1.

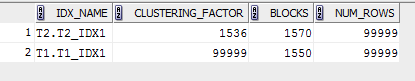
Summary table with all result and text description of analyses this results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1539 | Table is full, apparently we need 1539 requests to buffer cache to calculate count(\*) |
| 2 | 1664 | 0 | 0 | 1541 | All rows were deleted, but blocks are still occupied. Still 1541 gets to make sure all of blocks are not containing data. |
| 3 | 1664 | 1 | 1 | 1541 | INSERT 1 block so it’s 1 block used. Still reading 1541 because they are assigned to table. |
| 4 | 8 | 0 | 0 | 3 | TRUNCATE marked all blocks empty so it’s not needed anymore to read through them to calculate zero count of rows.  Only 1 block is occupied; |

Consistent gets – number of times a consistent read was requested for a block in the buffer cache.

2.

Screenshot of the step 5:



Description of the parameter clustering factor:

Defines how ordered the rows are in the index. If CLUSTERING\_FACTOR approaches the

number of blocks in the table, the rows are ordered. If it approaches the number of rows

in the table, the rows are randomly ordered. In such a case (clustering factor near the

number of rows), it is unlikely that index entries in the same leaf block will point to

rows in the same data blocks.

Note that typically only 1 index per table will be heavily clustered (if any). It would

be extremely unlikely for 2 indexes to be very clustered.

If you want an index to be very clustered -- consider using index organized tables. They

force the rows into a specific physical location based on their index entry.

Otherwise, a rebuild of the table is the only way to get it clustered (but you really

don't want to get into that habit for what will typically be of marginal overall

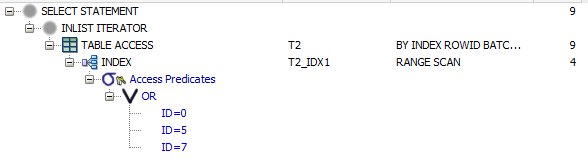
improvement).

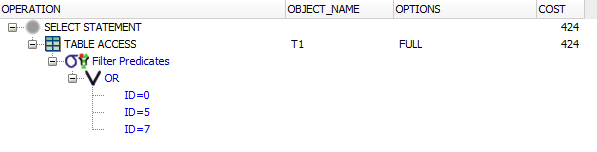
Explanation: why for indexes t1\_idx1 and t2\_idx1 we have different values:

Because id row on which we build indexes was generated in different ways. On table t2 we have id growing up in series, on the other hand, on table t1 ID values are pretty random.

Which Index has best selective performance in execution Select clause filtered by IN ( , list of values, ):

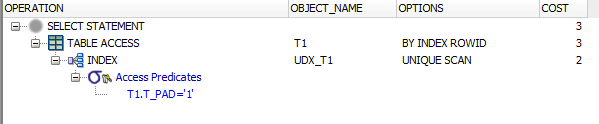
Index on table T2, because of using range scan access method.





3.

Screenshot of the step 2:

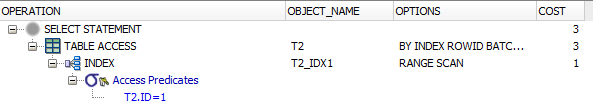


Description of process: How oracle read block on step 1:

Database used an index unique scan to locate the rowid for selected ID and read appropriate block.

4.

Screenshot of the step 2:

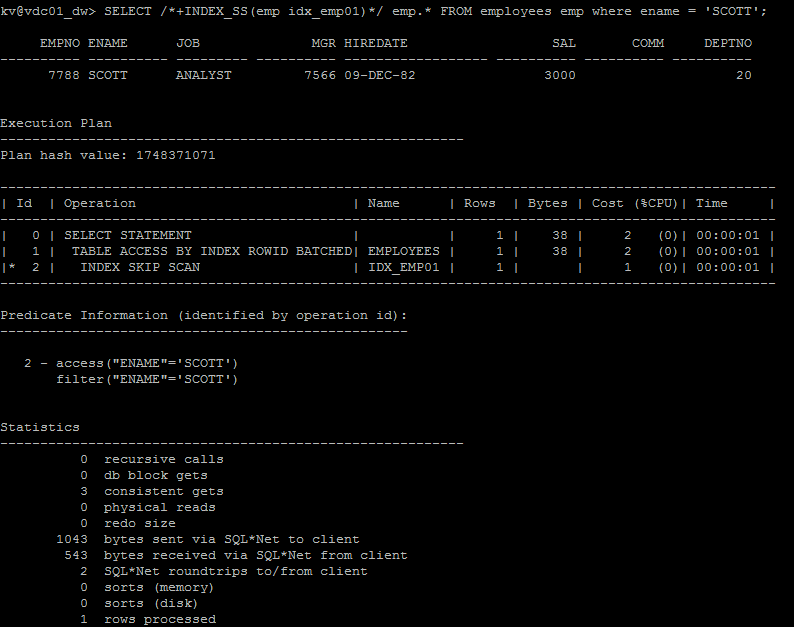


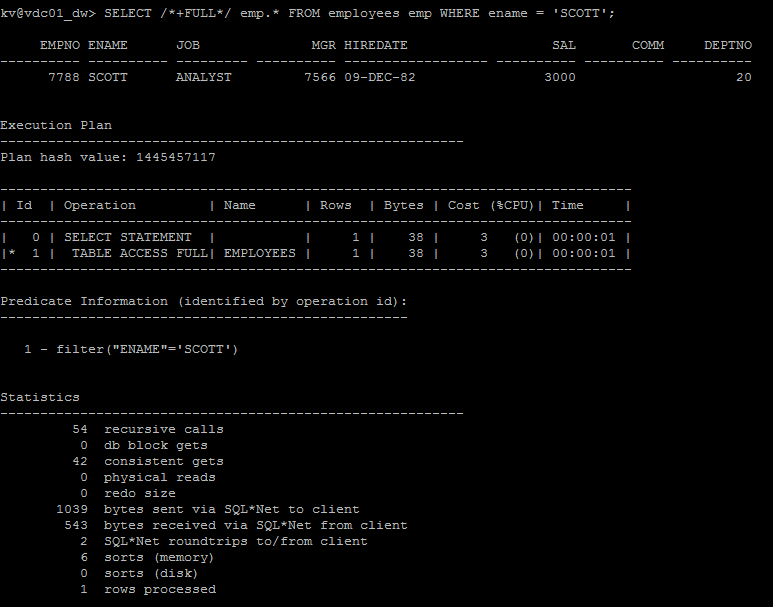
Description of process: How oracle read block on step 1:

Oracle consider that all values with T2.ID =1 are stored together and use Range Scan index to get rowed value.

5.

2 Screenshots of the step 3:





Description of process: How oracle analyses index that was created on step 2:

Oracle excludes EMPNO from scan and use resulting sub indexes only for ENAME values.