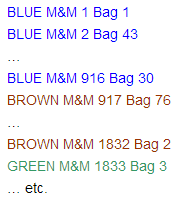
Finding All the Red M&Ms: A Story of Indexes and Full‑Table Scans

*In this guest post,* [*Chris Saxon*](http://www.chrissaxon.co.uk/) *explains a very important topic using an analogy with chocolates: When does a database use an index and when is it better not using it. Although Chris explanation has the Oracle database in mind, the principles apply to other databases too.*

A common question that comes up when people start tuning queries is “why doesn’t this query use the index I expect?”. There are a few myths surrounding when database optimizers will use an index. A common one I’ve heard is that an index will be used when accessing 5% or less of the rows in a table. This isn’t the case however - the basic decision on whether or not to use an index comes down to its cost.

How do databases determine the cost of an index?

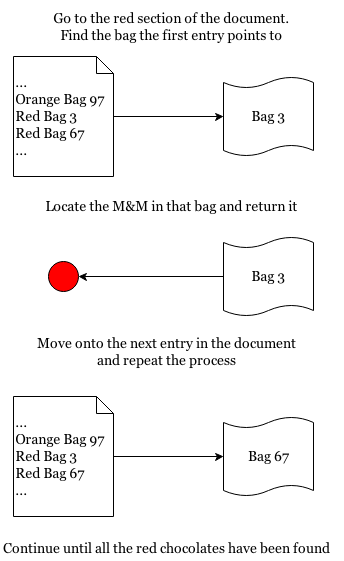
Before getting into the details, let's talk about chocolate! Imagine you have 100 packets of M&M's. You also have a document listing the colour of each M&M and the bag it’s stored in. This is ordered by colour, so we have all the blue sweets first, then the brown, green and so on like so:



You’ve been asked to find all the red M&M's. There’s a couple of basic ways you could approach this task:

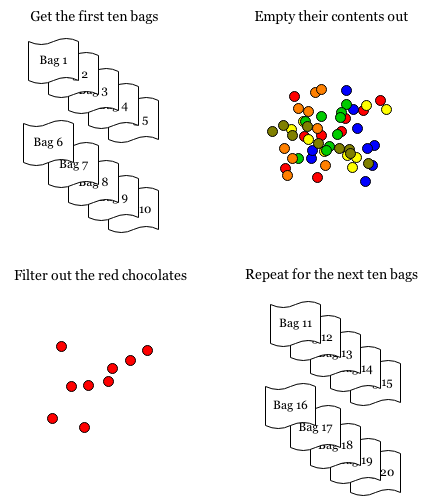
Method 1

Get your document listing the colour and location of each M&M. Go to the top of the “red” section. Lookup the location of the first red M&M, pick up the bag it states, and get the sweet. Go back to your document and repeat the process for the next red chocolate. Keep going back-and-forth between your document and the bags until you’ve reached the end of the red section.



Method 2

Pick up a number of bags at a time (e.g. 10), empty their contents out, pick out the red chocolates and return the others (back to their original bag).



Which approach is quicker?

Intuitively the second approach appears to be faster. You only select each bag once and then do some filtering of the items inside. Whereas with the first approach you have done a lot of back-and-forth between the document and the bags. This means you have to look into each bag multiple times.

We can be a bit more rigorous than this though. Let’s calculate how many operations we need to do to get all the red chocolates in each case.

When going between the document and the bags (method 1), each time you lookup the location of a new sweet and fetch that bag that’s a new operation. You have 100 bags with around 55 sweets in each. This means you’re doing roughly 920 (100 bags x 55 sweets / 6 colours) operations (plus some work to find the red section in your document). So the “cost” of using the document is around 920.

With the second approach you collect 10 bags in one step. This means you do ( 100 bags / 10 bags per operation = ) 10 operations (plus some filtering of the chocolates in them), giving a “cost” of 10.

Comparing these costs (920 vs. 10), method 2 is the clear winner.

Let’s imagine another scenario. Mars have started doing a promotion where around 1 in 100 bags contain a silver M&M. If you get the silver sweet, you win a prize. You want to find the silver chocolate!

In this case, using method 1, you go to the document to find the location of the single sweet. Then you go to that bag and retrieve the sweet. One operation (well two, including going to the document to find location of the silver chocolate), so we have a cost of two.

With method 2, you still need to pick up every single bag (and do some filtering) just to find one sweet - the cost is fixed at 10. Clearly method 1 is far superior in this case.

What have M&M's got to do with databases?

When Oracle stores a record to the database, it is placed in a block. Just like there are many M&Ms in a bag, (normally) there are many rows in a block. When accessing a particular row, Oracle fetches the whole block and retrieves the requested row from within it. This is analogous to us picking up a bag of M&Ms and then picking a single chocolate out.

When doing an index-range scan, Oracle will search the index (the document) to find the first value matching your where clause. It then goes back-and-forth between the index and the table blocks, fetching the records from the location pointed to by the index. This is similar to method 1, where you continually switch between the document and the M&M bags.

As the number of rows accessed by an index increases, the database has to do more work. Therefore the cost of using an index increases in line with the number of records it is expected to fetch.

When doing a full table scan (FTS), Oracle will fetch several blocks at once in a *multi-block read*. The data fetched is then filtered so that only rows matching your where clause are returned (in this case the red M&Ms) – the rest are discarded. Just like in method 2.

The expected number of rows returned has little impact on the work a FTS does. Its basic cost is fixed by the size of the table and how many blocks you fetch at once.

When fetching a “high” percentage of the rows from a table, it becomes far more efficient to get several blocks at once and do some filtering than it is to visit a single block multiple times.

When does an index scan become more efficient than a FTS?

In our M&M example above, the “full-table scan” method fetches all 100 bags in 10 operations. Whereas with “index” approach requires a separate operation for each sweet. So an index is more efficient when it points to 10 M&Ms or less.

Mars puts around 55 M&M's in each bag, so as a percentage of the “table” that’s just under ( 10 M&M's / (100 bags \* 55 sweets) \* 100 = ) 0.2%!

What if Mars releases some “giant” M&Ms with only 10 sweets in a bag? In this case there’s fewer sweets in total, so the denominator in the equation above decreases. Our FTS approach is still fixed at a “cost” of 10 for the 100 bags. This means the point at which an index is better is when accessing approximately ( 10/1000\*100 = ) 1% of the “rows”. A higher percentage, but still small in real terms.

If they released “mini” M&Ms with 200 in a bag, the denominator would increase. This means that the index is more efficient when accessing a very small percentage of the table!

So as you increase the space required to store a row, an index becomes more effective than a FTS. The number of rows accessed by the index remains fixed. The number of blocks required to store the data increases however, making the FTS more expensive and leading to it having a higher cost.

There’s a big assumption made in the above reasoning however. It’s that there’s no correlation between the order M&M's are listed in the document and which bag they are in. So, for example, the first red M&M (in the document) may be in bag 1, the second in bag 56, the third in bag 20, etc.

Let’s make a different assumption – that the order of red chocolates in the document corresponds to the order they appear in the bags. So the first 9 red sweets are in bag 1, the next 9 in bag 2 etc. While you still have to visit all 100 bags, you can keep the last bag accessed in your hand, only switching bags every 9 or so sweets. This reduces the number of operations you do, making the index approach more efficient.

We can take this further still. What if Mars changes the bagging process so that only one colour appears in each bag?

Now, instead of having to visit every single bag to get all the red sweets, you only have to visit around ( 100 bags / 6 colours) 16 bags. If the sweets are also placed in the bags in the same order they are listed in the document (so M&M's 1-55 are all blue and in bag 1, bag 2 has the blue M&M's 56-100, and so on until bag 100, which has yellow M&M's 496-550) you get the benefits of not switching bags compounded with the effect of having fewer bags to fetch.

This principle – how closely the order of records in a table matches the order they’re listed in a corresponding index – is referred to as the *clustering factor*. This figure is lower when the rows appear in the same physical order in the table as they do in the index (all sweets in a bag are the same colour) and higher when there’s little or no correlation.

The closer the clustering factor is to the number of blocks in a table the more likely it is that the index will be used (it is assigned a lower cost). The closer it is to the number of rows in a table, the more likely it is a FTS will be chosen (the index access is given a higher cost).

Bringing it all together

To sum up, we can see the cost-based optimizer decides whether to use an index or FTS by:

* Taking the number of blocks used to store the table and dividing this by the number of blocks read in a multi-block read to give the FTS cost.
* For each index on the table available to the query:
  + Finding the percentage of the rows in the table it expects a query to return (the selectivity)
  + This is then used to determine the percentage of the index expected to be accessed
  + The selectivity is also multiplied by the clustering factor to estimate the number of table blocks it expects to access to fetch these rows via an index
  + Adding these numbers together to give the expected cost (of the index)
* The cost of the FTS is then compared to each index inspected and the access method with the lowest cost used.

This is just an overview of how the (Oracle) cost-based optimizer works. If you want to see the formulas the optimizer uses have a read of Wolfgang Breitling’s “[Fallacies of the Cost Based Optimizer](http://www.centrexcc.com/Fallacies%20of%20the%20Cost%20Based%20Optimizer.pdf)” paper or Jonathan Lewis’ [Cost-Based Oracle Fundamentals](https://www.amazon.co.uk/Cost-Based-Oracle-Fundamentals-Experts-Voice/dp/1590596366/ref=sr_1_1?ie=UTF8&qid=1502472304&sr=8-1&keywords=Cost-Based+Oracle+Fundamentals) book. The blogs of [Jonathan Lewis](https://jonathanlewis.wordpress.com/), [Richard Foote](https://richardfoote.wordpress.com/) and of course Markus' articles on [this site](https://use-the-index-luke.com/) also contain many posts going into this subject in more detail.

About the Author



Chris Saxon is an Oracle Developer Advocate for SQL. His job is to help you get the best out of the Oracle Database and have fun with SQL!  
To help you with this he blogs at [All Things SQL](https://blogs.oracle.com/sql/) and creates videos on [The Magic of SQL](https://www.youtube.com/c/TheMagicofSQL).