|  |
| --- |
| EPAM Systems, RD Dep. |
| Oracle Data Access and Optimizer |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | Valeryia\_Lupanava | 04-NOV-2017 |  |  |

Содержание

[1. Table access 3](#_Toc497520210)

[1.1. Задание 1: Full Scan, High-Water Mark and Consistent Gets 3](#_Toc497520211)

[1.1.1. Шаг 1: 3](#_Toc497520212)

[1.1.2. Шаг 2: 3](#_Toc497520213)

[1.1.3. Шаг 3: 3](#_Toc497520214)

[1.1.4. Шаг 4: 4](#_Toc497520215)

[1.1.5. Шаг 5: 6](#_Toc497520216)

[1.1.6. Шаг 6: 7](#_Toc497520217)

[1.1.7. Результат: 8](#_Toc497520218)

[2. Index Scan types 9](#_Toc497520219)

[2.1. Задание 2: Index Clustering Factor 9](#_Toc497520220)

[2.1.1. Шаг 1: 9](#_Toc497520221)

[2.1.2. Шаг 2: 9](#_Toc497520222)

[2.1.3. Шаг 3: 9](#_Toc497520223)

[2.1.4. Шаг 4: 10](#_Toc497520224)

[2.1.5. Результат: 10](#_Toc497520225)

[2.2. Задание 3: Index Unique Scan 11](#_Toc497520226)

[2.2.1. Шаг 1: 11](#_Toc497520227)

[2.2.2. Шаг 2: 11](#_Toc497520228)

[2.2.3. Результат: 11](#_Toc497520229)

[2.3. Задание 4: Index Range Scan 12](#_Toc497520230)

[2.3.1. SELECT поля с обычным индексом: 12](#_Toc497520231)

[2.3.2. Результат: 12](#_Toc497520232)

[2.4. Задание 5: Index Skip Scan 13](#_Toc497520233)

[2.4.1. Шаг 1: 13](#_Toc497520234)

[2.4.2. Шаг 2: 13](#_Toc497520235)

[2.4.3. Шаг 3: 13](#_Toc497520236)

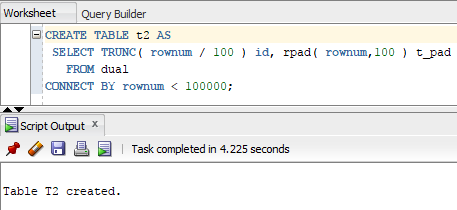
[2.4.4. Результат: 14](#_Toc497520237)

# Table access

## Задание 1: Full Scan, High-Water Mark and Consistent Gets

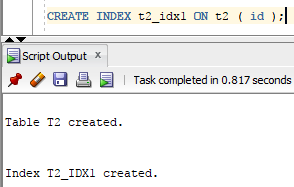
### Шаг 1:

Была создана таблица, в которой в качестве ID – целое число, полученное в результате деления ROWNUM/100, а T\_PAD – значение ID, дополненное числом от 0 до 100 по порядку. В таблицу должно записаться 99999 рядов.



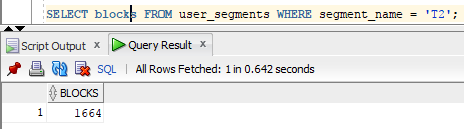
### Шаг 2:

Создаем индекс по полю ID.

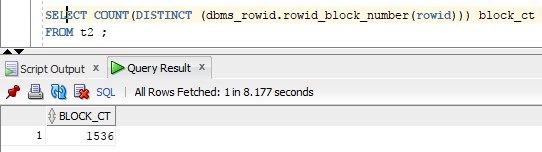


### Шаг 3:

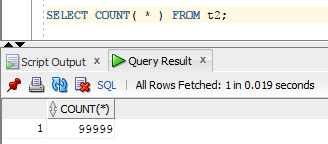
Количество блоков, в которые были выделены под таблицу.



Количество блоков из выделенных, которые забиты данными:

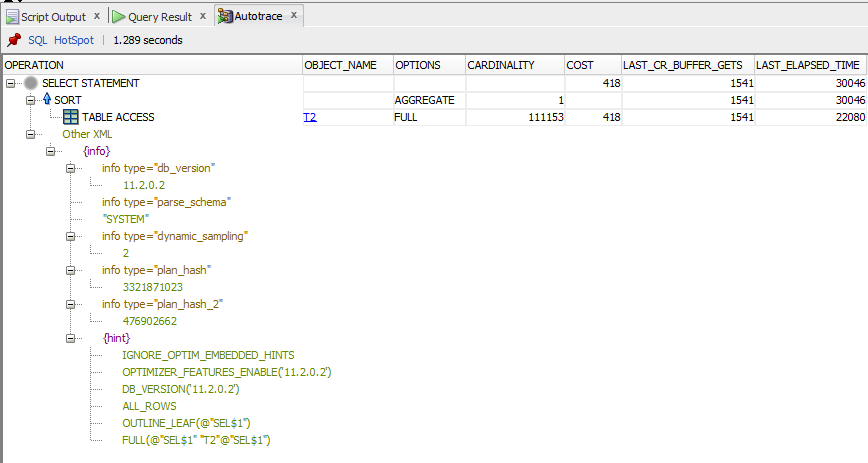


Количество строк в таблице:



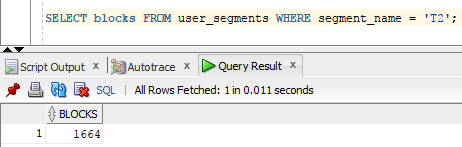
Количество раз последовательного чтения для блока можно увидеть в значении параметра **CONSISTENT GETS**:

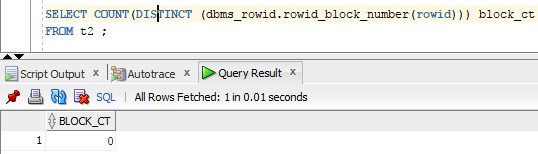
|  |  |
| --- | --- |
| consistent gets | 1618 |
| consistent gets from cache | 1618 |
| consistent gets from cache (fastpath) | 1618 |

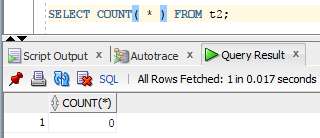


### Шаг 4:

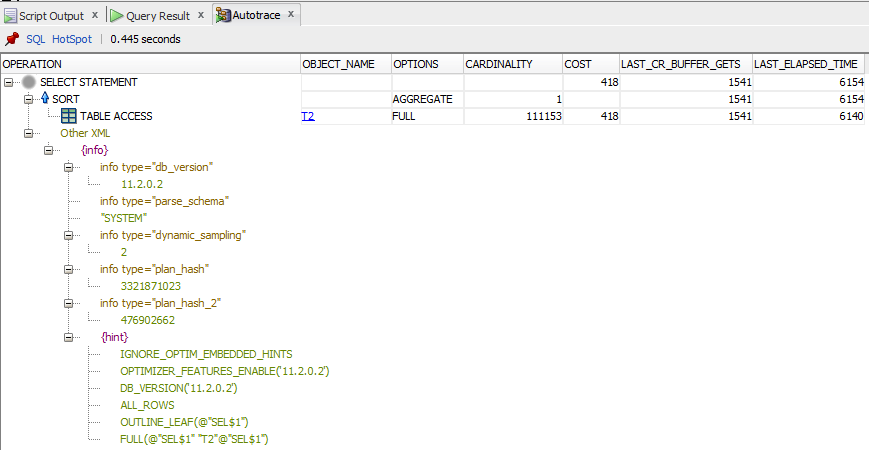
Удаляем все строки и собираем статистику снова.





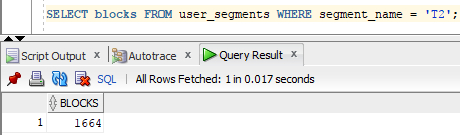


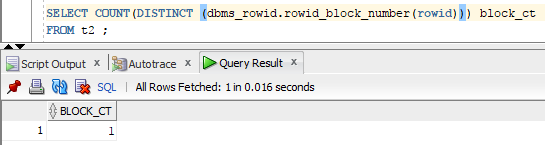
|  |  |
| --- | --- |
| consistent gets | 1541 |
| consistent gets from cache | 1541 |
| consistent gets from cache (fastpath) | 1541 |

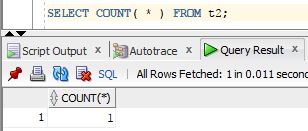


### Шаг 5:

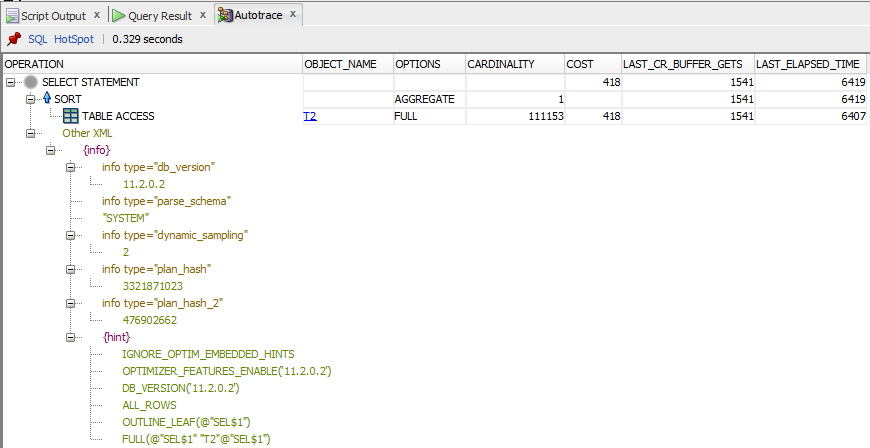
Вставляем одну строку и собираем статистику.





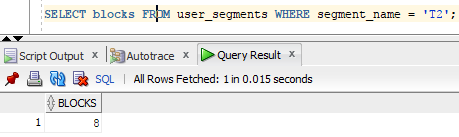


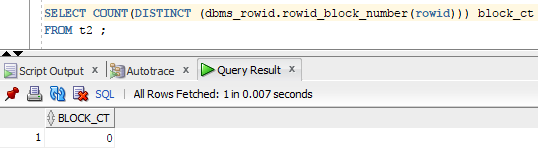
|  |  |
| --- | --- |
| consistent gets | 1541 |
| consistent gets from cache | 1541 |
| consistent gets from cache (fastpath) | 1541 |

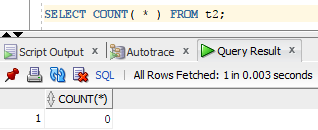


### Шаг 6:

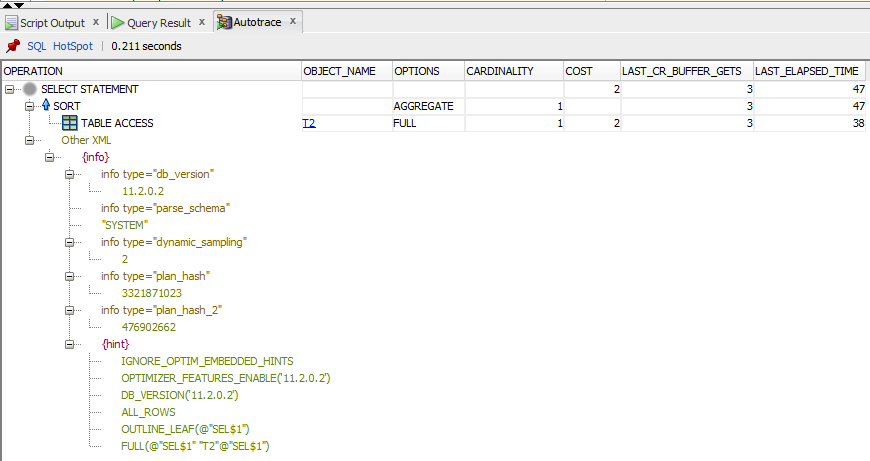
Очищаем таблицу и собираем статистику.







|  |  |
| --- | --- |
| consistent gets | 5 |
| consistent gets from cache | 5 |
| consistent gets from cache (fastpath) | 5 |



### Результат:

* Описание результатов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1. create | 1664 | 1536 | 99999 | 1618 | Была создана таблица с количеством строк 99999. Под таблицу было выделено 1664 блока данных. Фактически данными занять только 1536 блоков, эти же блоки получили метку HWM, поскольку были заполнены данными. При SELECT произошло 1618 последовательных чтений блоков с диска и с CASH. |
| 1. delete | 1664 | 0 | 0 | 1541 | Далее мы удалили все строки в таблице. Можно увидеть, что выделенные таблице блоки остались. При удалении Oracle хранит удаленные строки в ROLLBACK SEGMENTS, чтобы можно было отменить изменения позже, поэтому количество выделенных блоков не изменилось. Но видно, что они просто не используются, поскольку данных нет – 0 rows. Логическое чтение блоков продолжается, поскольку HWM остается. |
| 1. insert | 1664 | 1 | 1 | 1541 | После вставки данных мы снова задействуем один блок, поскольку одна строка вставлена. Снова логическое чтение блоков происходит из-за HWM. |
| 1. truncate | 8 | 0 | 0 | 5 | TRUNCATE сбрасывает HWM и освобождает экстенты, выделенные для таблицы, поэтому количество блоков уменьшилось в разы. |

# Index Scan types

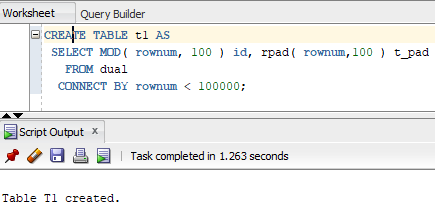
## Задание 2: Index Clustering Factor

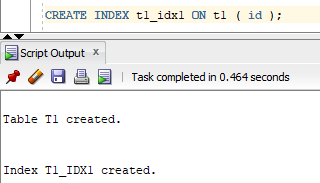
### Шаг 1:

Снова создали таблицу из первого задания с индексом по колонке ID.

### Шаг 2:

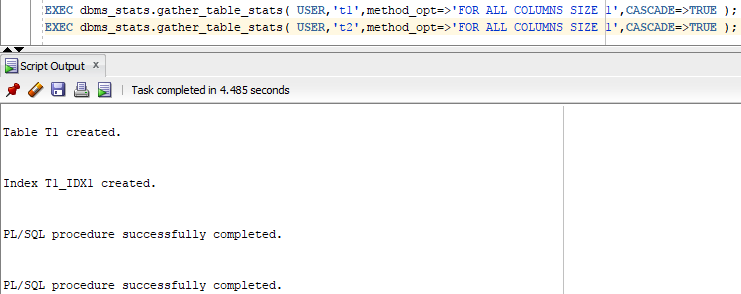
Создали вторую таблицу также с индексом по первому полю ID.





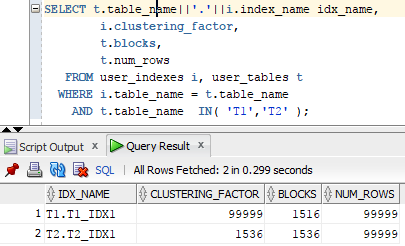
### Шаг 3:

Собрали статистику по таблицам.



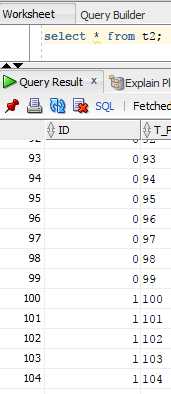
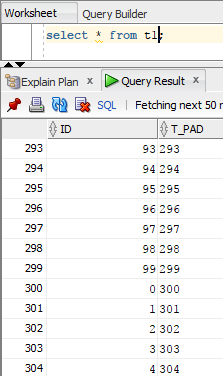
### Шаг 4:

Значения коэффициентов кластеризации для двух таблиц.



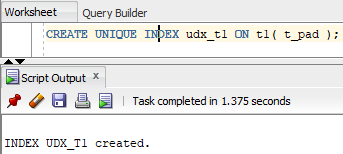
### Результат:

* **Коэффициента кластеризации.** Oracle применяет коэффициент кластеризации, чтобы определить оптимальный путь извлечения данных: полное сканирование или по индексу. В общем, коэффициент кластеризации показывает, сколько операций ввода-вывода будет выполнять БД, если бы она считывала каждую строку в этой таблице через индекс в порядке индекса. Если строки таблицы на диске отсортированы примерно в том же порядке, что и ключи индекса, база данных будет выполнять минимальное количество операций ввода-вывода в таблице, чтобы прочитать всю таблицу с помощью индекса.
* Поскольку в первой таблице индекс не отсортирован, а просто повторяется блоками, то БД придется сканировать всю таблицу, чтобы извлечь данные с нужным индексом. Во второй таблице напротив, индекс расположен по порядку, поэтому не нужно считывать всю таблицу целиком, а только извлечь нужные строки с интересующим нас индексом.
* Поэтому, логично, что выполнение второй таблицы будет оптимальнее и с меньшим КК. Можно увидеть, что для первой таблицы КК равен количеству строк, поскольку нужные БД строки будут разнесены по разным блокам данных, ей придется прочитать каждую строку каждого блока, чтобы извлечь нужные данные. Тогда как во второй благодаря сортировке, каждая следующая строка, необходимая в соответствии с ключом индекса, скорее всего будет следующей строкой в ​​блоке таблицы. Запрос не будет пропускаться по всей таблице, чтобы найти строку после строки - они естественно рядом друг с другом на блоке. Он просто прочитает первую строку каждого блока и выведет нужный блок. Поэтому во втором случае КК равен количеству блоков.

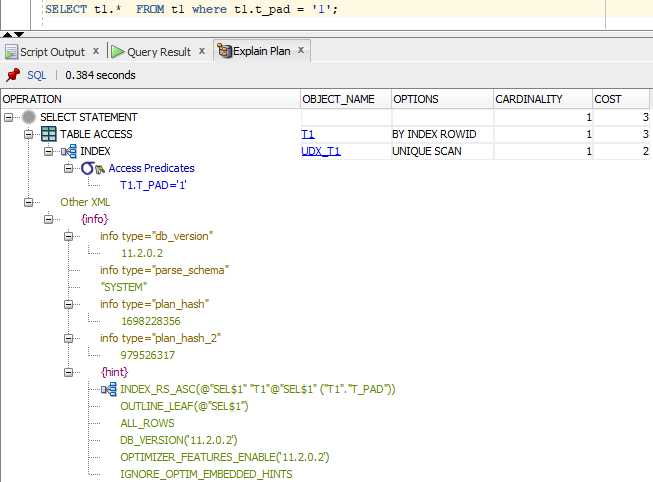


## Задание 3: Index Unique Scan

### Шаг 1:



### Шаг 2:

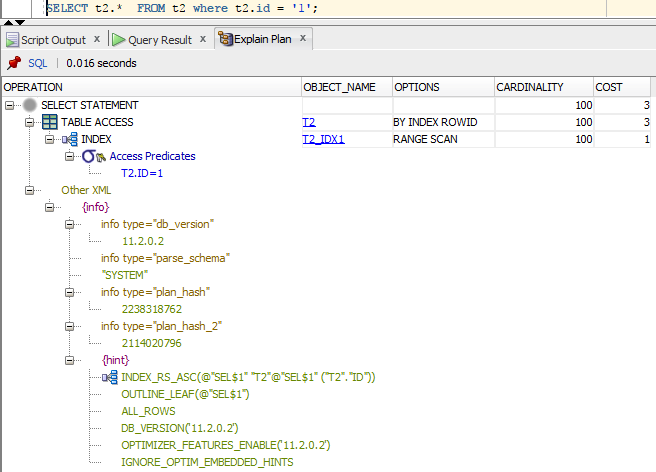


### Результат:

* Index unique scan был выбран в данном случае, поскольку мы определили поле T\_PAD как UNIQUE INDEX и в SELECT мы задали условие поиска именно по этому полю. UNIQUE INDEX гарантирует, что только одна строка будет возвращена для указанного значения. В этом случае индекс будет проходить от корневого блока до единичной записи листового блока, извлекать ROWID и использовать его для доступа к блоку данных таблицы, содержащему одну строку. Поэтому в EXPLAIN TABLE можно наблюдать доступ – BY INDEX ROWID.

## Задание 4: Index Range Scan

### SELECT поля с обычным индексом:

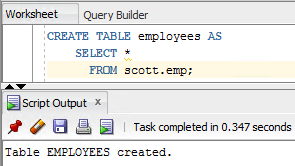


### Результат:

* Во-первых, по поле ID создан индекс. Во-вторых, поле ID содержит определенный диапазон данных с указанным в условии WHERE значением. Также наш диапазон сканирования достаточно селективный. БД не нужно в данном случае сканировать таблицу. Поиск начинается от корневого блока до первого листового блока, содержащего соответствующую условию запись. Далее извлекается ROWID этой записи и затем читается блок данных таблицы. После этого извлекается следующий ROWID блока. Это будет продолжаться до тех пор, пока не будут прочитаны все соответствующие записи индекса. Если диапазон записей, соответствующих условию, находится более, чем в одном блоке, то будет задействован указатель. Указатель сохраняется в текущем блоке листа. При необходимости чтения еще одного блока БД просто перейдет по указателю на следующий.

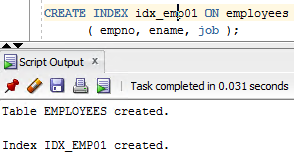
## Задание 5: Index Skip Scan

### Шаг 1:



### Шаг 2:

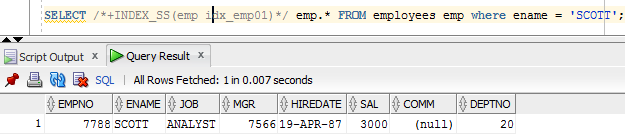
Создаем составной индекс по трем полям таблицы.

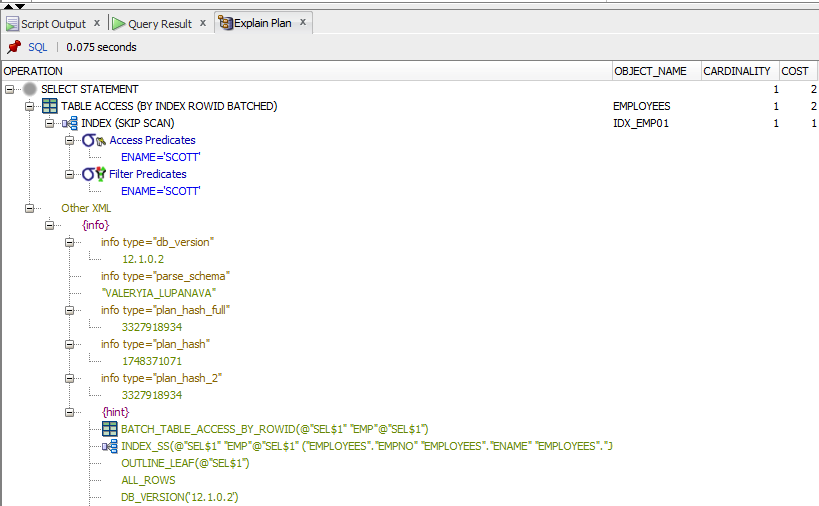


### Шаг 3:

Собираем статистику по запросам ниже.

SKIP SCAN HINT.





FULL SCAN HINT.



### Результат:

* В первом SELECT был применен SKIP SCAN, во-первый, поскольку в WHERE было пропущено ведущее поле EMPNO. БД разбивает составной индекс на подиндексы. Подиндекс ведущего поля должен включать как можно больше подиндексов второго по очереди в индексе поля. В таком случае при пропуске в WHERE ведущего поля и поиске по второму, например, будет достаточно селективный поиск. Если же индекс ведущего поля не содержит слишком мало подинексов, то БД совершит обычный FULL SCAN. А во-вторых, мы указали SS HINT, что означает, что в любом случае будет работать SKIP SCAN.

Во втором случае у нас осуществляется FULL SCAN, поскольку мы указали FULL HINT, что означает, что нам нужно полное сканирование таблицы, несмотря на то, что предложит оптимизатор.

Таким образом, очевидно, что HINT отключает оптимизатор запроса и выполняем запрос указанным явно способом.

Для проверки сделаем SELECT без HINT и увидим какой был выбран план оптимизатором:



Оптимизатор выбрал SKIP SCAN. Как упоминалось выше, поскольку мы осуществляем поиск не по ведущему полю составного индекса, оптимизатор предлагает SKIP SCAN.

Анализ AUTOTRACE для трех SELECT одинаковый.

* Результаты анализа (сделано, для таблицы EMPLOYEES, как в первом задании; описание не указываю, поскольку результаты те же).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets |
| 1. create | - | 1 | 14 | 11 |
| 1. delete | - | 0 | 0 | 3 |
| 1. insert | - | 1 | 1 | 3 |
| 1. truncate | - | 0 | 0 | 3 |