How do databases work internally?

I've been working with databases for the last few years and I'd like to think that I've gotten fairly competent with using them. However I was reading recently about Joel's Law of Leaky Abstractions and I realised that even though I can write a query to get pretty much anything I want out of a database, I have no idea how the database actually interprets the query. Does anyone know of any good articles or books that explain how databases work internally?

Some specific things I'm interested in are:

- What does a database actually do to find out what matches a select statement?
- How does a database interpret a join differently to a query with several "where key1 = key2" statements?
- How does the database store all its memory?
- How are indexes stored?

database reference internals

edited Oct 11 '08 at 5:23

Gulzar Nazim
42.4k 23 115 166

asked Oct 6 '08 at 0:47

Bonnici
696 2 8 19

+1 for the providing the link for Law of Leaky Abstractions. – Sorter Nov 20 '12 at 6:15

1 As of 2015, there is a this article which seems pretty good. – Piovezan Jul 7 '16 at 13:16

7 Answers

> What does a database actually do to find out what matches a select statement?

To be blunt, it's a matter of brute force. Simply, it reads through each candidate record in the database and matches the expression to the fields. So, if you have "select * from table where name = 'fred'", it literally runs through each record, grabs the "name" field, and compares it to 'fred'.

Now, if the "table.name" field is indexed, then the database will (likely, but not necessarily) use the index first to locate the candidate records to apply the actual filter to.

This reduces the number of candidate records to apply the expression to, otherwise it will just do what we call a "table scan", i.e. read every row.

But fundamentally, however it locates the candidate records is separate from how it applies the actual filter expression, and, obviously, there are some clever optimizations that can be done.

> How does a database interpret a join differently to a query with several "where key1 = key2" statements?

Well, a join is used to make a new "pseudo table", upon which the filter is applied. So, you have the filter criteria and the join criteria. The join criteria is used to build this "pseudo table" and then the filter is applied against that. Now, when interpreting the join, it's again the same issue as the filter -- brute force comparisons and index reads to build the subset for the "pseudo table".

> How does the database store all its memory?

One of the keys to good database is how it manages its I/O buffers. But it basically matches RAM blocks to disk blocks. With the modern virtual memory managers, a simpler database can almost rely on the VM as its memory buffer manager. The high end DB'S do all this themselves.

> How are indexes stored?

B+Trees typically, you should look it up. It's a straight forward technique that has been around for years. It's benefit is shared with most any balanced tree: consistent access to the nodes, plus all the leaf nodes are linked so you can easily traverse from node to node in key order. So, with an index, the rows can be considered "sorted" for specific fields in the database, and the database can leverage that information to it benefit for optimizations. This is distinct from, say, using a hash table for an index, which only lets you get to a specific record quickly. In a B-Tree you can quickly get not just to a specific record, but to a point within a sorted list.

The actual mechanics of storing and indexing rows in the database are really pretty straight forward and well understood. The game is managing buffers, and converting SQL in to efficient query paths to leverage these basic storage idioms.

Then, there's the whole multi-users, locking, logging, and transactions complexity on top of the storage idiom.

answered Oct 6 '08 at 1:40

Will Hartung
80.4k 14 91 174

2 I just wanted to say that this is a really interesting and helpful answer. Have you written in more length on this topic anywhere? – Nathan Long Feb 22 '11 at 20:51