Начальные действия

Для последующей работы с базой данных PaymentData, которая генерируется посредством представленного скрипта, произведено ознакомление с формулами расчета баланса, скриптом создания базы данных, программными компонентами (функции и триггеры). На втором шаге была произведена генерация тестовых данных с помощью программного обеспечения RedGate, которое было выбрано благодаря своим возможностям (тонкая настройка, объем данных, тип данных) и благодаря возможности пользоваться пробным периодом.

В конце концов был разработан тест на корректность расчета баланса.

Задачи I уровня

В данном разделе стоит задача реализации индексов, повышающих производительность операций вставки и изменения платежей без модификации программных компонент. В результате, был разработан код транзакции для операции вставки данных в таблицу dbo.Payment, а также код транзакции для операции изменения данных в таблице dbo.Payment.

В качестве метрики использовалось среднее время, затраченное на исполнение одной транзакции, подсчеты велись с помощью программного обеспечения SQLQueryStress. Было создано подключение к базе данных PaymentData и проведены тесты для операций вставки(100, 400 и 1000), а также для операций изменения(100, 400). Результаты представлены на рисунках ниже.



Рис. 1 Реализация операции вставки данных в таблицу Payment 100 итераций.



Рис. 2 Реализация операции вставки данных в таблицу Payment 400 итераций.



Рис. 3 Реализация операции вставки данных в таблицу Payment 400 итераций.

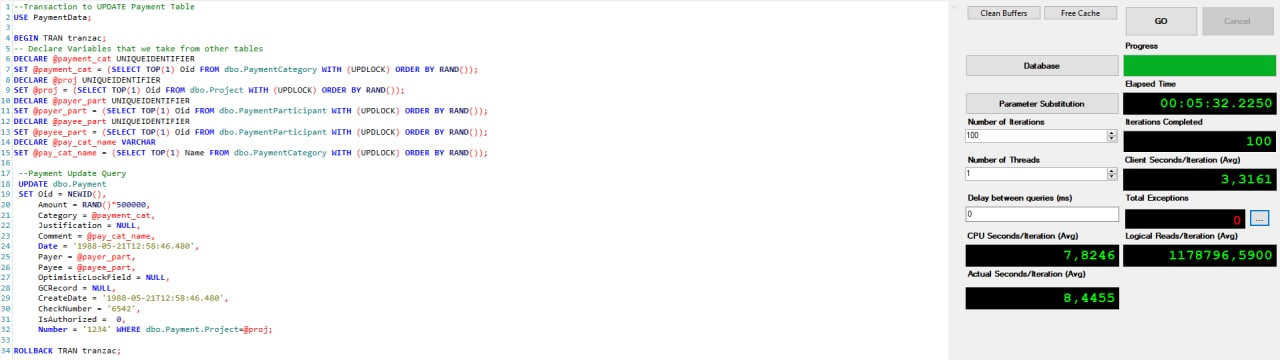


Рис. 4 Реализация операции изменения данных в таблицу Payment 100 итераций.

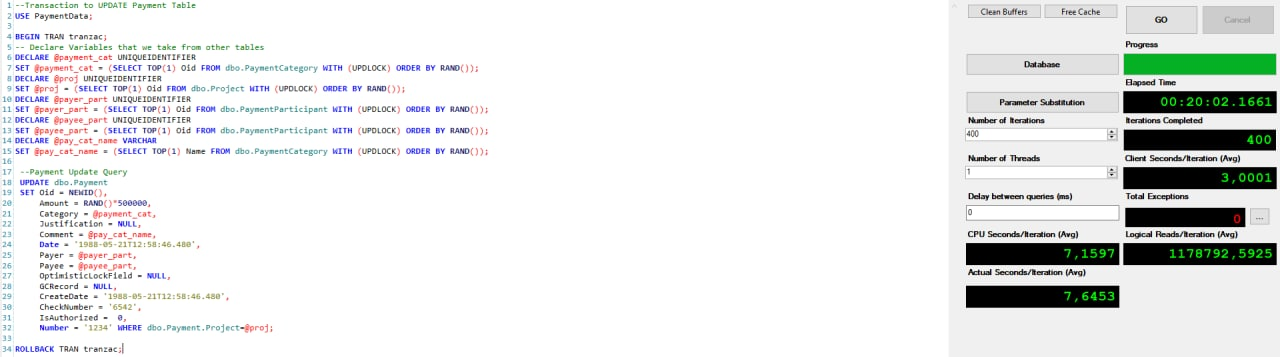


Рис. 5 Реализация операции изменения данных в таблицу Payment 400 итераций.

Далее, для реализации индексов был рассмотрен триггер [T\_Payment\_AI]. Если используются поля, в которых отсутствует индекс, то его, в таком случае, необходимо создать. Если данный триггер срабатывает, то вызываются функции:

* [dbo].[F\_CalculatePaymentParticipantBalance] – необходимо добавить индекс для полей PaymentCategory.NotInPaymentParticipantProfit, PaymentCategory.Name, AccountType.Name;
* [dbo].[F\_CalculateBalanceByMaterial] – необходимо добавить индекс для полей Supplier.ProfitByMaterialAsPayer, Supplier.ProfitByMaterialAsPayee, AccountType.Name, PaymentCategory.Name, PaymentCategory.ProfitByMaterial, PaymentCategory.CostByMaterial;
* [dbo].[F\_CalculateBalanceByWork] – необходимо добавить индекс для полей AccountType.Name, PaymentCategory.Name;
* [dbo].[F\_CalculateProjectBalance] – необходимо добавить индекс для полей PaymentCategory.Name;

В результате, были создать следующие индексы: PCProfByMatInd, PCCostByMatInd, PCNPayPartInd, PCNameInd, ATNameInd, SuppProfByMatPayerInd, SuppProfByMatPayeeInd. Результат представлен на рисунке ниже.

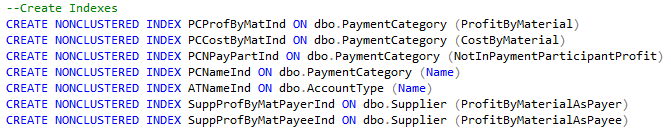


Рис. 6. Создание некластеризованных индексов.

Далее были заново проведены тесты для операций вставки данных в таблицу и изменений данных в таблице. Результаты представлены на рисунках ниже.

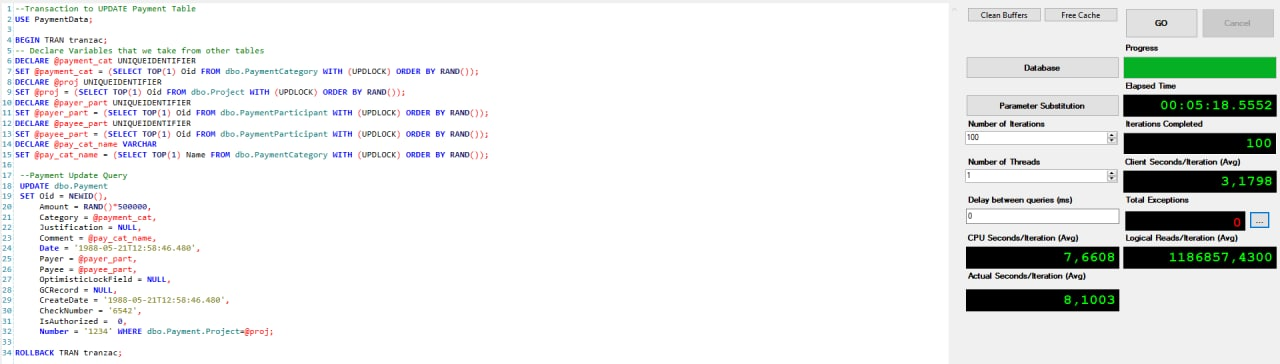


Рис. 7 Реализация операции изменения данных в таблицу Payment 100 итераций после добавления некластеризованных индексов.

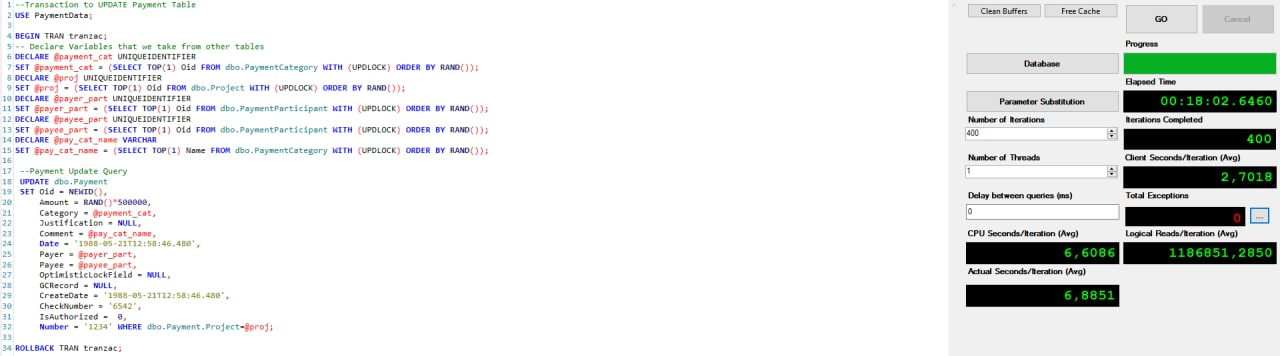


Рис. 8 Реализация операции изменения данных в таблицу Payment 400 итераций после добавления некластеризованных индексов.



Рис. 8 Реализация операции вставки данных в таблицу Payment 100 итераций после добавления некластеризованных индексов.



Рис. 8 Реализация операции вставки данных в таблицу Payment 400 итераций после добавления некластеризованных индексов.



Рис. 8 Реализация операции вставки данных в таблицу Payment 1000 итераций после добавления некластеризованных индексов.

Более детально результаты среднего времени на исполнение одной транзакции представлены в таблице ниже.

|  |  |  |
| --- | --- | --- |
| Название, количество итераций | Время перед добавлением индексов, с | Время после добавления индексов, с |
| Операция вставки |  |  |
| 100 | 1,3778 | 1,4129 |
| 400 | 1,2484 | 1,0496 |
| 1000 | 1,3476 | 1,0024 |
| Операция изменения |  |  |
| 100 | 8,4455 | 8,1003 |
| 400 | 7,6453 | 6,8851 |

В результате, можно заметить, что при меньшем количестве операций операция вставки работает дольше, время увеличилось на 2.5%, возможно, данный феномен связан с техническим оснащением. Далее, при 400 итерациях прирост скорости операции вставки составляет около 16%, а при 1000 итерациях около 26%. Если говорить об операции изменения данных, то при 100 итерациях прирост составляет около 4%, а при 400 итерациях около 10%.