# Table access

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

Consistent gets – это количество раз считываемых блоков из буферного кэша, необходимых для установления консистентного состояния. Смысл консистентности в том, чтобы гарантировать согласованное состояние данных, включая видимые изменения, сделанные собственными транзакциями пользователя, и совершенные транзакции других пользователей. Так вот consistent gets как раз показывает количество чтение из буфера для получения этого состояния.

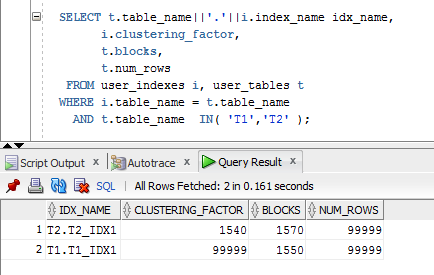
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1539 | Create - При создании таблицы и вставки в нее строк из буфера считываются 1539 блоков = 1536 с заполненными данными + 3 на метаданные. |
| 2 | 1664 | 0 | 0 | 1541 | Delete - При удалении данных Oracle считывает из буфера 1539 блоков + 2 на данные для отката операции. |
| 3 | 1664 | 1 | 1 | 1541 | Insert - После удаления строк HWM остался на прежнем месте, статистика не была пересобрана, поэтому при вставке 1 строки Oracle вставляет ее в первое свободное место и количество сonsistent gets не изменяется. |
| 4 | 8 | 0 | 0 | 3 | Truncate - Поскольку операция truncate очищается и сбрасывает HWM, то значение сonsistent gets минимально (метаданные) |

Для только что созданной таблицы HWM-маркер начинается в 1 блоке. По мере помещения данных в таблицы количество занятых блоков увеличивается и HWM-маркер сдвигается вправо. При удалении строк из таблицы, появляется много блоков, которые больше не содержат данных, но по-прежнему находятся перед HWM-маркером и остаются там до тех пор, пока объект не будет перестроен, очищен или сжат.

Во время оператора TRUNCATE он сбрасывает HWM-маркер таблицы практически в ноль и также очищает все ассоциированные индексы. Поэтому при удалении всех строк из таблицы, оператор TRUNCATE будет наилучшим выбором.

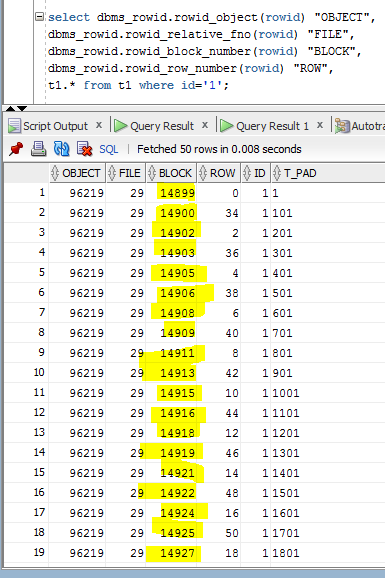
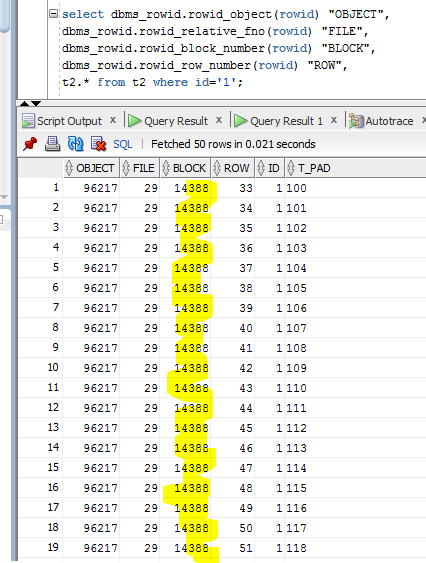
# Index Scan types

## Task 2: Index Clustering Factor

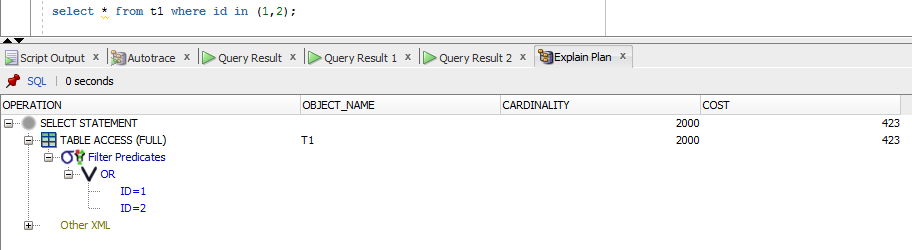


Кластерный фактор - это показатель, который выражает степень распределения записей в таблице относительно индексных ключей. Или другими словами, это число, представляющее количество логических операций ввода-вывода в таблице, которые должны быть выполнены для чтения всей таблицы через индекс. Если значение близко к количеству блоков, то таблица очень хорошо упорядочена. Если же записи распределены случайным образом, а значит, имеется большое CF значение, которое близко к количеству записей в таблице, то это плохой индекс. Oracle производит сканирование диапазона по структуре индекса, и если обнаруживается, что следующая строка рядом, то еще одна операция ввода-вывода для получения блока таблицы из буферного кеша не выполняется. Если следующая строка находится не в том же самом блоке, то Oracle освобождает этот блок и выполняет еще одну операцию ввода-вывода в буферном кеше, чтобы извлечь следующий блок для обработки.

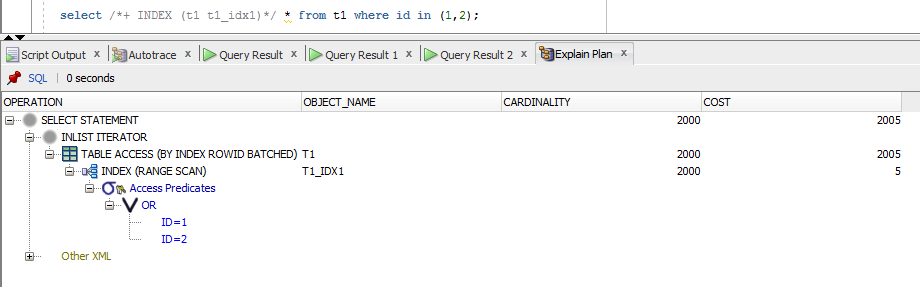
В нашем случае значения данных таблицы t2 для одного индексного ключа содержатся в одном блоке, поэтому Oracle посчитал для индекса t2\_idx1 отличный кластерный фактор. В случае таблицы t1 все с точностью наоборот, при обращении к данным таблицы через использование индекса t1\_idx1, Oracle придется прочитать для каждой извлекаемой строки блок данных.

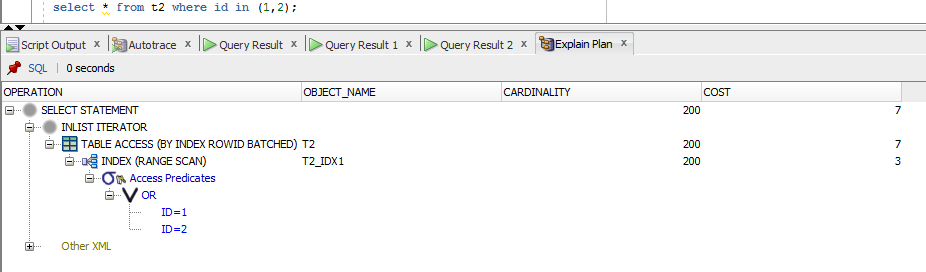
Ниже представлены планы запросов. Использование индекса t1\_idx1 в таблице t1 настолько нецелесообразно, что Oracle выбирает полное сканирование таблицы:



Я использовала хинт, чтобы посмотреть насколько все плохо. Видно, что cost при использовании индекса с кластерным фактором равном количеству строк почти в 5 раз выше, чем при FULL TABLE SCAN.



Cost запроса к данным таблицы t2 через индекс с кластерным фактором по значению близким к количеству блоков просто изумительный:

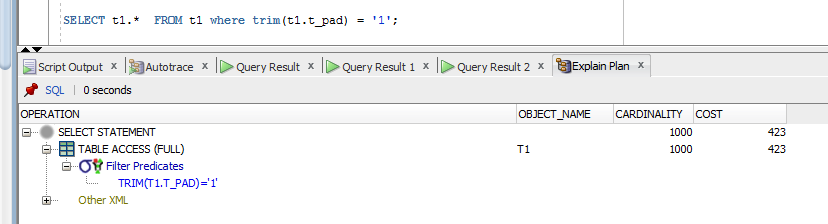


## Task 3: Index Unique Scan

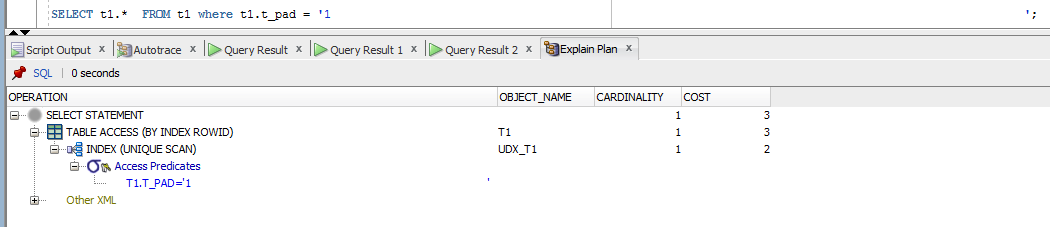
Результат запроса select t1.\* from t1 where t1.t\_pad = '1'; вернул ноль строк, т.к. данные в

t\_pad хранятся вместе с пробелами. Вариант с функцией trim не дал нам нужное

применение индекса, потому что функция должна лежать в основе создания индекса.

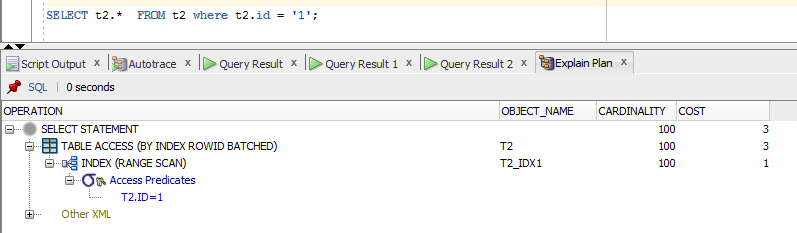


Чтобы получить необходимое использование индекса, используем следующий селект:



INDEX UNIQUE SCAN Oracle использует в случае, когда в условии where находится уникальный столбец или столбец PK из таблицы. Это гарантия того, что 1 записи индекса соответствует одна запись в таблице. Получается на шаге 2, Oracle читает индекс в поиске нужного нам t\_pad, опускаясь до нужного leaf, забирает rowid нужной нам записи и бережно идет с ней в таблицу t1, уверенный, что будет возвращена 1 запись.

## Task 4: Index Range Scan



Основная суть INDEX RANGE SCAN в том, что он используется при возвращении диапазона данных!

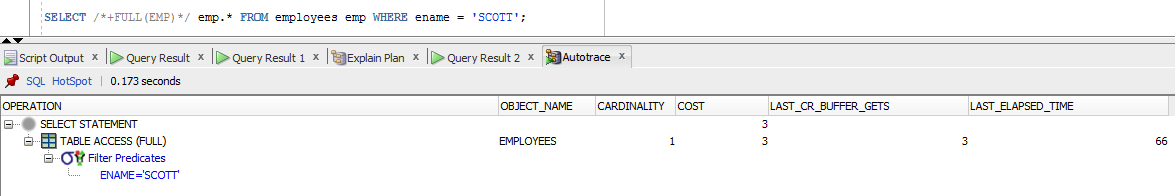
TABLE ACCESS (BY INDEX ROWID BATCHED) означает, что база данных извлекает несколько rowid из индекса, а затем пытается получить доступ к строкам в порядке блоков, чтобы улучшить производительность, сократив количество раз, которое база данных должна получить доступ к блоку.

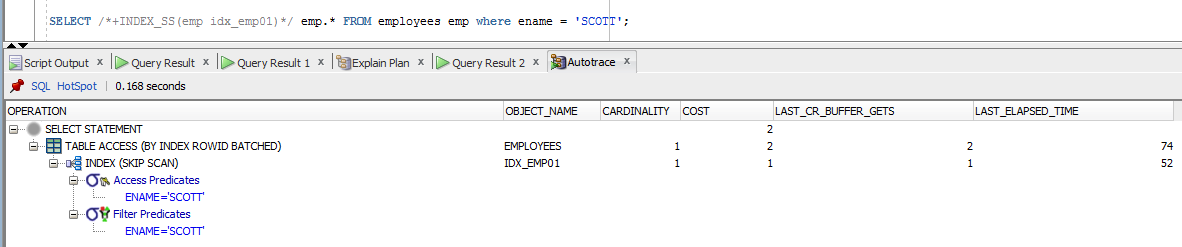
План же запроса, в котором за INDEX RANGE SCAN следует TABLE ACCESS BY INDEX ROWID, говорит о том, что Oracle прочитает индекс, после чего для его записей будет выполнять чтение блока БД, чтобы получить данные строк. Это не самый эффективный метод, если нужно получать доступ к большому проценту строк в таблице T через индекс.

В нашем случае, Oracle читает индекс в поисках записей с id=1, вытягивает все значения rowid для этого ключа и с ними идет в таблицу.

## Task 5: Index Skip Scan

CREATE INDEX idx\_emp01 ON employees ( empno, ename, job );





В первом запросе Oracle просто читает всю таблицу в буфер и выбирает нужные нам записи по ename=’SCOTT’.

Во втором запросе INDEX SKIP SCAN (по ename) говорит о том, что Oracle бегло просматривает индекс в поисках мест, где столбец empno меняет значение, и читает вниз по дереву с этого места в поисках ename = ’SCOTT’ в каждом рассматриваемом субиндексе. Этот происходит в случаях, когда индекс составной и поиск производится по второму столбцу индекса, при этом значения первого столбца в индексе должны быть минимально уникальными, т.е. должно быть много повторяющихся значений. В противном случае Oracle посчитает, что быстрее выполнить FULL TABLE SCAN. Работает аналогично и при большем количестве столбцов в составном индексе.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 8 | 1 | 14 | 2 | Index |
| 2 | 8 | 1 | 14 | 3 | Full |

При выполнении наших запросов при использовании индекса Oracle понадобилось запросить только 2 блока (consistent gets) из буферного кеша, тогда как при полном просмотре таблицы – 3 блока.