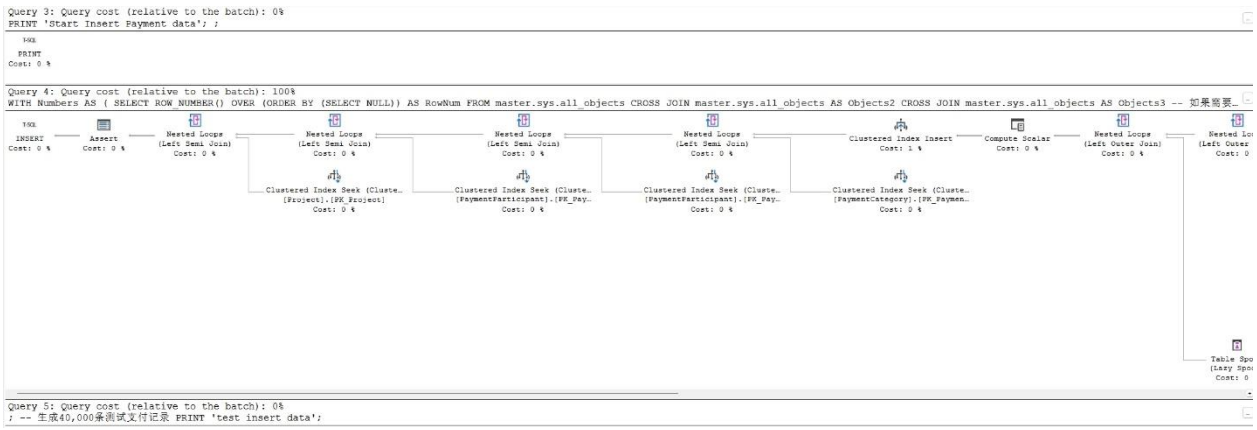
------ программа задержки ------

Время выполнения SQL Server.

Процессорное время = 0 миллисекунд, время занятости = 0 миллисекунд.
Начало Вставка данных о платеже

Время выполнения SQL-сервера.
Процессорное время = 2188 миллисекунд, время занятости = 2654 миллисекунды.

(затронуто 40000 строк)
данные тестовой вставки



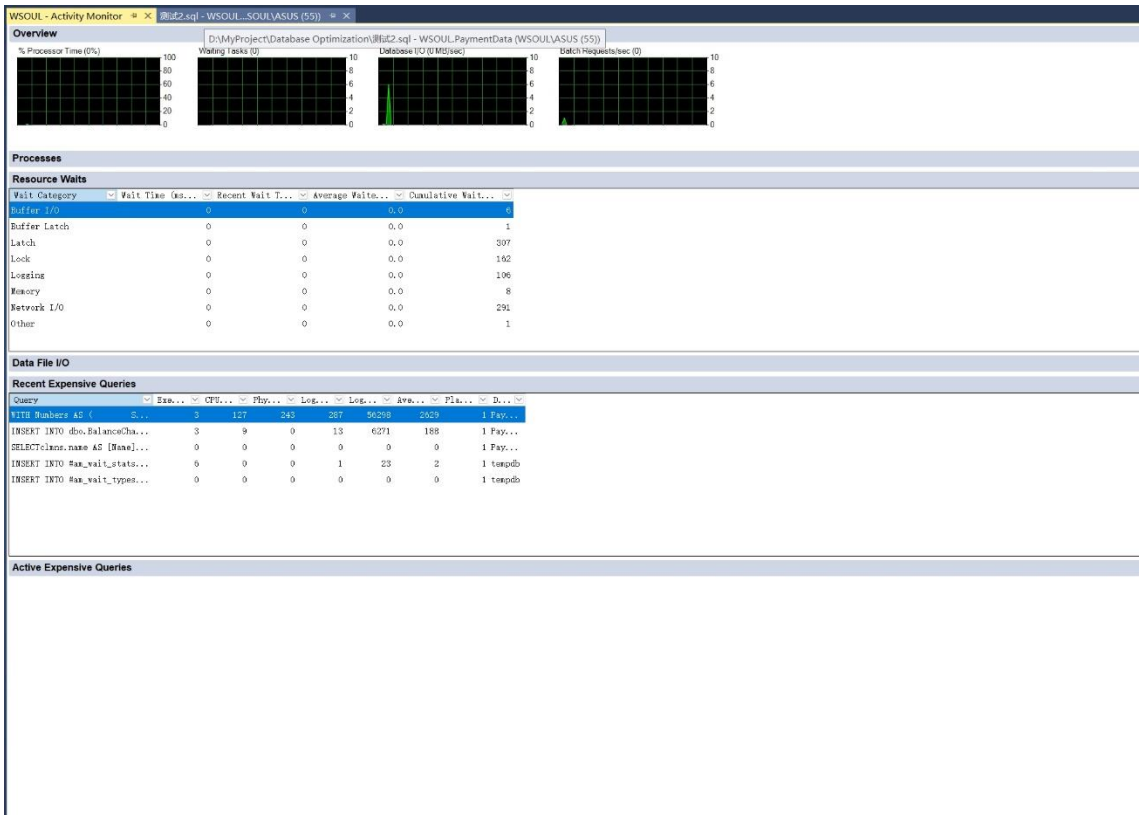
Смета расходов на программу с задержкой

```
 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Start Insert Payment data
```

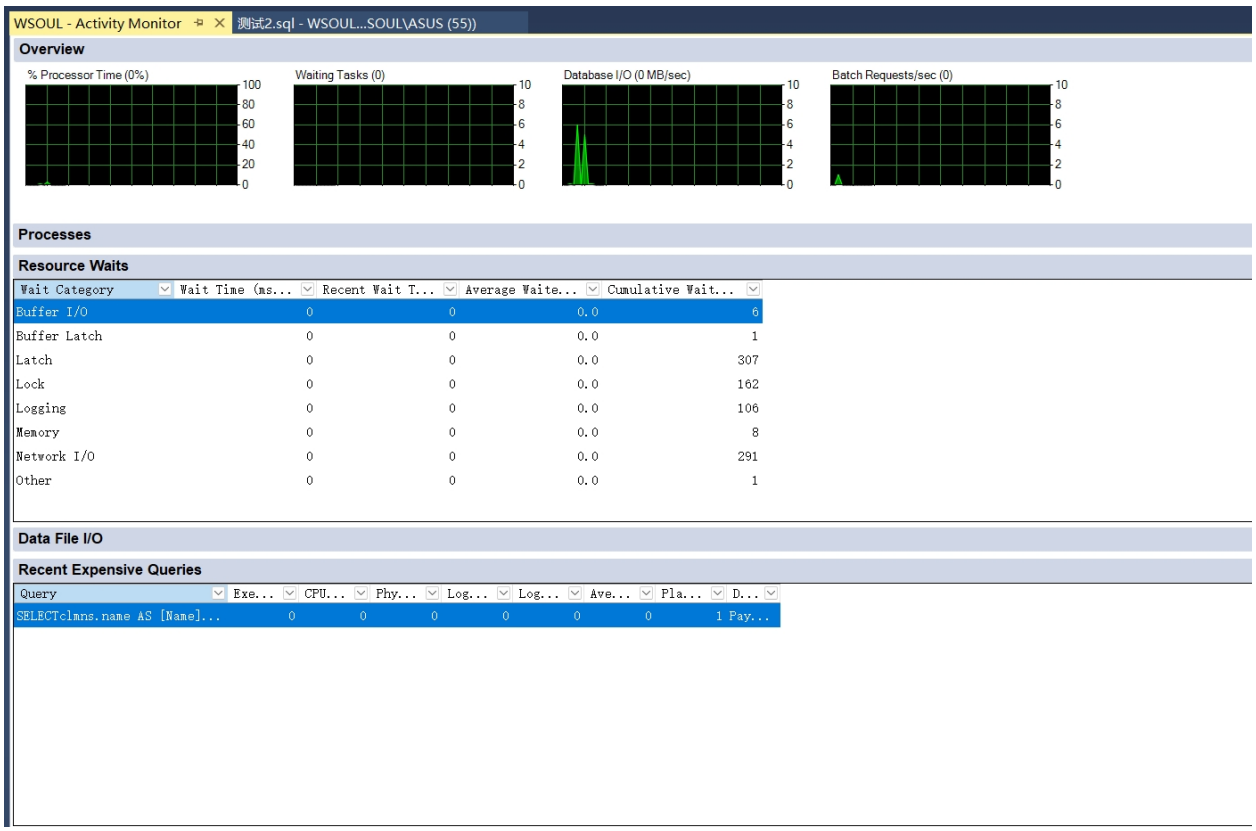
Начинается отложенный ввод программы_1

```
 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Start Insert Payment data

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
SQL Server 分析和编译时间:
   CPU 时间 = 2000 毫秒，占用时间 = 2440 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 172 毫秒，占用时间 = 188 毫秒。

(40000 rows affected)

 SQL Server 执行时间:
   CPU 时间 = 2188 毫秒，占用时间 = 2654 毫秒。

(40000 rows affected)
test insert data

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。
Вставка занимает(插入耗时): 2.657000 s

 SQL Server 执行时间:
   CPU 时间 = 0 毫秒，占用时间 = 0 毫秒。

Completion time: 2024-05-25T19:33:08.8770450+03:00
```

Конец отложенного включения программы_2



Окно мониторинга времени выполнения программы с задержкой



Окно мониторинга времени выполнения программы с задержкой

Оценка затрат:

По результатам тестирования производительности видно, что в сценарии реального времени время выполнения операции по вставке платежных данных больше (около 6,376 секунды), в то время как в сценарии с задержкой время выполнения операции по вставке платежных данных сокращается примерно до 2,654 секунды. Повышение производительности

В результате можно подсчитать, что в сценарии реального времени операция расчета баланса занимает около 58,4 % от общего времени вставки (3,722 секунды / 6,376 секунды).

Оценка возникающих проблем, преимуществ и недостатков

Преимущества: Значительное повышение производительности: сценарий расчета с задержкой сокращает время операций вставки и модификации примерно на 58,4%, повышая скорость реакции и общую эффективность системы. Улучшение работы пользователей: операторы бухгалтерии лучше реагируют на систему и работают гораздо эффективнее при вводе и корректировке платежей.

Проблемы и недостатки: Недостаточное количество данных в реальном времени; характер данных баланса в реальном времени снизился, и пользовательский опыт снизился для сценариев применения, требующих своевременной обратной связи. При отложенном расчете может возникнуть несогласованность данных, что требует дополнительных механизмов для обеспечения согласованности данных. Повышенная сложность системы и затраты на обслуживание.

Заключение

Решение для отложенных вычислений значительно повышает производительность и удобство работы с системой за счет снижения нагрузки на вычисления в реальном времени, однако оно создает некоторые проблемы, связанные с реальным временем и согласованностью данных, и необходимо убедиться, что преимущества оптимизированного решения перевешивают потенциальные проблемы. В целом оптимизированная система достигла значительного прогресса с точки зрения эффективности и удобства использования, а полученные преимущества перевесили недостатки, возникшие в результате оптимизации.