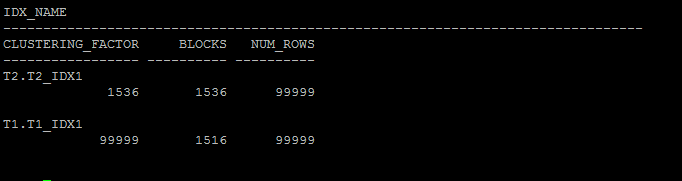
**Labwork 4**

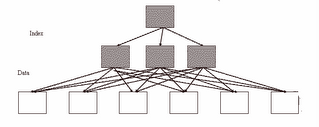
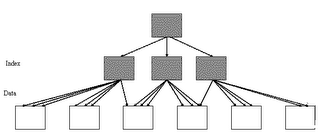
## Task 1: Full Scans and the High-water Mark and Block reading

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1623 | We insert 99999 rows, which need 1536 blocks. But DB allocate blocks by extends discretely. And also we have large amount of consistent gets1. |
| 2 | 1664 | 0 | 0 | 1541 | After deleting all rows, we don’t use any block, but it count is still 1664, because the fixed for the table. Consistent gets are still high due to the fact that HWM saved the same value for the next insert in block. |
| 3 | 1664 | 1 | 1 | 1541 | We insert 1 row and so use 1 block. Count of Blocks and count of Consistent gets review below. |
| 4 | 8 | 0 | 0 | 3 | We truncate table count of blocks became smaller, and HWM became default so consistent gets became 3. |

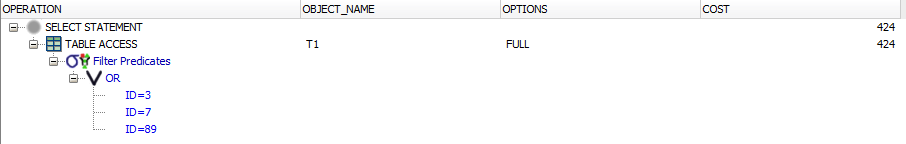
1 The consistent gets Oracle metric is the number of times a consistent read (a logical RAM buffer I/O) was requested to get data from a data block. Part of Oracle tuning is to increase logical I/O by reducing the expensive disk I/O (physical reads), but high consistent gets presents it's own tuning challenges, especially when we see super high CPU consumption (i.e. the "top 5 timed events" in an AWR report).

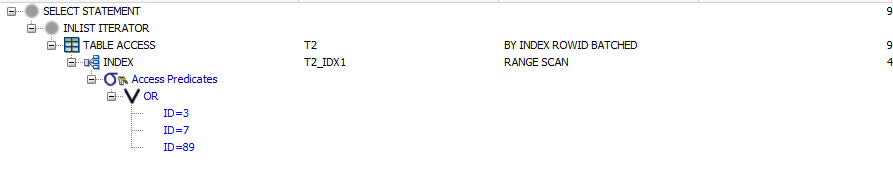
## Index Clustering factor parameter



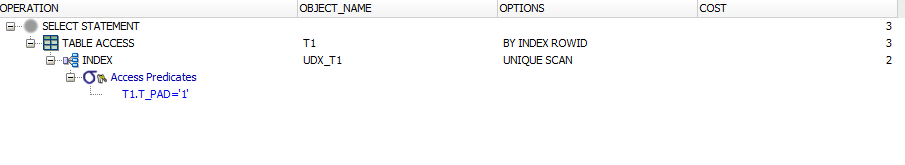
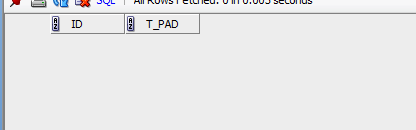
The clustering factor is a number which represent the degree to which data is randomly distributed in a table.  
  
In simple terms it is the number of “block switches” while reading a table using an index.  
  
[](http://photos1.blogger.com/blogger2/7814/275520293397574/1600/Bad_CF.gif)  
Figure: Bad clustering factor  
  
The above diagram explains that how scatter the rows of the table are. The first index entry (from left of index) points to the first data block and second index entry points to second data block. So while making index range scan or full index scan, optimizer have to switch between blocks and have to revisit the same block more than once because rows are scatter. So the number of times optimizer will make these switches is actually termed as “Clustering factor”.  
  
[](http://photos1.blogger.com/blogger2/7814/275520293397574/1600/Good_CF.gif)  
  
Figure: Good clustering factor  
  
The above image represents "Good CF”. In an event of index range scan, optimizer will not have to jump to next data block as most of the index entries points to same data block. This helps significantly in reducing the cost of your SELECT statements.

For reason of different data storage in tables t1 and t2, optimizer choose full table access for t1 and index in t2, that make cost of select statement for t1 more less then for t2.





## Index Unique Scan



An index unique scan is chosen when a predicate contains a condition using a column defined with a

UNIQUE or PRIMARY KEY index. These types of indexes guarantee that only one row will ever be returned

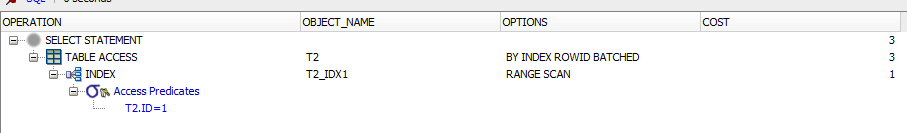
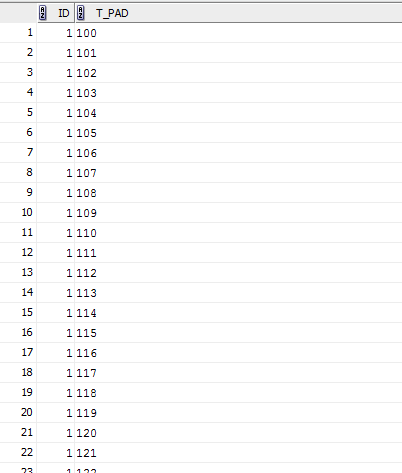
for a specified value. In this cases, the index structure will be traversed from root to leaf block to a

single entry, retrieve the rowid, and use it to access the table data block containing the one row. The TABLE ACCESS BY INDEX ROWID step in the plan indicates the table data block access. The number of

block accesses required will always be equal to the height of the index plus one unless there are special

circumstances like the row is chained or contains a LOB that is stored elsewhere.

**Index Range Scan**

****

An index range scan is chosen when a predicate contains a condition that will return a range of data.

The index can be unique or non-unique as it is the condition that determines whether or not multiple

rows will be returned or not. The conditions specified can use operators such as <, >, LIKE, BETWEEN and

even =. In order for a range scan to be selected, the range will need to be fairly selective.

A range scan will traverse the index structure from the root block to the first leaf block containing

an entry matching the specified condition. From that starting point, a rowid will be retrieved from the

index entry and the table data block will be retrieved (TABLE ACCESS BY INDEX ROWID). After the first

row is retrieved, the index leaf block will be accessed again and the next entry will be read to retrieve

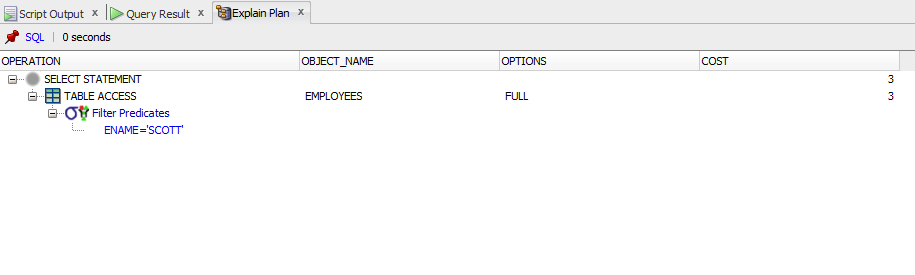
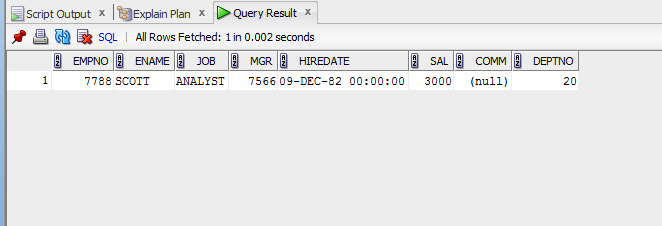
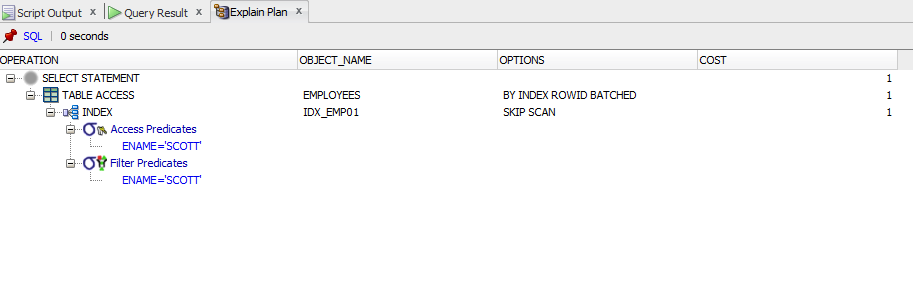
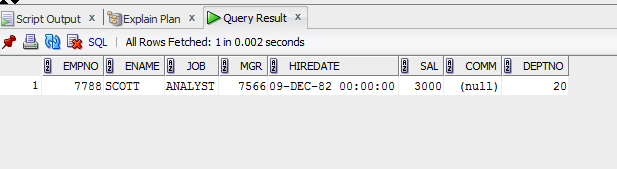
the next rowid. This back-and-forth between the index leaf blocks and the data blocks will continue

until all the matching index entries have been read. Therefore, the number of block accesses required

will include the number of branch blocks in the index (this can be found using the blevel statistic for

the index) plus the number of index entries that match the condition multiplied by two.

## Index Skip Scan



An index skip scan is chosen when the predicate contains a condition on a non-leading column in an

index and the leading columns are fairly distinct.

A skip scan works by logically

splitting a multi-column index into smaller subindexes. The number of logical subindexes is

determined by the number of distinct values in the leading columns of the index.