1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 5**

1. **«Оптимизация запросов»**
2. по дисциплине «Системы управления базами данных»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. доцент, к.т.н. Полтавцева М.А.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цель**

Получение навыков анализа производительности реляционной базы  
данных и оценки влияние методов оптимизации на производительность базы  
данных.

**Задачи**

1. Заполните созданную ранее базу данных не менее чем 10 000 записями  
в каждом отношении, *задействованном в запросах* (возможно  
применение автоматической генерации).  
a) Число тестовых записей может быть увеличено или уменьшено в  
зависимости от характеристик аппаратного обеспечения  
(*ориентируйтесь на время выполнения одного запроса более 2-3  
секунд*).  
b) ***Селективность*** свойственная предметной области должна быть  
относительно соблюдена в отношении заполняемых данных.  
2. Ознакомьтесь с инструментарием, который ваша СУБД предоставляет  
для отслеживания состояния транзакций и параметров выполнения  
запросов. Ознакомьтесь с инструментарием просмотра планов  
выполнения запросов, определите какие типы отображения планов  
доступны.  
3. Для каждого запроса, созданного к данным ранее согласно варианту:  
a) Составьте логический план в соответствии с правилами  
логической оптимизации  
b) Определите и приведите физический план запроса при помощи  
инструментов СУБД,  
⎯ в графической форме (если это возможно)  
⎯ **и** текстовой или XML форме.  
Если использовано только графическое представление плана, приведите  
после каждого плана таблицу с указанием стоимостей выполнения  
операций.  
c) Если возможно построение предварительного и реального планов  
выполнения, приведите оба плана, **в случае, если они  
различаются**.  
d) Зафиксируйте время выполнения запроса  
**e)** Соотнесите физический план запроса с реляционным  
представлением запроса и логическим планом. **Сделайте выводы.**4. Определите «узкие места» при выполнении запроса, воспользовавшись  
инструментами СУБД. При необходимости увеличьте число тестовых  
записей.  
5. Создайте необходимые индексы исходя из запросов **и характеристик  
данных**.  
6. Для каждого запроса:  
a) Получите новый план запроса, представив его в графической **или**краткой текстовой (XML) форме.  
b) Зафиксируйте новое время выполнения каждого запроса.  
c) Укажите, какие индексы используются при выполнении плана.  
Если индекс не использован, определите причину его  
неэффективности.  
d) Сравните эффективность исходной записи, первого и второго  
планов на запросе, сделайте выводы.  
7. Если это возможно, преобразуйте запросы для большей эффективности  
выполнения, в том числе, преобразовав логические операции. Получите  
новый план и сравните его показатели с предыдущими.  
8. Сделайте выводы.

**Ход работы**

Были проведены оптимизации запросов из 2-ой лабораторной работы.

**Первый запрос**

Вывести всех обучающихся и указать оценку по курсу «Литература» для всех, кто его проходил в формате: ФИО, Оценка

|  |
| --- |
| SELECT T1.FIO, T2.ItogovayaOcenka FROM  (SELECT DISTINCT (SELECT Chelovek.FIO FROM Chelovek WHERE Chelovek.NumerPassporta = ObuchayushchijsyaNumerPassporta) FIO FROM Ocenka) T1  LEFT JOIN    (SELECT (SELECT Chelovek.FIO FROM Chelovek WHERE Chelovek.NumerPassporta = ObuchayushchijsyaNumerPassporta) FIO ,ItogovayaOcenka FROM Ocenka WHERE ZHurnal\_OcenokKursNazvanie='Литература') T2  ON T2.FIO=T1.FIO; |

Запрос в РА:

При

k=

Разработанный логический план (без оптимизаций):

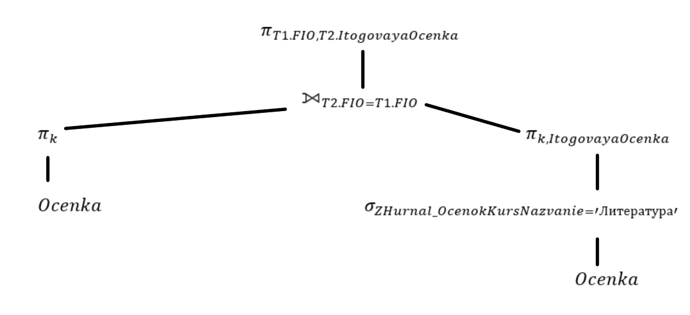


Рисунок 1 – Логический план первого запроса (без оптимизаций).

На рисунке 2 изображён план выполнения запроса, построенный MS SQL. Предварительный и реальный планы совпадают.

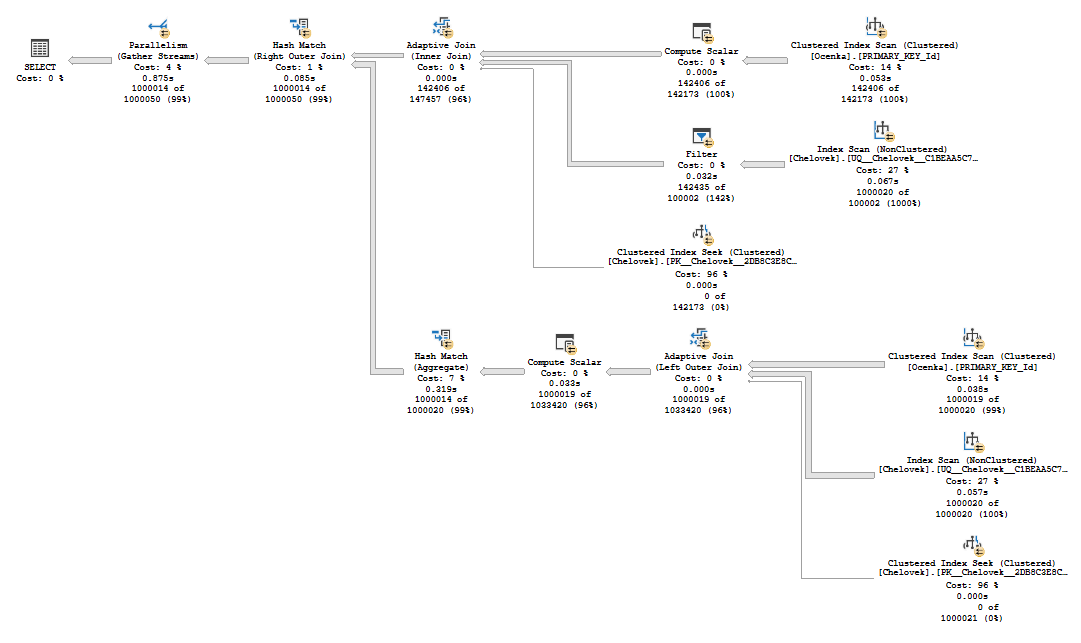


Рисунок 2 – Физический план выполнения запроса.

Этот же план доступен в XML-виде, однако слишком большой и нечитаемый при помещении в отчёт. Он, как и другие планы, будут прикреплены к отчёту отдельными документами.

По этому плану видно, что нет баланса, большой перекос в выполняемых операциях. Присутствуют операция с высокой стоимостью и широкой стрелкой (много данных – узкое место).

В таблице 1 приведено время выполнения запроса для 5 попыток и среднее по всем попыткам.

|  |  |  |  |
| --- | --- | --- | --- |
| Неоптимизированный запрос | | Оптимизированный запрос | |
| Без индексов | Индекс на столбцы FIO и ObuchayushchijsyaNumerPassporta | Без индексов | Индексы на ZHurnal\_OcenokKursNazvanie,  ItogovayaOcenka,  ObuchayushchijsyaNumerPassporta |
| 6015 | 6666 | 5860 | 5680 |
| 5921 | 6081 | 5835 | 5756 |
| 5953 | 6124 | 5871 | 5601 |
| 5859 | 6015 | 5841 | 5729 |
| 5934 | 6140 | 5821 | 5686 |
| 5936.4 | 6205.2 | 5845.6 | 5690.4 |

Таблица 1. Время выполнения первого запроса в мс.

Индексы в неоптимизированном запросе неэффективны, так как результат подзапроса, в отличие от таблиц, никаких индексов не имеет, поэтому их применение бессмысленно.

Полученный физический план отличается от составленного логического плана, так как, видимо, полагается на дополнительные операции и **изменение толщины стрелок**.

Например,

* Изменение толщины стрелок. Ширина стрелки пропорциональна количеству строк. Больше количество строк – шире стрелка - больше данных между этими операторами.
* **Clustered Index Seek** - означает просмотр индекса в порядке упорядочивания, по B-дереву. Представляет собой прямой доступ SQL Server к требуемым строкам данных. То есть вытаскивает строки из кластеризованного индекса.
* **Index Scan** - обычная операция просмотра всех записей таблицы, аналогичная Table Scan. То есть производится сканирование некластеризованного индекса. Обычно наличие этой операции плохо отражается на производительности, поскольку она предполагает последовательное чтение индекса для извлечения большого числа строк, приводя к более медленной обработке.
* **Оператор Compute Scalar** вычисляет выражение и выдает вычисляемую скалярную величину. Затем эту величину можно вернуть пользователю или сослаться на нее в каком-либо запросе, а также выполнить эти действия одновременно.
* **Filter** - оператор просматривает входные данные и возвращает только те строки, которые удовлетворяют критерию фильтрации (предикату)
* **Parallelism** - Операторы параллелизма обычно считаются хорошими вещами: SQL Server дробит данные на множество частей для асинхронной обработки на множестве процессоров, сокращая общее время работы, требуемое для выполнения вашего запроса. Однако параллелизм может стать плохим, если большинство запросов используют его.
* **Adaptive Join** – он выбирает оператор соединения: Hash Match или Nested Loop
* **Hash Match** - создает хеш-таблицу (в памяти) для требуемых столбцов для каждой строки, затем создает хеш-таблицу для второй таблицы и ищет совпадения по каждой строке. Это очень дорогая операция и требует много памяти.
* **Nested Loop** - выполняет поиск во внутренней (меньшей) таблице для каждой строки внешней (большей) таблицы. Операция менее дорогая, чем Hash Match, и идеально подходит для малого числа строк на входе. Это наиболее быстрый оператор соединения, который требует наименьшего количества операций ввода/вывода с наименьшим количеством строк для сравнения.

Таким образом, физический план даёт более ясное представление о действиях сервера.

SQL-код (оптимизированный)

|  |
| --- |
| SELECT DISTINCT FIO, ItogovayaOcenka  from Chelovek  left join Ocenka on NumerPassporta = ObuchayushchijsyaNumerPassporta  and ZHurnal\_OcenokKursNazvanie=N'Литература' |

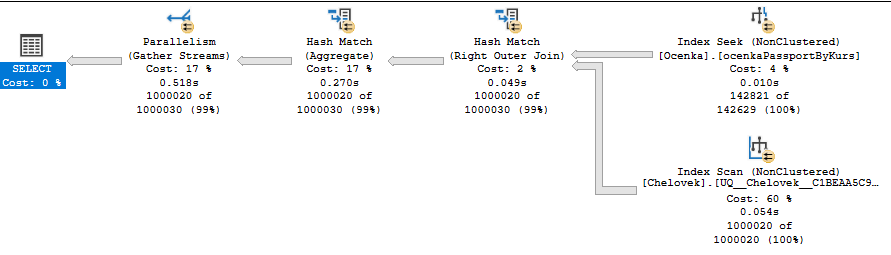


Рисунок 3 – Оптимизированный запрос и его физический план.

План запроса стал проще, исчезли многие ветви (рисунок 3).

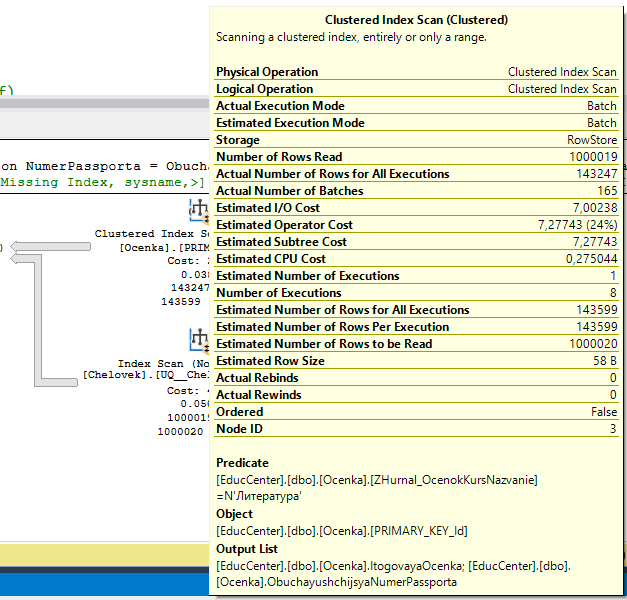


Рисунок 4 – указание использованных индексов.

Оптимизация заключалась в следующем - добавить индекс под поиск по ZHurnal\_OcenokKursNazvanie и выборку ItogovayaOcenka, ObuchayushchijsyaNumerPassporta - его и подсказывает SSMS:

|  |
| --- |
| CREATE NONCLUSTERED INDEX ocenkaPassportByKurs  ON [dbo].[Ocenka] ([ZHurnal\_OcenokKursNazvanie])  INCLUDE ([ItogovayaOcenka],[ObuchayushchijsyaNumerPassporta]) |

Итого: оптимизация хоть и сильно уменьшила размер физического плана, но выигрыш составил ~300 мс, или 5%. Узким местом оказывается Filter, оптимизации его и затрагивают.

**Второй запрос**

Вывести список ФИО преподавателей, которые никогда не выставляли ни одной оценки ниже 3.

|  |
| --- |
| SELECT Chelovek.FIO FROM Chelovek  JOIN  (SELECT P.NumerPassporta FROM Prepodavatel AS P  JOIN --даёт из названий курсов номера пасппаротов  (SELECT b.ZHurnal\_OcenokKursNazvanie FROM Ocenka b -- все курсы  EXCEPT -- минус  SELECT a.ZHurnal\_OcenokKursNazvanie FROM Ocenka a WHERE a.PromezhutochnyeOcenka1<=3 OR a.PromezhutochnyeOcenka2<=3 OR a.PromezhutochnyeOcenka3<=3 OR  a.ItogovayaOcenka<=3) Raznost -- таблица, где хотя бы 1 тройка  ON P.RaspisanieKursNazvanie=Raznost.ZHurnal\_OcenokKursNazvanie) Pass  ON Pass.NumerPassporta=Chelovek.NumerPassporta; |

Запрос в РА:

RAZNOST= A – B

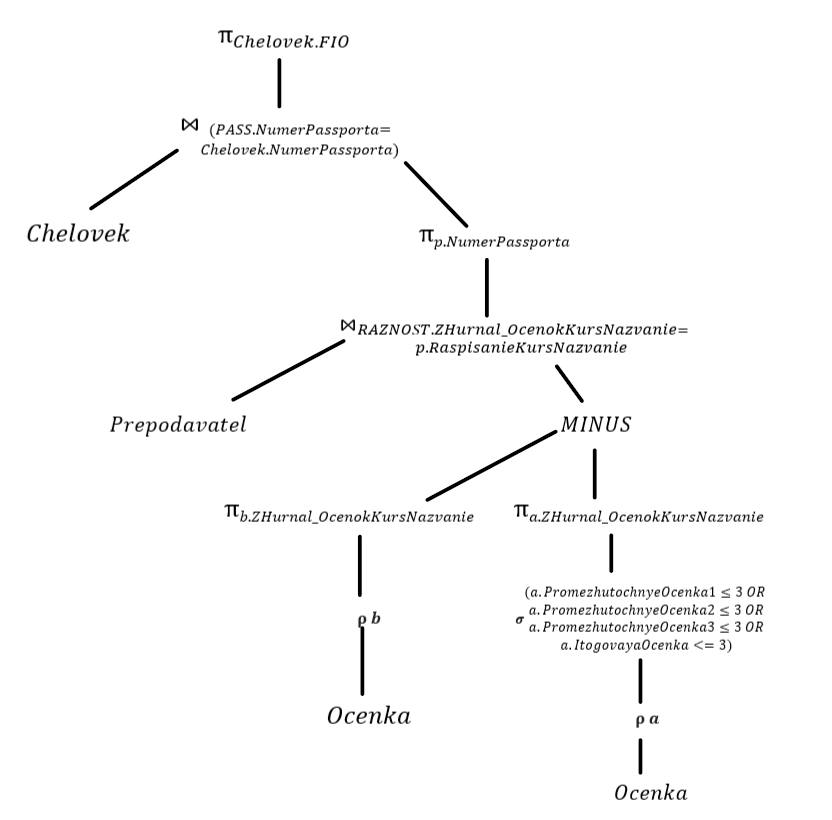


Рисунок 5 – логический план второго запроса.

Предварительный и реальный планы совпадают.

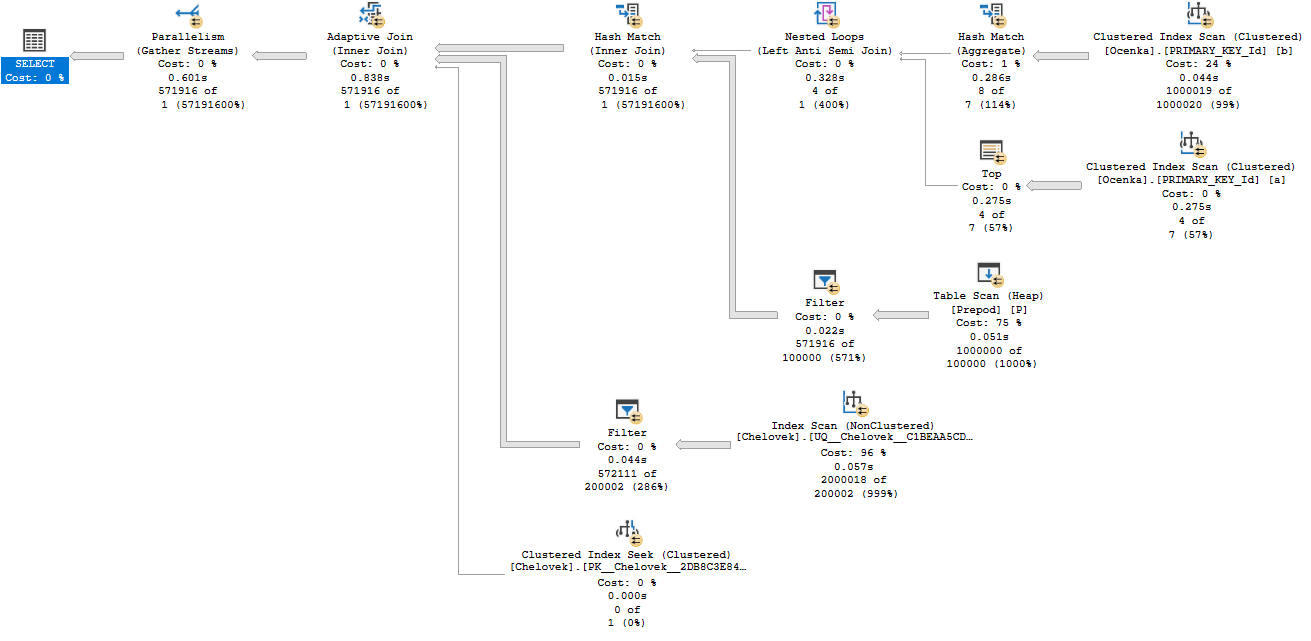


Рисунок 6 – реальный план выполнения запроса 2.

В таблице 2 указано время запроса с индексами и без на оригинальный запрос из 2 лабораторной.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Неоптимизированный запрос | | | | | |
| Без индексов | Индекс на FIO | Индекс на Prepod.  NumerPassporta | Индекс на Prepod.  NumerPassporta и Prepod. RaspisanieKursNazvanie | Индекс на  Ocenka.  ZHurnal\_OcenokKursNazvanie | Индексы на все столбцы |
| 3618 | 3627 | 3735 | 3447 | 3568 | 3449 |
| 3991 | 3536 | 3584 | 3478 | 3806 | 3450 |
| 3485 | 3698 | 3646 | 3448 | 3560 | 3406 |
| 3469 | 3698 | 3658 | 3485 | 3506 | 3375 |
| 3783 | 3546 | 3590 | 3370 | 3770 | 3417 |
| 3669.2 | 3621 | 3642.6 | 3445.6 | 3642 | 3419.4 |

Таблица 2 – неоптимизированный запрос и применение на него индексов.

Лучшее время неоптимизированного запроса с индексами было показано при применении всех индексов или при использовании индексов Prepod.

NumerPassporta и Prepod. RaspisanieKursNazvanie. Рассмотрим план неоптимизированного запроса с индексами:

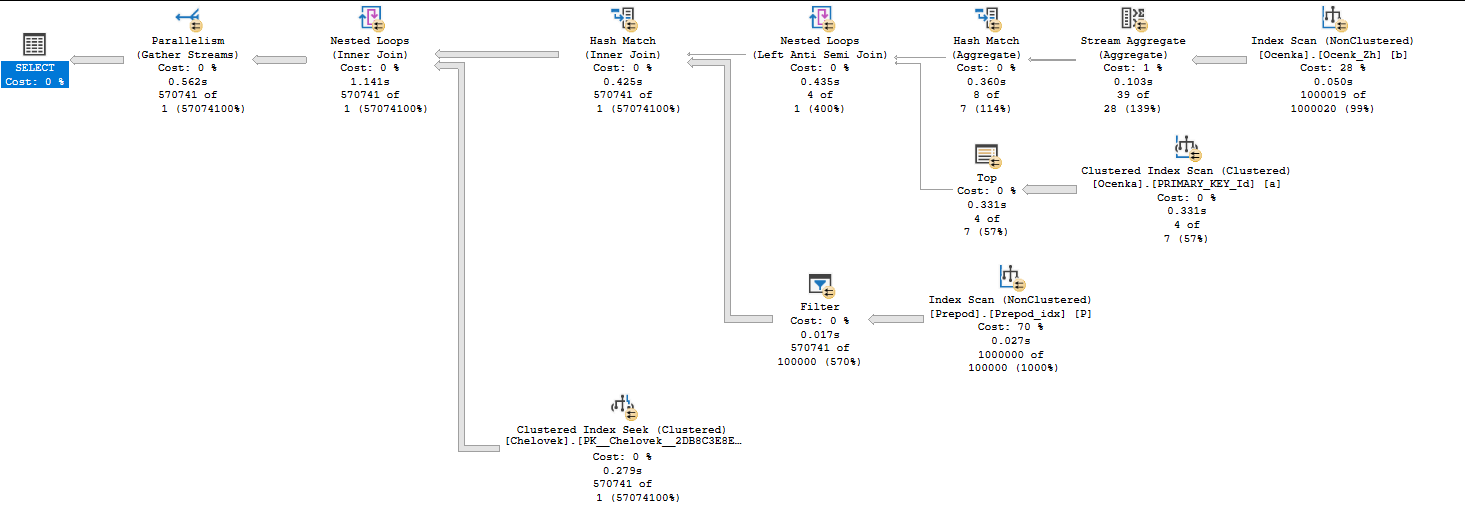


Рисунок 7 - План неоптимизированного запроса с индексами, исчезла ветка с Filter.

Применим оптимизации:

SQL-код оптимизированный

|  |
| --- |
| SELECT Chelovek.FIO FROM Chelovek  JOIN  (SELECT P.NumerPassporta FROM Prepodavatel AS P  JOIN --даёт из названий курсов номера пасппаротов  (SELECT a.ZHurnal\_OcenokKursNazvanie FROM Ocenka a  GROUP BY a.ZHurnal\_OcenokKursNazvanie  HAVING MIN(a.PromezhutochnyeOcenka1)>3 AND MIN(a.PromezhutochnyeOcenka2)>3 AND MIN(a.PromezhutochnyeOcenka3)>3 AND MIN(a.ItogovayaOcenka)>3) Raznost  ON P.RaspisanieKursNazvanie=Raznost.ZHurnal\_OcenokKursNazvanie) Pass  ON Pass.NumerPassporta=Chelovek.NumerPassporta; |

Тогда план нового запроса принимает вид:

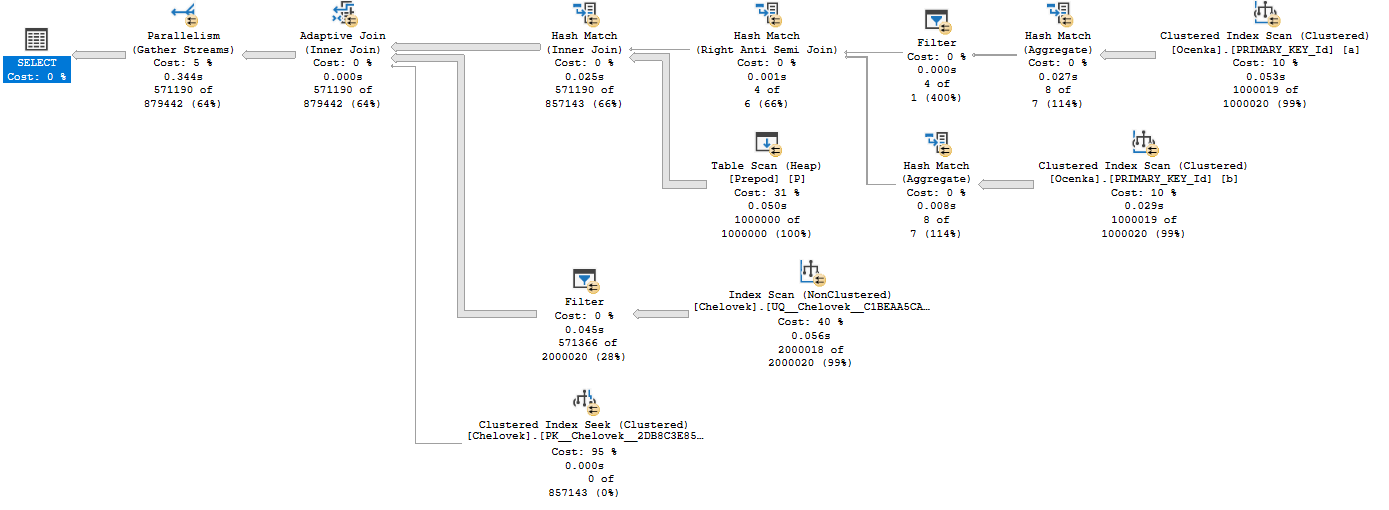


Рисунок 8 – Физический план оптимизированного запроса без индексов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| оптимизированный запрос | | | | | |
| Без индексов | Индекс на FIO | Индекс на Prepod.  NumerPassporta | Индекс на Prepod.  NumerPassporta и Prepod. RaspisanieKursNazvanie | Индекс на  Ocenka.  ZHurnal\_OcenokKursNazvanie | Индексы на все столбцы |
| 3132 | 3225 | 3186 | 3154 | 3196 | 3113 |
| 3175 | 3184 | 3189 | 3243 | 3152 | 3131 |
| 3128 | 3152 | 3272 | 3185 | 3212 | 3156 |
| 3155 | 3142 | 3230 | 3159 | 3199 | 3147 |
| 3151 | 3153 | 3200 | 3258 | 3262 | 3132 |
| 3148.2 | 3171.2 | 3215.4 | 3199.8 | 3204.2 | 3135.8 |

Таблица 3 – Время исполнения оптимизированного запроса с различными индексами.

Тогда физический план лишился ветки и ведущего к ней элемента Hash Match.

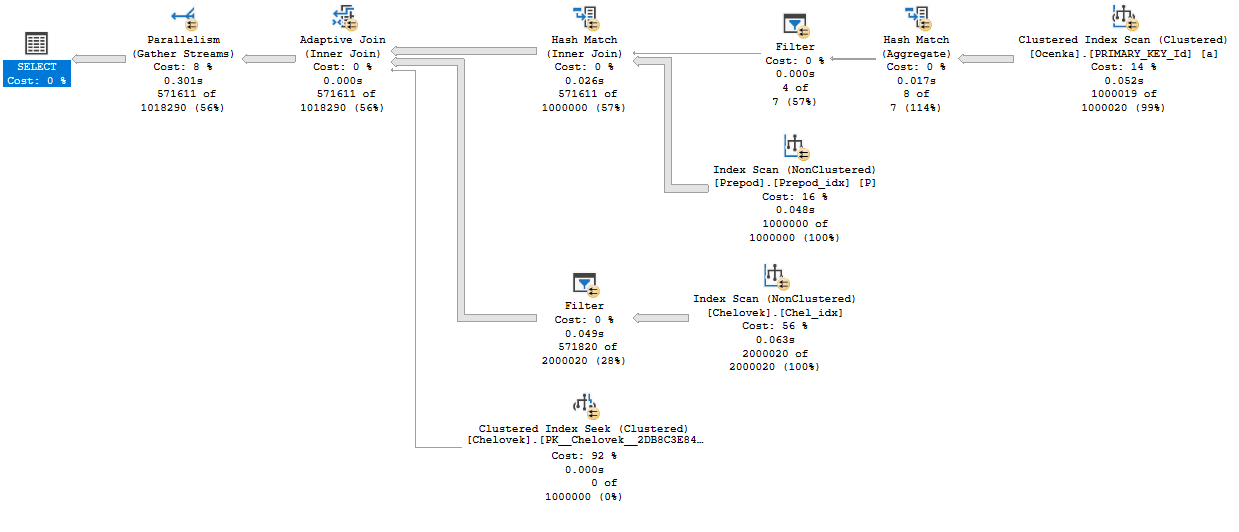


Рисунок 9 – Физический план оптимизированного запроса.

Таким образом, в этом случае мы убрали две ветки и получили выигрыш в 0,532 секунды. Или же 15%. Узким местом оказывается распараллеливание и фильтрация.

**Третий запрос**

Определить средний балл для каждого студента по завершенным курсам (по которым есть итоговая оценка). Выведите результат в виде ФИО – Средняя оценка, отсортировав по оценке от максимальной к минимальной.

|  |
| --- |
| SELECT (SELECT Chelovek.FIO FROM Chelovek WHERE Chelovek.NumerPassporta = ObuchayushchijsyaNumerPassporta) FIO, AVG(ItogovayaOcenka) as Average  FROM Ocenka  GROUP BY ObuchayushchijsyaNumerPassporta  ORDER BY Average DESC; |

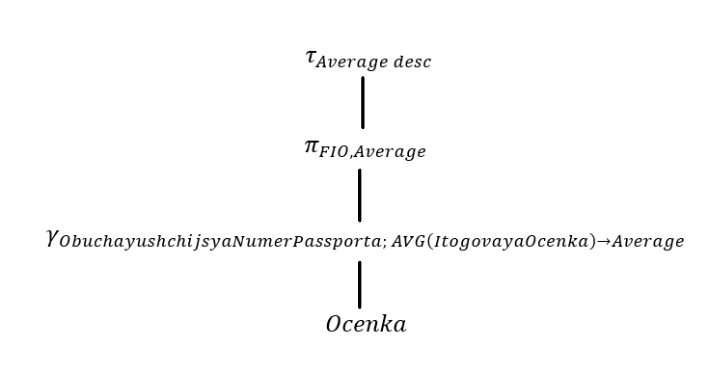


Рисунок 10 - Логическая схема третьего запроса.

РА:

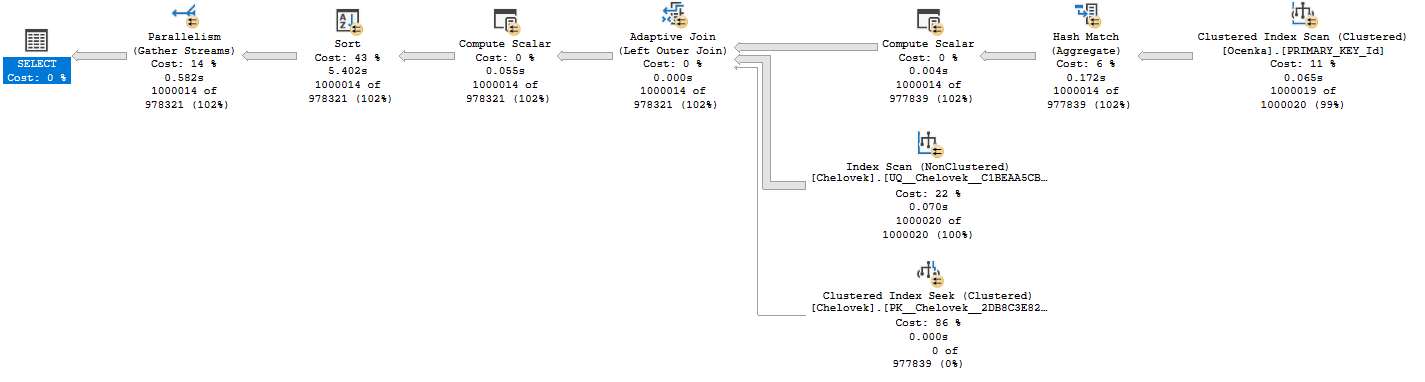
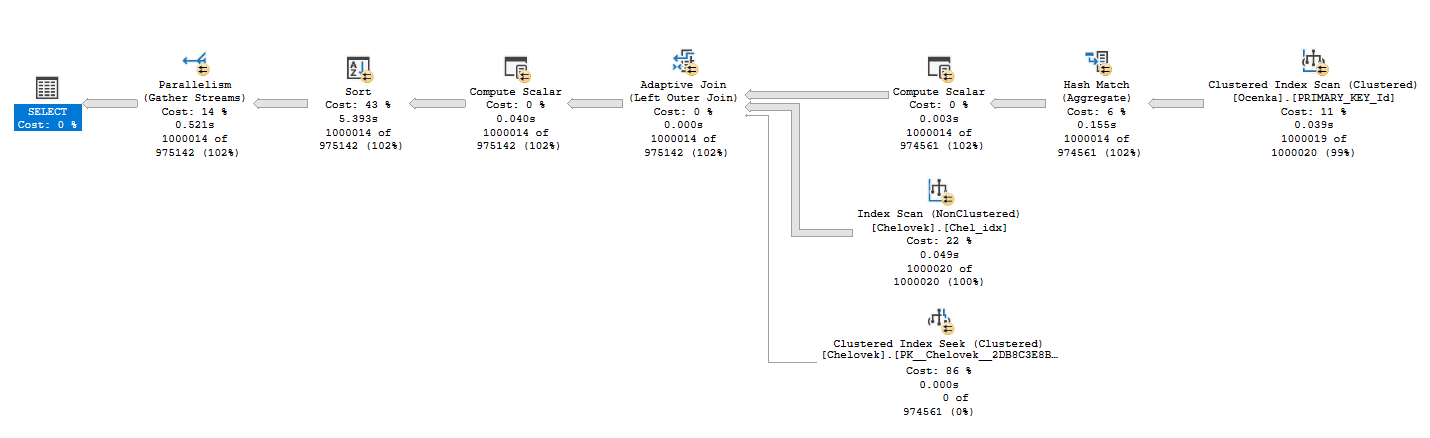


Рисунок 11 - Реальный план без оптимизаций.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запрос и его оптимизации индексами(+ option recompile) | | | | |
| Без индексов | Индекс на FIO | Индекс на Ocenka. ItogovayaOcenka | Индекс на Ocenka. ObuchayushchijsyaNumerPassporta | На все столбцы |
| 6433 | 5704 | 5694 | 5756 | 5774 |
| 6012 | 5772 | 5786 | 5765 | 5775 |
| 6722 | 5747 | 5759 | 5695 | 5664 |
| 5989 | 5710 | 5888 | 5670 | 5624 |
| 5889 | 5636 | 5711 | 5762 | 5673 |
| 6209 | 5713.8 | 5767.6 | 5729.6 | 5702 |

Таблица 4 – Оптимизации запроса индексами.

Хоть оптимизации и улучшили временные показатели, физический план остался без изменений:



Таким образом, выигрыш составил ~ 8% (507 мс). Узким местом является параллелизм и сортировка.

Общий перечень низкоуровневых операций, использованных при выполнении всех запросов, представлен в таблице 5.

|  |
| --- |
| Index Seek |
| Clustered Index Scan |
| Parallelism |
| Sort |
| Nested Loops |
| Hash Match |
| Index Spool |
| Compute Scalar |
| Stream Aggregation |
| Merge Join |
| Table Spool |

Таблица 5 - Перечень низкоуровневых операций.

**Выводы**

Были изучены этапы построения и выполнения запросов, а также основные принципы оптимизации в РСУБД. Были получены практические навыки по построению логических планов запроса, а также по оптимизации запросов на основе логических планов с применением законов реляционной алгебры. Были получены навыки по применению индексов для оптимизации запросов.

Среди рассмотренных в данной работе запросов, лишь несколько оказались достаточно сложными, чтобы из можно было попытаться оптимизировать. MS SQL предоставляет обширный функционал для анализа производительности запросов, позволяет строить удобные планы для анализа эффективности отдельных операций в запросе, а также упрощает поиск узких мест.

**Список источников**

1. Базы данных. Лабораторный практикум. Полтавцева М.А.: учебное пособие / М.А. Полтавцева. – СПб.: Изд-во Политехн. Ун-та, 2016 г.
2. Конспект лекций по СУБД.
3. Документация к MS SQL. <https://docs.microsoft.com/ru-ru/sql/relational-databases/>