Daniel Abadi Speaks Out

by Marianne Winslett and Vanessa Braganholo



Daniel Abadi

http://cs-www.cs.yale.edu/homes/dna/

Welcome to this installment of ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Indianapolis, site of the 2010 SIGMOD and PODS conference. I have here with me Daniel Abadi, who is an assistant professor at Yale University[1]*. Daniel is the recipient of the 2009 ACM SIGMOD Jim Grey Dissertation Award, for his dissertation entitled "Query Execution in Column Oriented Databases". Daniel's PhD is from MIT. So, Daniel, welcome!*

So, Daniel, what is the thesis of your thesis?

So my thesis was, we looked into query execution inside column store database systems. In relational database systems, we have a bunch of two dimensional tables, where rows correspond to entities and relationships, and columns are the attributes. So the majority of database systems that exist, that existed in the '70s and '80s, and up until recently, store data row by row. So when they have to map a two dimensional table to a one dimensional interface of storage, they store the first row, then the second row, then the third row, and so on, all the rows in this table. So what column-stores do is instead of storing it row by row, they do it column by column. So they store the whole first column, and all second column, and so on. The main reason why this is good is that for analytical queries, these queries tend to read a bunch of tuples in the same query. Say you want to aggregate or summarize them, then column-stores are much more I/O efficient.

[1] Daniel Abadi is currently an Associate Professor at Yale.

The reason is that if you store data row by row, since a block size of storage tends to be larger than a tuple, you end up retrieving a bunch of data from storage, more than you actually need to answer the query. Meanwhile, in a column-store, if a query accesses say only three out of a hundred columns in a table, the column store is able to read just those three columns off disk, and therefore get a bunch of I/O improvement that way.

There are some disadvantages. The most disadvantages have to do with writes. So if you want to insert a new tuple into the database, in a row-store you can generally do this with a single write --- you find where you want it to go, and in a single disk write, set the whole tuple on the disk. But in a column-store, if you want to insert a new tuple, you have to break it up into its pieces and write each piece separately. So if you have a hundred attributes, that could be up to a hundred different disk writes (a hundred different disk seeks to execute them). So it is a basic read/write trade off, although it is a little more complicated than that, but that is kind of an overview.

It turns out that the demand is increasing for analytical workloads because people want to just look at the data and analyze it. Once it is already there, they just spend a lot of time trying to get the most of what the data means. Workloads are becoming increasingly read-mostly and that gives column-stores a big advantage.

The great thing about the job market is you get to meet all these new people, and you get to sort of start collaborations, or at least get into the network somehow.

What my thesis did within this context is in two main pieces. The first piece was looking at compression. So it turns out the column-stores also provide a big compression advantage relative to row-stores. The reason is as follows: row-stores store tuples row by row, but a row consists of many attributes. So you have very unsimilar data near each other. However, with column-stores, you store data from the same attribute domain consecutively in the storage. What happens is that you end up getting lower data entropy, and therefore a better compression ratio in column-stores. So immediately after that, we get better compression just from having more similarity in data.

But then there are some additional advantages as well. So one thing that column-stores get you is since they are designed for read-mostly workloads, they tend to very aggressively sort data by the attributes. So row-stores make some guarantees about sort, but they won't generally totally sort the data and store it densely on storage. Whereas column-stores will do that, so in general, you might have the same table stored redundantly multiple times in different sort orders. And sorting also improves compression ratio again because you get more self-similarity in data. Also, when you sort data in a column-store, you get some compression techniques which are possible in a column-store but not possible in a row-store. The most obvious example is run-length encoding. So, with run-length encoding, one data item may appear in several rows consecutively. Let's say you have a table, you are a retail company, and every time a product gets sold you just store the quarter that it was sold, and then the part name, and the customer who bought it. But the quarter that it was sold in (quarter one, quarter two, quarter three), you might have millions of

transactions in that quarter. So if you store your data and order it by quarter, you are going to have multiple repeats in the quarter attribute. So if you want to encode that you might have a million quarter ones in a row that could be encoded to just three integers essentially. So that will obviously get you very good compression. In column-stores it is very easy to do this because in a column-store you store all the data from the same attribute in a row. In a row-store it is not so easy to do this because you have multiple tuples from the same quarter in a row, but you also have these other attributes as well, the product ID, and the customer ID, and the store ID, and so on that are integrated into the data. So it is much harder to do this kind of run-length encoding in a row-store.

So that was the first piece. But then it gets even more interesting. By the way, the reason why we compressed it in the first place is not to save space. Space is nice, but it is generally not a big cost factor. The reason why we compressed data was for performance. Lots of these workloads are I/O limited. If you have less data that needs to be read off disk because it is compressed, then you are able to save time by skipping all that in the query. So that's great. You do have some disadvantages which is that you have to decompress the data eventually. So you have some big I/O advantages, but then before you can actually process the data, in general most systems will go and decompress the data before running the database operators.

So another big part of what my thesis did was we looked at how to operate directly over compressed data. So we looked at compression algorithms which are very amenable to do direct operation, and then we looked at what the architecture of the query executor should look like to be able to extend a database with multiple compression algorithms and not have to totally rewrite the query executor to handle direct operations over compressed data. So, that was one major piece of my thesis.

The other major piece was the tradeoff between early materialization and late materialization of data. So, in column-stores, the data is stored in columns, column by column, but in general, you want this to be just a storage-layer optimization. So at the interface of the database system, you still want people to be able to execute queries using SQL. You want to give users rows back over the connection with the database. So at some point in the query plan data has to be converted from columns to rows. One thing we did in this thesis was we looked at when is the right time to do this conversion, and in general we found that the later the better. There are a variety of reasons for that. The first was this direct operation over compressed data. If data is compressed column by column, and you can operate directly on compressed data, then you can keep data in columns as you go through the query plan. But once you converted columns to rows, since we compressed each column separately, you end up having to decompress data before stitching the tuples together in the same row. So you totally lose the direct operation over compressed data advantage, which is a big negative.

There are other reasons as well. The other main reason is that most queries tend to restrict the tuples over time, so they are either applying a predicate, or they are aggregating data, so the number of tuples at the output of the query are much less than the number of tuples at the input of the query. So especially right at the bottom of the query plan, it's pointless to spend all this time stitching all these columns together into rows if you are going to go and drop the tuple on the floor immediately. In that case, you want to at least wait until after the selection operator, before constructing these tuples. So this is kind of the same reason why we push selects past joins inside the row-store database systems. In general, we have found that late materialization

was good. However, especially when it comes to joins, the tradeoff gets a little bit more subtle, more interesting. For some joins, you do want to still keep data late-materialized, but for some joins you want to actually materialize before the joins, at least the inner-table in nested loop joins or hash joins. I think that is the two main pieces of my thesis that I can talk about.

Do you have any advice for young people who are on the job market?

Yes, I think I have a few experiences that might be useful. So, I interviewed, I guess now a few years ago, on the job market. Actually that was a year, almost a year before I actually graduated, so, a few things here. First of all, I didn't interview at any research labs. I think that was a mistake. So I only interviewed in academic institutions because I thought that is what I wanted, and that is what I wanted, I don't mean to say I am not happy in academia, but by not interviewing in research labs, I didn't get exposed to, I didn't get any connections to research labs. The great thing about the job market is you get to meet all these new people, and you get to sort of start collaborations, or at least get into the network somehow. So I totally missed out on that. And also, looking back right now, I think a lot of the best of systems research is being done at research labs. The Googles, the Yahoos, the Facebooks, you know, they have the best datasets, they have great access to engineers, so they kind of see through the systems through the end. That is one thing in academia, is you can start with a prototype, and work with the students to get a sort of bare bones part of the system in place, but to finish it off and make it useful is really hard to do in academia. And I think that one thing you have in industry is more access to these developers and to real data and to real problems which I think, should not be overlooked in the decision. And looking back on it now I really encourage people that graduate now to look at, even if you think you want to go into academia, to at least interview at research labs, and at least get exposed to that part of the world. That's one thing.

[…] analytical queries tend to read a bunch of tuples in the same query. Say you want to aggregate or summarize them, then column-stores are much more I/O efficient.

The other thing was that I didn't take any time off between graduating and starting my job. In fact, for two months I was trying to finish my thesis and start my new job at the same time. And that really was fairly unpleasant. So I encourage people to, once you have a job in place, most places will say "you don't have to join us immediately", "you can wait six months". Some places will let you wait a year, and then maybe, if you go to academia, then you may spend that year at a research lab, or start a company, or get some exposure at another part of the world, or take time off, which is also probably a good idea, because once you start, it gets very crazy. So that is something to think about as well.

Great advice. If you could change one thing about yourself as a computer science graduate student, what would it be?

I think I would do an internship. So, I never did an internship as a graduate student. I went directly from starting as a student all the way through to the end. I did two weeks at HP labs as a research in residence program, but that obviously wasn't very long, so I think doing internships is a really good way to make some more connections in industry, which I never did.

Great, well, thank you very much for talking with me today!

Thank you very much!