Карпухина Анна

Группа М60-307Б-18

**Оглавление**

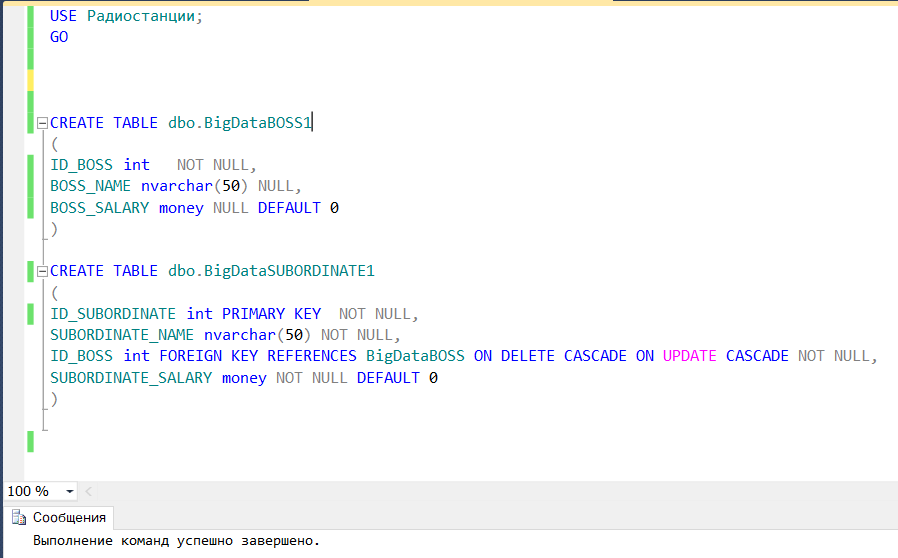
[**Задание 1** 2](#_Toc72299957)

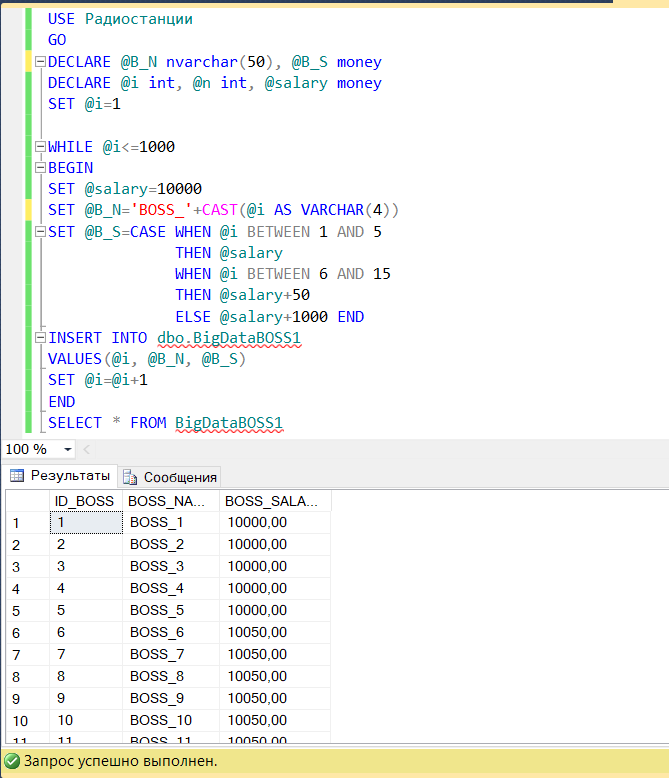
[**Задание 2** 6](#_Toc72299958)

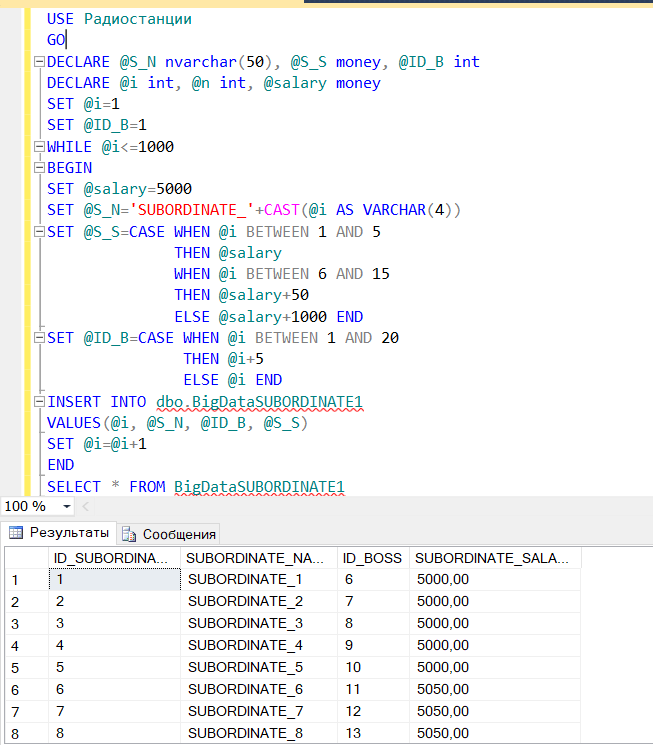
[**Задание 3** 12](#_Toc72299959)

# **Задание 1**

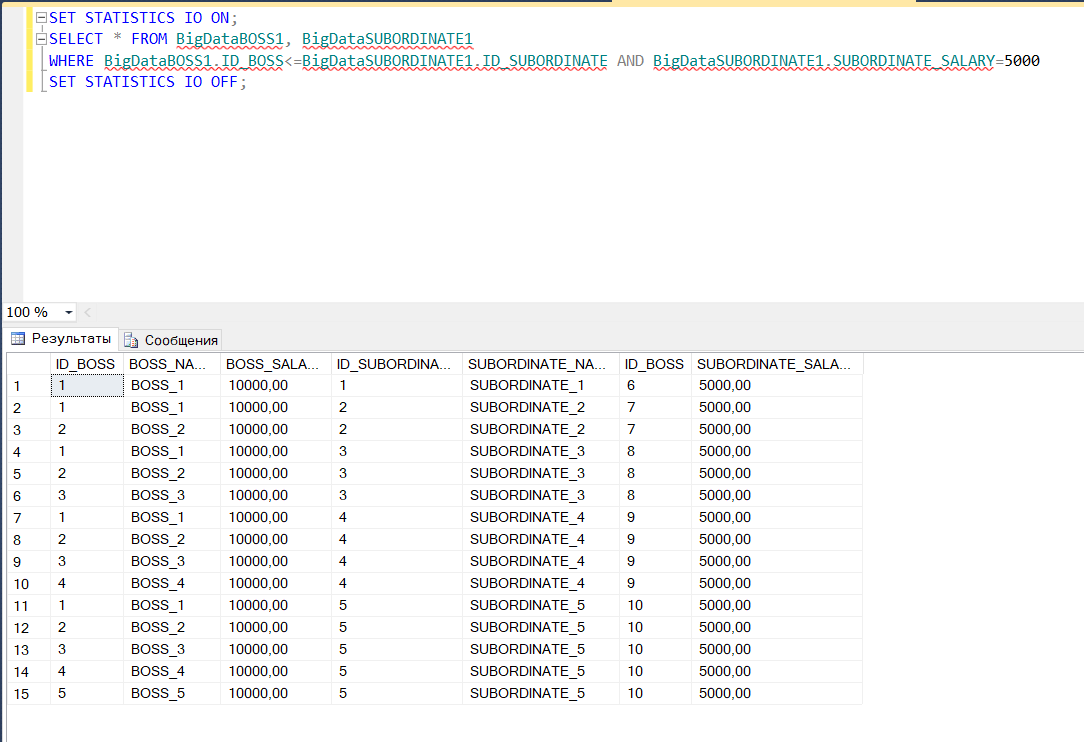
Создадим две таблицы и заполним их большим количеством данных (1000 строк).







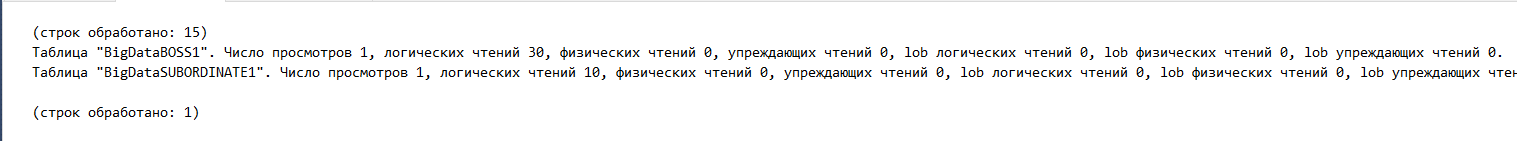
Приступим к осуществлению запроса и его анализу.



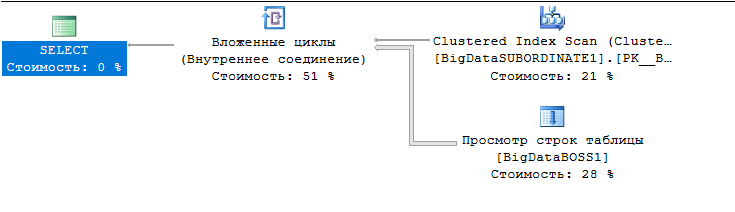
***Анализ:***

У этого запроса у таблицы BigDataBOSS1 осуществилось 30 обращений к внешней памяти, у таблицы BigDataSUBORDINATE1 10 обращений к внешней памяти. Столько раз компонент database engine запросил страницу из буферного кэша.

Данные уже находятся в кэше СУБД (в памяти) поэтому физических чтений 0.



Теперь рассмотрим план выполнения запроса.



На рисунке выше представлена последовательность операций, необходимых для получения результата SQL запроса. Этот план был сохранен в кэше.

В данном примере было произведено сканирование – худший вариант с точки зрения производительности, так как в поиске нужной информации просматривается весь индекс. Лучшим вариантом будет оптимизация такого запроса. Так, можно, например, использовать кластеризованный индекс и использовать SELECT \*, а указать конкретные столбцы.

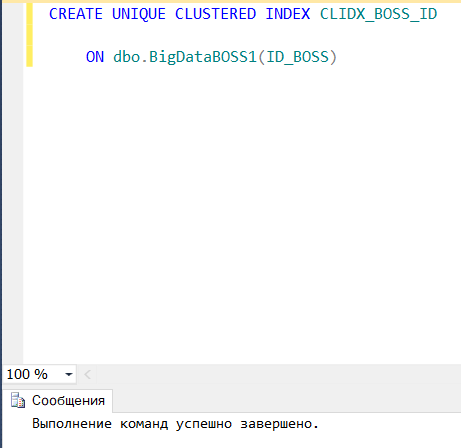
План выполняется справа налево и сверху вниз.

# **Задание 2**

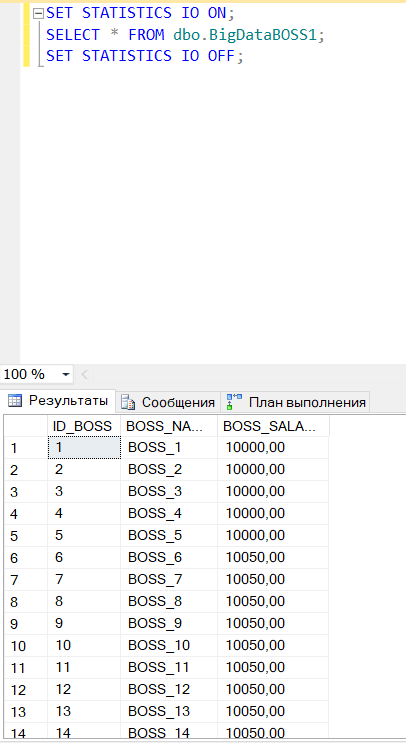
Можно считать, что селективность-это оценочное количество строк, которые могут быть выбраны при поиске по каждому значению индекса. Уникальный индекс имеет максимально возможную селективность, потому что он не может быть выбран более одной строки для каждого значения.

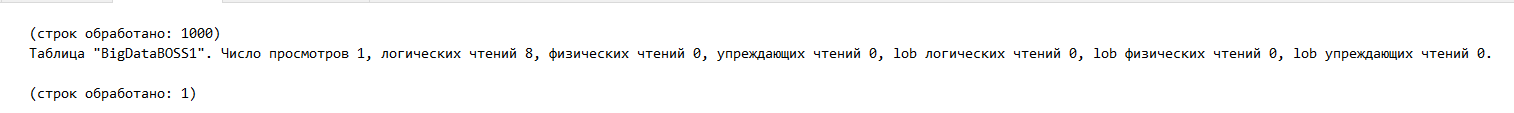
Кластеризованные индексы сортируют и хранят строки данных в таблицах или представлениях на основе их ключевых значений. Этими значениями являются столбцы, включенные в определение индекса. Этот индекс можно считать —древовидная структура данных, при которой значения индекса хранятся вместе с данными, им соответствующими. И индексы, и данные при такой организации упорядочены. При добавлении новой строки в таблицу, она дописывается в нужную ветку древовидной структуры, соответствующую ей по сортировке.

Созданим кластеризованный индекс:



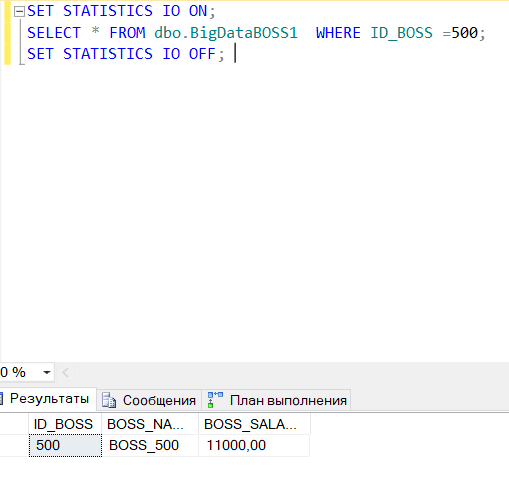
Обращение к таблице:

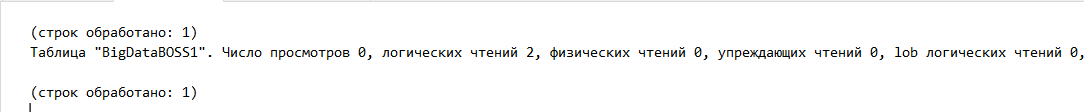




Обращений к памяти 8. Примерно столько же обращений было совершено и без использования кластеризованного индекса. Запрос низкоселективный.

Теперь же упростим запрос, добавим более конкретное условие и посмотрим как оно влияет на план.

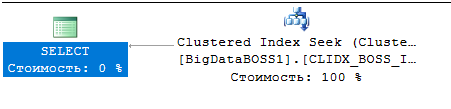




Заметим, что логических чтений теперь 2, а не 8. Запрос считается высокоселективным.

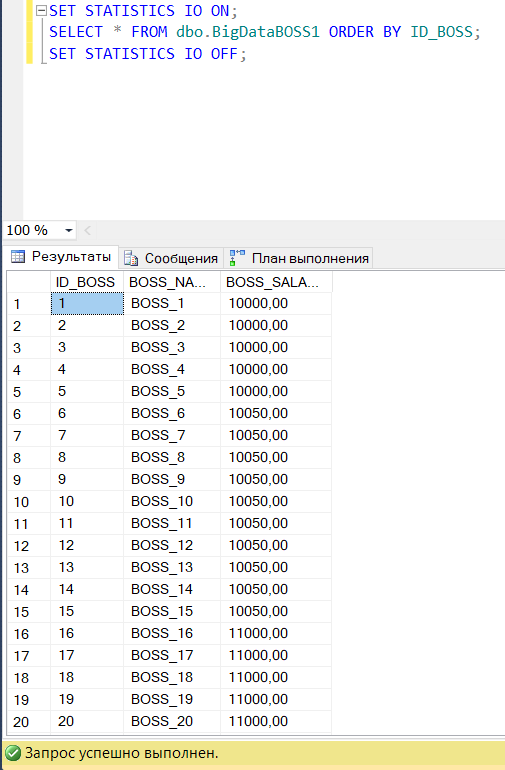
Требует гораздо меньше количества операций ввода/вывода, чем поиск в каждой странице данных.

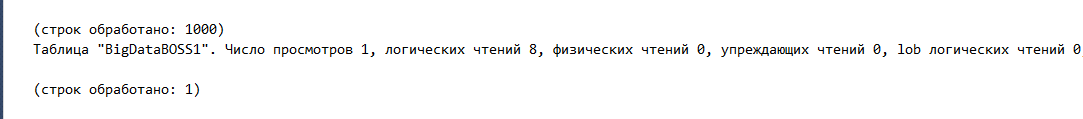
Отличается от  запроса, когда не было индекса.



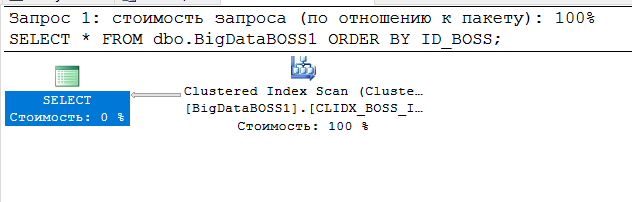
Также видим, что вместо Index Scan сканирования в плане запроса Index Seek осуществляется поиск – прямой доступ к данным.

Теперь посмотрим на план запроса, осуществив сортировку.



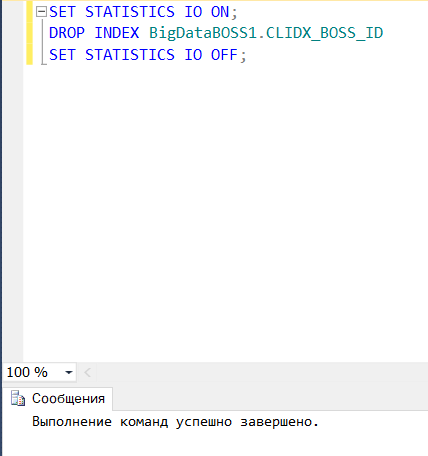


В результате получился следующий план выполнения запроса:

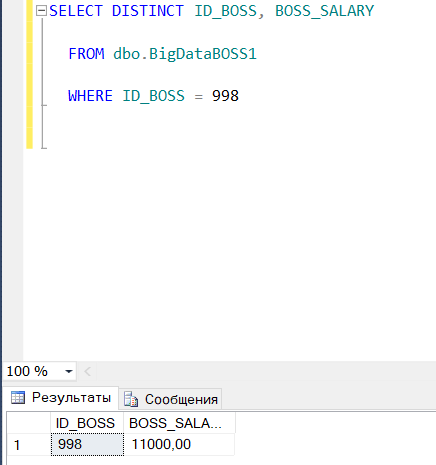


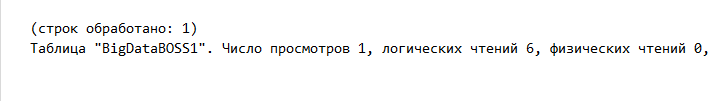
Количество обращений совпадает при обыкновенной выборкой с использованием кластеризованного индекса.

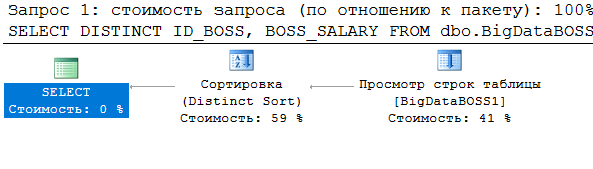
Удалим кластеризованный индекс:



После удаления произведем запрос без кластеризованного индекса.



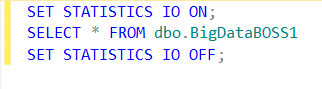


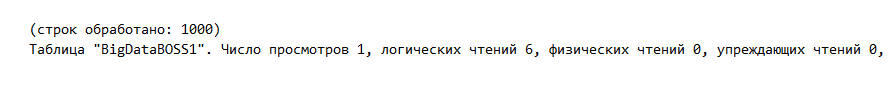


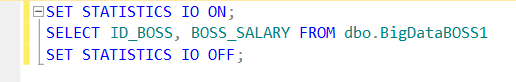
***Вывод***: имеет смысл определять кластеризованный индекс в столбцах, которые часто используются в качестве аргумента сортировки или группировки критериев в агрегате, поскольку агрегация данных требует, чтобы SQL Server сначала сортировал данные в соответствии с критериями группировки.

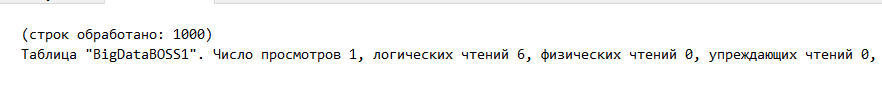
# **Задание 3**

Лучше использовать конкретные имена столбцов после оператора select, вместо «\*». Это позволит увеличить быстроту выполнения запроса и уменьшению сетевого трафика.



При таком запросе 6 обращений к памяти.Теперь изменим количество полей, конкретизируем выборку:

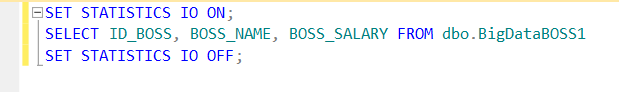


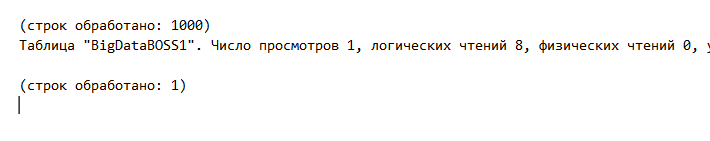


В данном случае вышло одинаковое количество логических чтений, но на большем количестве данных будет явно замечена разница в количестве обращений к внешней памяти.

Теперь посмотрим влияние оператора выборки с использованием в таблице кластеризованного индекса.

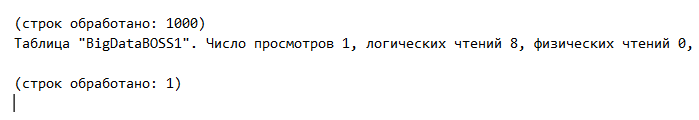
Снова создадим кластеризованный индекс.





Итак, при запросе с конкретным указанием полей – 8 логических чтений.

Теперь рассмотрим случай, когда в операторе выбора используется «\*».



Также 8 логических чтений.

Можно сделать вывод, что в данном случае кластеризация индекса не играет большой роли.

# **Задание 4**

*Соединение с вложенными циклами (Nested loop)*

Начнем с соединения вложенными циклами (nested loop), как с самого простого. Его алгоритм таков: для каждой строки одного из наборов перебираем и возвращаем соответствующие ему строки второго набора. По сути, это два вложенных цикла, отсюда и название способа.

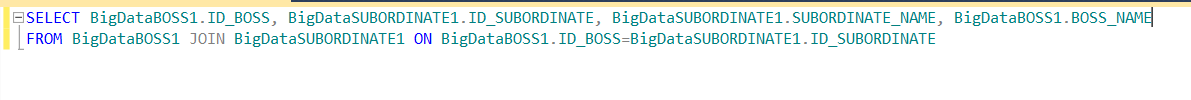
Ко второму (внутреннему) набору мы будем обращаться столько раз, сколько строк в первом (внешнем) наборе. Если нет эффективного метода доступа для поиска «соответствующих» строк во втором наборе (то есть, попросту говоря, индекса на таблице), то придется неоднократно перебирать большое количество строк, не относящихся к делу. Очевидно, это будет не лучший выбор.

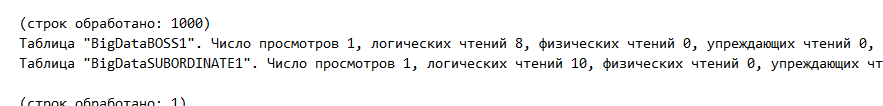
Соединение при помощи вложенных циклов каждый раз формирует в оперативной памяти единственную строку результата. Требуется немного оперативной памяти. Место на диске не нужно. Можно создавать огромные результирующие таблицы при ограниченной оперативной памяти.

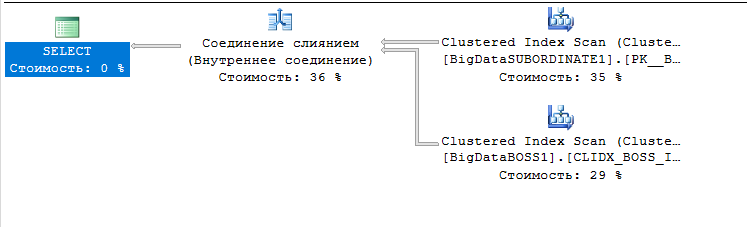
Алгоритм следующий:

Каждая найденная строка передается внутреннему циклу, который, перебирая строки ведомой таблицы, ищет по одной все подходящие строки. Первая такая строка формирует первую строку результата, передавая ее в итоговую таблицу. После того, как будут перебраны все строки ведомой таблицы, внешний цикл, выберет следующую строку ведущей таблицы и т.д.

Создадим запрос с соединением с вложенными циклами.







*Вывод:*

Сильной стороной способа соединения вложенными циклами является его простота: не требуется никаких подготовительных действий, мы может начать возвращать результат практически моментально.

Обратная сторона состоит в том, что этот способ крайне неэффективен для больших объемов данных. Ситуация та же, что и с индексами: чем больше выборка, чем больше накладных расходов.

*Соединение с сортировкой слиянием*

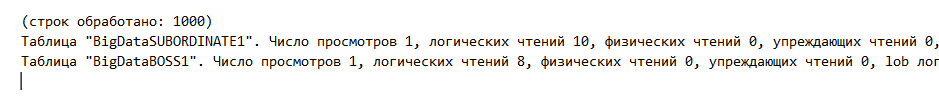
Таблицы считываются независимо. Оба результирующих набора предварительно сортируются по ключу соединения и затем соединяются.

Два отсортированных списка помещаются рядом. Указатели смещаются с верхних записей только вниз. При временно фиксированном левом указателе правый идет вниз до конца, задерживаясь только на соединяемых записях. Затем левый указатель опускается на шаг, а правый начинает движение вниз с позиции на строку ниже, чем в предыдущем цикле.

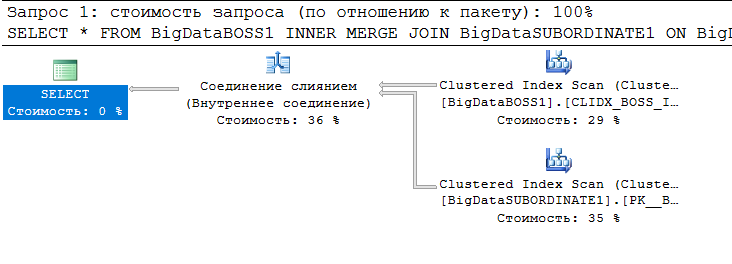
**Merge Joins** теоретически являются самыми быстрыми физическими операторами соединения, однако они требуют, чтобы данные обоих входов были отсортированы.

*Алгоритм соединения сортировки слиянием :*

Базовый алгоритм работает следующим образом: SQL Server сравнивает первые строки обоих отсортированных входов. Затем сравнение продолжается со следующими строками второго входа до тех пор, пока значения соответствуют значению первого входа.  
  
Если соответствий больше нет, SQL Server переходит к следующей строке того входа, который имеет меньшее значение - и затем продолжает выполнение сравнений, выводя каждую соединенную запись.



План выполнения запроса:



При сравнении можно заметить, что логических чтений было больше, чем при соединении с вложенными циклами.

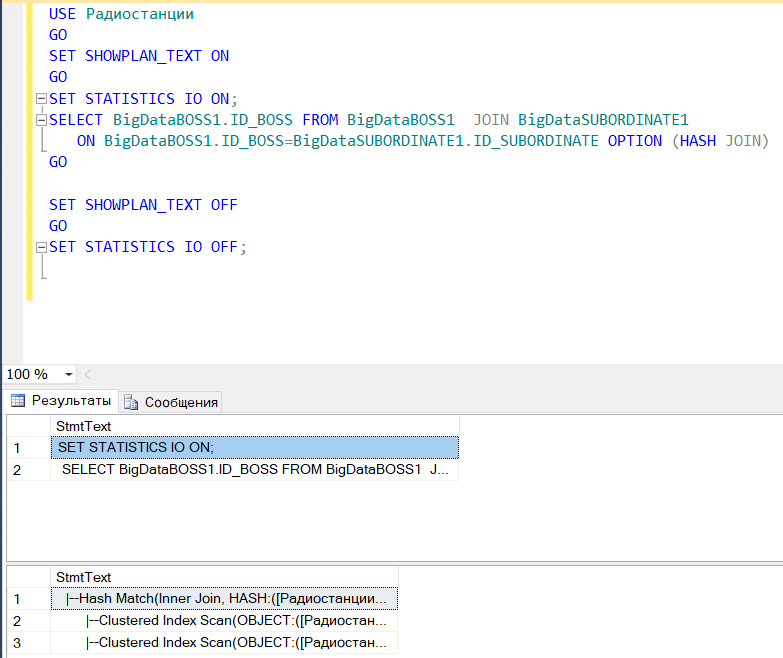
*Хеширование соединения*

Хеш-таблица (hash table) — это специальная структура данных для хранения пар ключей и их значений. По сути это ассоциативный массив, в котором ключ представлен в виде хеш-функции.

Хеш-функция имеет следующие свойства:

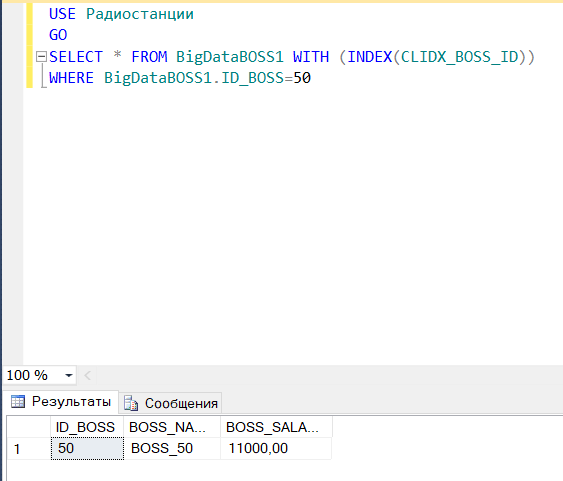
* Всегда возвращать один и тот же адрес для одного и того же ключа;
* Не обязательно возвращает разные адреса для разных ключей;
* Использует все адресное пространство с одинаковой вероятностью;
* Быстро вычислять адрес.

Т.е.— элементы с одинаковым хешем попадают в одну область памяти.

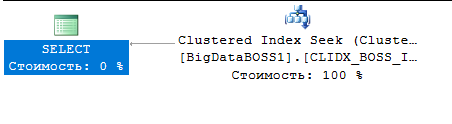


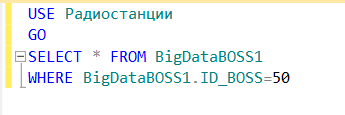
# **Задание 5**

*Подсказка таблицы INDEX*

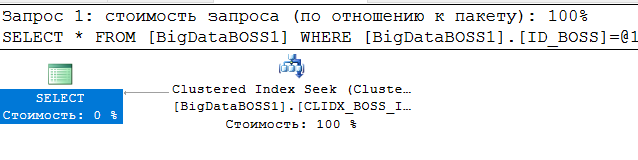


Измененный план с подсказкой





Соответствующий запросу выше план без подсказки.

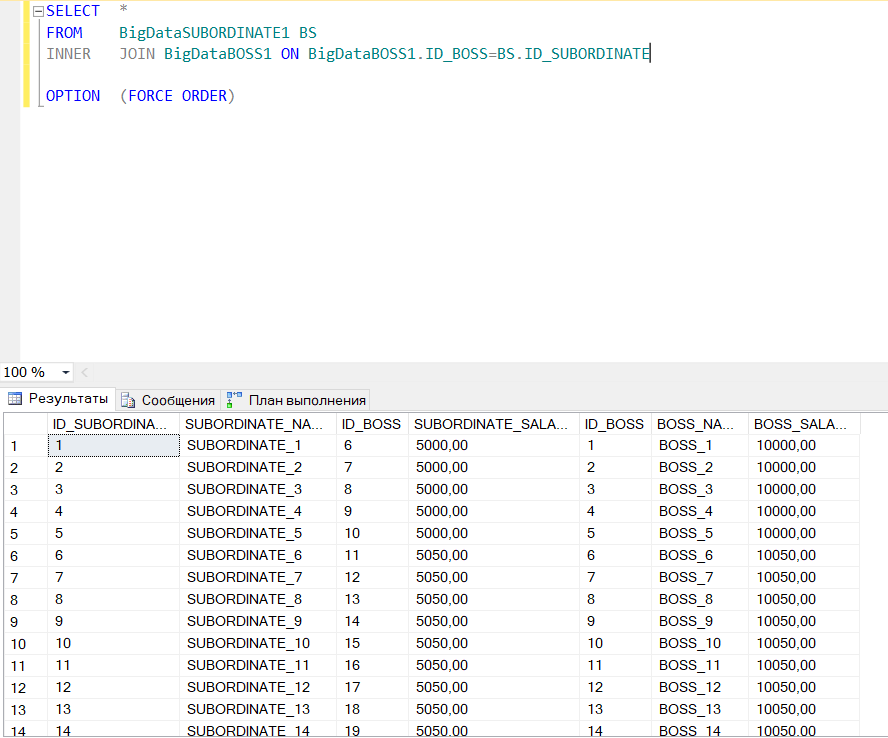


# **Задание 6**

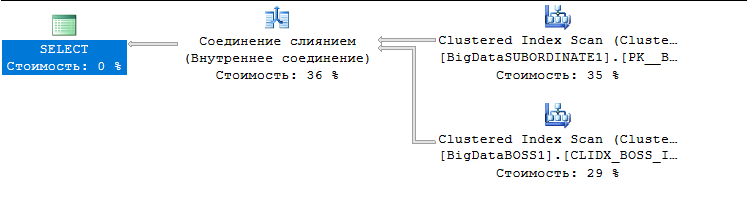
*Подсказка соединения FORCE ORDER*

Подсказки соединения дают указания оптимизатору запросов, как должны выполняться операции соединения в запросе. Они заставляют оптимизатор либо соединять таблицы в том порядке, в каком они указаны в предложении FROM оператора SELECT, либо использовать техники выполнения соединения, явно указанные в операторе. Database Engine поддерживает несколько подсказок соединения, включающих и FORCE ORDER.

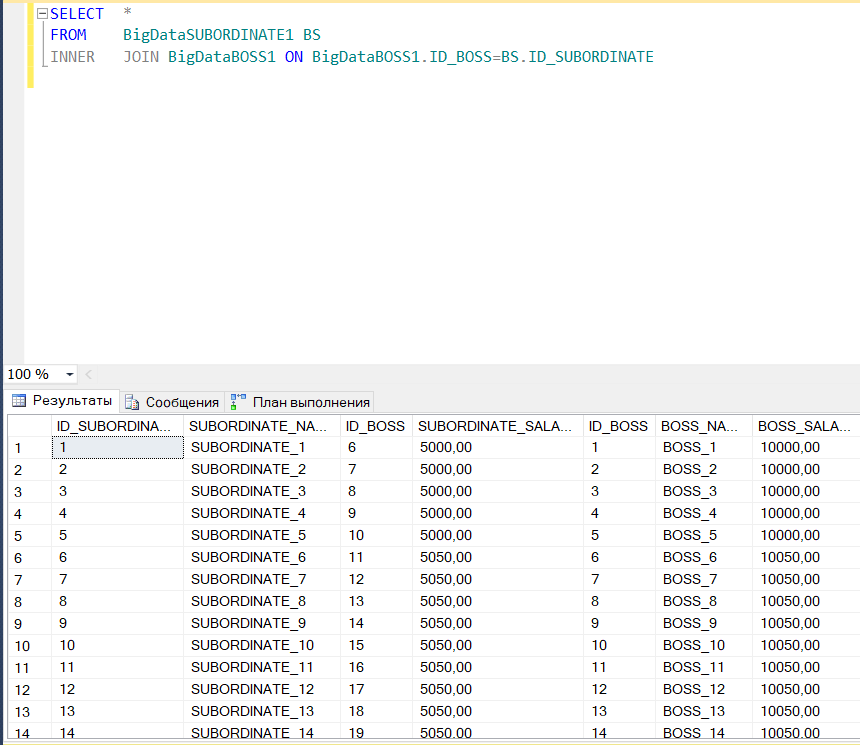
Запрос с подсказкой FORCE ORDER.



План с подсказкой FORCE ORDER



Запрос без подсказки FORCE ORDER



План без подсказки FORCE ORDER

