**Understanding Indexes**

**What is an Index?**

This is covered in the Oracle Concepts manual, of course, but here's the Cliff Notes version.

**Blocks**

First you need to understand a **block**. A block - or **page** for Microsoft boffins - is the smallest unit of disk that Oracle will read or write. All data in Oracle - tables, indexes, clusters - is stored in blocks. The block size is configurable for any given database but is usually one of 4Kb, 8Kb, 16Kb, or 32Kb. Rows in a table are usually much smaller than this, so many rows will generally fit into a single block. So you never read "just one row"; you will always read the entire block and ignore the rows you don't need. Minimising this wastage is one of the fundamentals of Oracle Performance Tuning.

Oracle uses two different index architectures: b-Tree indexes and bitmap indexes. Cluster indexes, bitmap join indexes, function-based indexes, reverse key indexes and text indexes are all just variations on the two main types. b-Tree is the "normal" index, so we will come back to Bitmap indexes another time.

**The "-Tree" in b-Tree**

A b-Tree index is a data structure in the form of a tree - no surprises there - but it is a tree of database *blocks*, not rows. Imagine the leaf blocks of the index as the pages of a phone book.

* Each page in the book (leaf block in the index) contains many entries, which consist of a name (indexed column value) and an address (ROWID) that tells you the physical location of the telephone (row in the table).
* The names on each page are sorted, and the pages - when sorted correctly - contain a [COMPLETE[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) sorted list of every name and address

A sorted list in a phone book is fine for humans, beacuse we have mastered "the flick" - the ability to fan through the book looking for the page that will contain our target *without reading the entire page*. When we flick through the phone book, we are just reading the first name on each page, which is usually in a larger font in the page header. Oracle cannot read a single name (row) and ignore the reset of the page (block); it needs to read the entire block.

If we had no thumbs, we may find it convenient to create a separate ordered list containing the first name on each page of the phone book along with the page number. This is how the branch-blocks of an index work; a reduced list that contains the first row of each block plus the address of that block. In a large phone book, this reduced list containing one entry per page will still cover many pages, so the process is repeated, creating the next level up in the index, and so on until we are left with a single page: the *root* of the tree.

To find the name *Gallileo* in this b-Tree phone book, we:

* Read page 1. This tells us that page 6 starts with *Fermat* and that page 7 starts with *Hawking*.
* Read page 6. This tells us that page 350 starts with *Fyshe* and that page 351 starts with *Garibaldi*.
* Read page 350, which is a leaf block; we find Gallileo's address and phone number.

That's it; 3 blocks to find a specific row in a million row table. In reality, index blocks often fit 100 or more rows, so b-Trees are typically quite shallow. I have never seen an index with more than 5 levels. Curious? Try this:

SELECT index\_name, blevel+1 FROM user\_indexes ORDER BY 2;

user\_indexes.blevel is the number of branch levels. Always add 1 to include the leaf level; this tells you the number of blocks a unique index [SCAN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) must read to reach the leaf-block. If you're really, really, insatiably curious; try this in SQL\*Plus:

ACCEPT index\_name PROMPT "Index Name: "

ALTER SESSION SET TRACEFILE\_IDENTIFIER = '&index\_name';

COLUMN object\_id NEW\_VALUE object\_id

SELECT object\_id

FROM user\_objects

WHERE object\_type = 'INDEX'

AND object\_name = upper('&index\_name');

ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME TREEDUMP LEVEL &object\_id';

ALTER SESSION SET TRACEFILE\_IDENTIFIER = "";

SHOW PARAMETER user\_dump\_dest

Give the name of an index on a smallish table (because this will create a BIG file). Now, on the Oracle server, go to the directory shown by the final SHOW PARAMETER user\_dump\_dest command and find your trace file - the file name will contain your index name. Here is a sample:

\*\*\* 2007-01-31 11:51:26.822

----- begin tree dump

branch: 0x68066c8 109078216 (0: nrow: 325, level: 1)

leaf: 0x68066c9 109078217 (-1: nrow: 694 rrow: 694)

leaf: 0x68066ca 109078218 (0: nrow: 693 rrow: 693)

leaf: 0x68066cb 109078219 (1: nrow: 693 rrow: 693)

leaf: 0x68066cc 109078220 (2: nrow: 693 rrow: 693)

leaf: 0x68066cd 109078221 (3: nrow: 693 rrow: 693)

...

...

leaf: 0x68069cf 109078991 (320: nrow: 763 rrow: 763)

leaf: 0x68069d0 109078992 (321: nrow: 761 rrow: 761)

leaf: 0x68069d1 109078993 (322: nrow: 798 rrow: 798)

leaf: 0x68069d2 109078994 (323: nrow: 807 rrow: 807)

----- end tree dump

This index has only a root branch with 323 leaf nodes. Each leaf node contains a variable number of index entries up to 807! A deeper index would be more interesting, but it would take a while to dump.

**"B" is for...**

Contrary to popular belief, *b* is not for *binary*; it's *balanced*.

As you insert new rows into the table, new rows are inserted into index leaf blocks. When a leaf block is full, another insert will cause the block to be split into two blocks, which means an entry for the new block must be added to the parent branch-block. If the branch-block is also full, it too is split. The process propagates back up the tree until the parent of split has space for one more entry, or the root is reached. A new root is created if the root node splits. Staggeringly, this process ensures that every branch will be the same length. Try it on paper for yourself!

**How are Indexes used?**

Indexes have three main uses:

* To quickly find specific rows by avoiding a Full Table [SCAN](http://www.orafaq.com/node/1403)

We've already seen above how a Unique Scan works. Using the phone book metaphor, it's not hard to understand how a Range Scan works in much the same way to find all people named "Gallileo", or all of the names alphabetically between "Smith" and "Smythe". Range Scans can occur when we use >, <, LIKE, or BETWEEN in a WHERE clause. A range scan will find the first row in the range using the same technique as the Unique Scan, but will then keep reading the index up to the end of the range. It is OK if the range covers many blocks.

* To avoid a table [ACCESS](http://www.orafaq.com/node/1403) altogether

If all we wanted to do when looking up Gallileo in the phone book was to find his address or phone number, the job would be done. However if we wanted to know his [DATE](http://www.orafaq.com/node/1403) of birth, we'd have to phone and ask. This takes time. If it was something that we needed all the time, like an email address, we could save time by adding it to the phone book.

Oracle does the same thing. If the information is in the index, then it doesn't bother to read the table. It is a reasonably common technique to add columns to an index, not because they will be used as part of the index scan, but because they save a table access. In fact, Oracle may even perform a Fast Full Scan of an index that it cannot use in a Range or Unique scan just to avoid a table access.

* To avoid a sort

This one is not so well known, largely because it is so poorly documented (and in many cases, unpredicatably implemented by the Optimizer as well). Oracle performs a sort for many reasons: ORDER BY, GROUP BY, DISTINCT, Set operations (eg. UNION), Sort-Merge Joins, uncorrelated IN-subqueries, Analytic Functions). If a sort operation requires rows in the same order as the index, then Oracle may read the table rows via the index. A sort operation is not necessary since the rows are returned in sorted order.

Despite all of the instances listed above where a sort is performed, I have only seen three cases where a sort is actually avoided.

* 1. **GROUP BY**
  2. 1 select src\_sys, sum(actl\_expns\_amt), count(\*)
  3. 2 from ef\_actl\_expns
  4. 3 where src\_sys = 'CDW'
  5. 4 and actl\_expns\_amt > 0
  6. 5\* group by src\_sys
  7. -------------------------------------------------------------
  8. | Id | Operation | Name |
  9. -------------------------------------------------------------
  10. | 0 | SELECT STATEMENT | |
  11. | 1 | SORT GROUP BY NOSORT | |
  12. |\* 2 | TABLE ACCESS BY GLOBAL INDEX ROWID| EF\_ACTL\_EXPNS |
  13. |\* 3 | INDEX RANGE SCAN | EF\_AEXP\_PK |
  14. -------------------------------------------------------------
  15. Predicate Information (identified by operation id):
  16. ---------------------------------------------------
  17. 2 - filter("ACTL\_EXPNS\_AMT">0)

3 - access("SRC\_SYS"='CDW')

Note the NOSORT qualifier in Step 1.

* 1. **ORDER BY**
  2. 1 select \*
  3. 2 from ef\_actl\_expns
  4. 3 where src\_sys = 'CDW'
  5. 4 and actl\_expns\_amt > 0
  6. 5\* order by src\_sys
  7. ------------------------------------------------------------
  8. | Id | Operation | Name |
  9. ------------------------------------------------------------
  10. | 0 | SELECT STATEMENT | |
  11. |\* 1 | TABLE ACCESS BY GLOBAL INDEX ROWID| EF\_ACTL\_EXPNS |
  12. |\* 2 | INDEX RANGE SCAN | EF\_AEXP\_PK |
  13. ------------------------------------------------------------
  14. Predicate Information (identified by operation id):
  15. ---------------------------------------------------
  16. 1 - filter("ACTL\_EXPNS\_AMT">0)
  17. 2 - access("SRC\_SYS"='CDW')

Note that there is no SORT operation, despite the ORDER BY clause. Compare this to the following:

1 select \*

2 from ef\_actl\_expns

3 where src\_sys = 'CDW'

4 and actl\_expns\_amt > 0

5\* order by actl\_expns\_amt

-------------------------------------------------------------

| Id | Operation | Name |

-------------------------------------------------------------

| 0 | SELECT STATEMENT | |

| 1 | SORT ORDER BY | |

|\* 2 | TABLE [ACCESS[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) BY GLOBAL INDEX ROWID| EF\_ACTL\_EXPNS |

|\* 3 | INDEX RANGE [SCAN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) | EF\_AEXP\_PK |

-------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - filter("ACTL\_EXPNS\_AMT">0)

3 - access("SRC\_SYS"='CDW')

* 1. **DISTINCT**
  2. 1 select distinct src\_sys
  3. 2 from ef\_actl\_expns
  4. 3 where src\_sys = 'CDW'
  5. 4\* and actl\_expns\_amt > 0
  6. -------------------------------------------------------------
  7. | Id | Operation | Name |
  8. -------------------------------------------------------------
  9. | 0 | SELECT STATEMENT | |
  10. | 1 | SORT UNIQUE NOSORT | |
  11. |\* 2 | TABLE ACCESS BY GLOBAL INDEX ROWID| EF\_ACTL\_EXPNS |
  12. |\* 3 | INDEX RANGE SCAN | EF\_AEXP\_PK |
  13. -------------------------------------------------------------
  14. Predicate Information (identified by operation id):
  15. ---------------------------------------------------
  16. 2 - filter("ACTL\_EXPNS\_AMT">0)

3 - access("SRC\_SYS"='CDW')

Again, note the NOSORT qualifier.

This is an extraordinary tuning technique in OLTP systems like SQL\*Forms that return one page of detail at a time to the screen. A SQL with a DISTINCT, GROUP BY, or ORDER BY that uses an index to sort can return just the first page of matching rows *without having to fetch the entire result set* for a sort. This can be the difference between sub-second response time and several minutes or hours.

**Everybody repeat after me: "Full table Scans are not bad"**

Up to now, we've seen how indexes can be good. It's not always the case; sometimes indexes are no help at all, or worse: they make a query *slower*.

A b-Tree index will be no help at all in a reduced [SCAN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) unless the WHERE clause compares indexed columns using >, <,LIKE, IN, or BETWEEN operators. A b-Tree index cannot be used to scan for any NOT style operators: eg. !=, NOT IN, NOT LIKE. There are lots of conditions, caveats, and complexities regarding joins, sub-queries, OR predicates, functions (inc. arithmetic and concatenation), and casting that are outside the scope of this article. [CONSULT[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) a good SQL tuning manual.

Much more interesting - and important - are the cases where an index makes a SQL *slower*. These are particularly common in batch systems that process large quantities of data.

To explain the problem, we need a new metaphor. Imagine a large deciduous tree in your front yard. It's Autumn, and it's your job to pick up all of the leaves on the lawn. Clearly, the fastest way to do this (without a rake, or a leaf-vac...) would be get down on hands and knees with a bag and work your way back and forth over the lawn, stuffing leaves in the bag as you go. This is a Full Table Scan, selecting rows in no particular order, except that they are nearest to hand. This metaphor works on a couple of levels: you would grab leaves in handfuls, not one by one. A Full Table Scan does the same thing: when a bock is read from disk, Oracle caches the next few blocks with the expectation that it will be asked for them very soon. Type this in SQL\*Plus:

SHOW PARAMETER db\_file\_multiblock\_read\_count

Just to shake things up a bit (and to feed an undiagnosed obsessive compulsive disorder), you decide to pick up the leaves in order of size. In support of this endeavour, you take a digital photograph of the lawn, write an image analysis [PROGRAM[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) to identify and measure every leaf, then load the results into a Virtual Reality headset that will highlight the smallest leaf left on the lawn. Ingenious, yes; but this is clearly going to take a lot longer than a full table scan because you cover much more distance walking from leaf to leaf.

So obviously Full Table Scan is the faster way to pick up *every* leaf. But just as obvious is that the index (virtual reality headset) is the faster way to pick up *just the smallest leaf*, or even the 100 smallest leaves. As the number rises, we approach a break-even point; a number beyond which it is faster to just full table scan. This number varies depending on the table, the index, the database settings, the hardware, and the load on the server; generally it is somewhere between 1% and 10% of the table.

The main reasons for this are:

* 1. As implied above, reading a table in indexed order means more movement for the disk head.
  2. Oracle cannot read single rows. To read a row via an index, the entire block must be read with all but one row discarded. So an index scan of 100 rows would read 100 blocks, but a FTS might read 100 rows in a single block.
  3. The db\_file\_multiblock\_read\_count setting described earlier means FTS requires fewer visits to the physical disk.
  4. Even if none of these things was true, [ACCESSING[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) the entire index *and* the entire table is still more IO than just accessing the table.

So what's the lesson here? Know your data! If your query needs 50% of the rows in the table to resolve your query, an index scan just won't help. Not only should you not bother creating or investigating the existence of an index, you should *check to make sure* Oracle is not already using an index. There are a number of ways to influence index usage; once again, [CONSULT[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.orafaq.com/node/1403)](http://www.orafaq.com/node/1403) a tuning manual. The exception to this rule - there's always one - is when all of the columns referenced in the SQL are contained in the index. If Oracle does not have to access the table then there is no break-even point; it is generally quicker to scan the index even for 100% of the rows.

**Summary**

Indexes are not a dark-art; they work in an entirely predictable and even intuitive way. Understanding how they work moves Performance Tuning from the realm of guesswork to that of science; so embrace the technology and read the manual.