Задание уровня 2. Цепелева Рита.

Пункт 1.

В первую очередь, для того, чтобы выполнить задание 2.1, я пользовалась Azure Data Studio. С помощью данного ПО я смогла получить план запроса на INSERT и планы вложенных запросов, а также некоторую статистику, указав в начале sql-скрипта SET STATISTICS XML ON.

Результатом работы данного скрипта стал вывод нескольких таблиц (а именно, 13 таблиц), строками которой являются соответствующие операции из дерева операций нашего основного запроса INSERT и его подзапросов. В этих таблицах для меня самыми интересными были колонки Estimated I/O Cost и Estimated CPU Cost (на их основе делалась количественная оценка). Кроме того, данные из этих таблиц служат основой для плана запроса (графическое представление выполнения запроса).

Итак, на основе анализа триггера [T_Payment_AI], который я провела на подготовительном этапе и на этапе выполнения задания уровня 1, я знаю, что при изменении/добавления платежа он срабатывает и в теле данного триггера содержатся следующие операции:

- вычисление баланса inserted.Payer
- обновление баланса inserted.Payer
- вычисление баланса inserted.Payee
- обновление баланса inserted.Payee
- вычисление баланса deleted.Payer
- обновление баланса deleted.Payer
- вычисление баланса deleted.Payee
- обновление баланса deleted.Payee
- вычисление баланса inserted.Project
- обновление баланса inserted.Project
- вычисление баланса deleted.Project
- обновление баланса deleted.Project

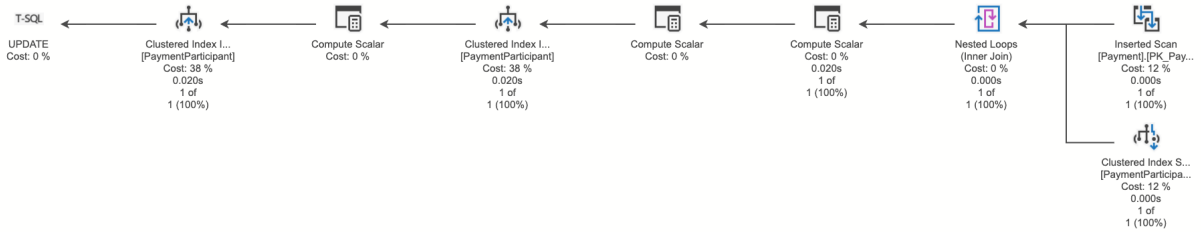
Если ко всем этим операциям добавить непосредственно внесение платежа в [dbo].[Payment], получатся те самые 13 операций, для каждой из которых у нас есть таблица с данными.

Для того, чтобы сократить себе работу и не рассматривать каждые из 13 операций, я решила объединить их в смысловые группы. Во-первых, мы видим, что есть 2 типа операций: вычисление и обновление. Во-вторых, сначала в триггере ведется вычисление/обновление над Payment Participants, а в конце над проектами. Таким образом, группы я составила следующие:

- payment participants calculate
- payment participants update
- project calculate
- project update
- insert operation

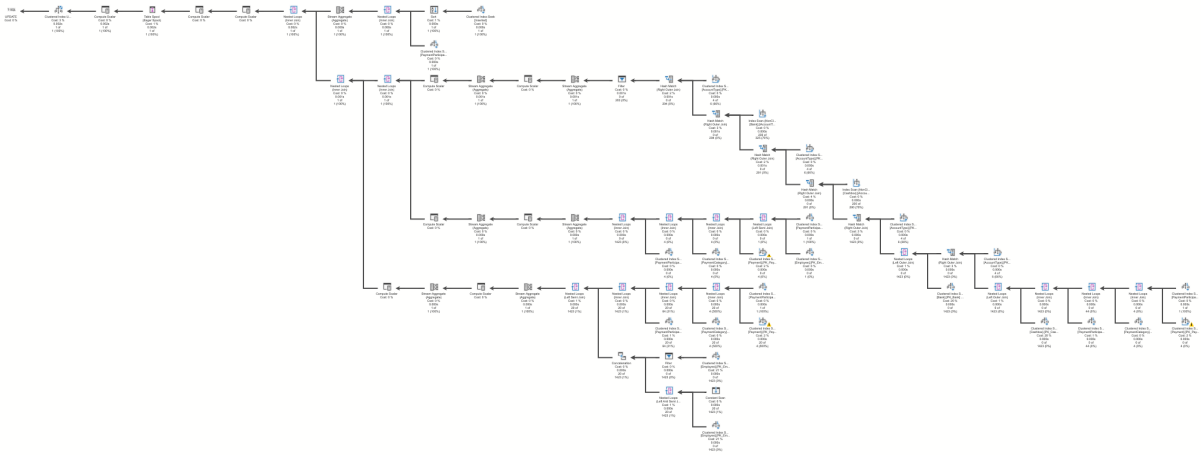
Для каждой из групп я сохранила план запроса (см папку plans), рассмотрим подробнее:

- payment participants calculate



Здесь самыми затратными операциями являются Clustered Index Insert (Cost 38%).

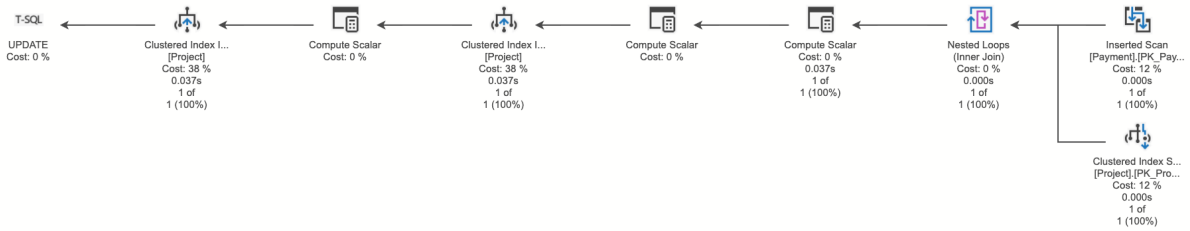
- payment participants update



Здесь план выполнения конечно очень большой и вряд ли удастся что-то увидеть именно на скриншоте, поэтому если интересно, то план можно посмотреть в plans/updPaymParticipant.sqlplan.

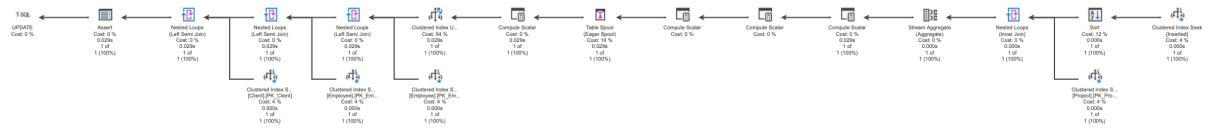
Но здесь я прокомментирую: самые затратные операции при обновлении балансов участников платежей располагаются в самом начале, и это Clustered Index Seek(20-21%)

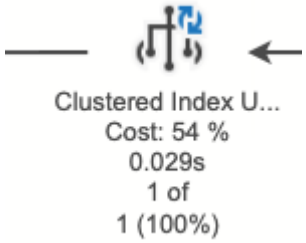
- project calculate



Аналогично с участниками платежей, самыми затратными операциями здесь являются Clustered Index Insert(38%).

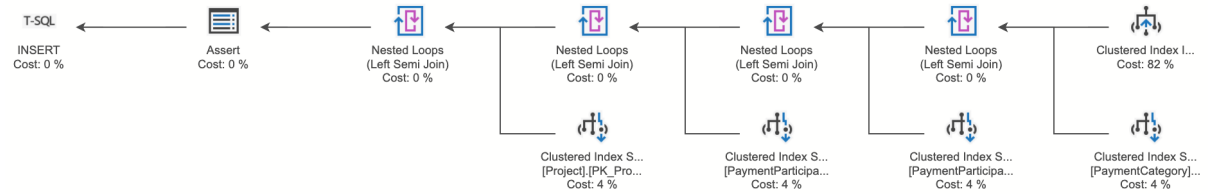
- project update





Самой затратной операцией здесь является Clustered Index Update(54%).

- insert payment



В самой операции добавления самой затратной операцией является Clustered Index Insert(82%).

Также в планах выполнения запроса можно наблюдать время, затраченное как на выполнение отдельных блоков, так и на выполнение в целом.
На основе данной информации я составила относительную оценку затрат по времени на исполнение ОДНОЙ операции каждой группы операций, выделенных ранее:

- payment participants calculate = 1,5%
- payment participants update = 6%
- project calculate = 11%
- project update = 23,3%
- insert operation = 1%

Пункт 2.

В изначальной реализации операции INSERT/UPDATE над базой [dbo].[Payment] влекут за собой изменение балансов участников и проектов. В задаче уровня 2 предлагается ввести новые роли: оператора и бухгалтера. Оператор будет отвечать за внесение/корректировку данных, бухгалтер - за проведение анализа всего скоупа полученных данных от оператора.
Таким образом, очевидным предложением (или выводом), на мой взгляд, в данной задаче является предложение копить данные, а только потом их обрабатывать, так как, исходя из выполненной ранее оценки затрат, один запрос на добавление пачки записей требует меньше времени, чем выполнение множества запросов на добавление отдельных записей.
Копить записи на добавление/изменение платежей от оператора можно по-разному. Для этого, например, можно создать отдельную таблицу, которая будет выступать в роли буфера, в котором копятся записи и, например, при достижении определенного

количества записей должен включаться в работу бухгалтер, а буфер должен очищаться и снова копить данные (это можно сделать при помощи триггера или например при помощи скрипта на python, который будет отслеживать время от времени кол-во данных в таблице и при достижении порога очищать буфер и запускать работу бухгалтера).
Также стоит рассмотреть внедрение в систему режима сессий: пока не окончена сессия оператора, буфер получает от него данные и копит их. Как только оператор заканчивает свою сессию, бухгалтер начинает свою сессию и работает с данными, накопленными во время предыдущей сессии оператора.

Пункт 3.
Конечно, недостатком со стороны пользователя любой системы отложенных платежей будет являться неактульность баланса в определенные промежутки времени.
Но если рассматривать предложенные варианты и сравнивать эти предложения между собой, то можно выделить следующие пункты:

- например, если оценить реализацию очищения буфера при достижении порога, то я предлагала либо написать триггер, либо сторонний скрипт. И с одной стороны, скрипт будет являться лучшим решением, тк триггер может привести к взаимоблокировке. С другой стороны, если писать триггер, то можно обойтись без сторонних компонент и сохранить целостность решения;
- также если сравнить сценарий с порогом и режим сессий, то явно внедрение режима сессий будет являться лучшим решением по нескольким причинам:
    - не нужно будет подбирать порог (на мой взгляд, подбор оптимального порога может занять длительное время)
    - так как данные могут пополняться/изменяться в произвольное время, порог может не набираться долгое время или наоборот в какие-то промежутки времени набираться слишком быстро. Таким образом либо будет наблюдаться застой и пользователи слишком долго не смогут увидеть актуальный баланс, либо в моменты активного добавления данных нагрузка на сервер с базой будет очень высока. В этом плане режим сессий поможет наладить более гибкую работу: в моменты застоя не дожидаться достижения порога, а по факту добавления данных делать апдейт, либо в моменты высокой нагрузки оператор не будет завершать сессию до тех пор, пока идут запросы, при снижении нагрузки завершать сессию и отдавать данные бухгалтеру.

Также хотелось бы предложить выводить пользователям время, прошедшее с момента последнего апдейта (по аналогии с сайтом приемной комиссии, где ранжированные списки обновлялись каждые 30мин и вверху отображалось время последнего обновления). Это позволит пользователям самим судить об актуальности данных.