Lecture Materials

Overview of Databases 15 min
Basic SQL Operations 22 min
Data Types in SQL 9 min
Database Keys and Indexes 13 min

Bonus Materials

Assignment

Database Keys and Indexes

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:08
So we just got done talking about whether it's an integer or a character, how long it is, whether it's like a gif image or whatever. Now, we're going to talk, additionally, not just about what's in the columns but how we're going to use those columns. And so, this whole thing of creating tables and defining columns has to do with how we're going to use these things. And now we're going to give it some more detail. So, here we're going to drop this table Users and we're going to create a brand new table and I'll focus first on this User ID column. And let me just sort of foreshadow upcoming lecture where we start having two tables and we got to connect one row in this table to another row on this table. So, this table has like row one, two, three, four. And then this other table has a row that we want to point at row four. So we do this by just putting four in there. And so these are called keys and these are called primary foreign keys. We'll get to all that. And so, the mechanism is we find that we want to enter rows into tables and have just some number. We don't care what the number is, as a matter of fact, we can let someone else come up with a number. And that's exactly what auto increment does. Auto increment is a way to tell the database that we're going to make an integer column and so if we look at this User ID column, it's an int which is short for integer. Unsigned meaning it's positive only. Not null which means it's required and auto increment means please supply it if I don't. And the way it supplies it is it starts with like one, two, three, four. You can insert a row and the database knows all about this stuff and so it just like oh four, next one's five. I know how to do that and so we don't have to in our programs keep track of this stuff. So this is a way to say that. And then we have the two fields, name and email, Varchar(128). So which we did exactly like we did before. And now what we're also going to tell it something about. Remember, I told you about these indexes that were like shortcuts to various places on desk. We're telling, with these next two statements, something about the data and how we're going to use them. So what we're saying with primary key for User ID? We're saying this column is something we're going to use a primary key which means we're going to use it a lot and you better be able to look it up really fast and you better throw some extra stuff on the desk. Extra bread crumbs here and there, so that we can get to that super fast. Okay? So, User ID is a number and I want a very good index and I want it really fast lookups. So the other one, this index, what we're saying there is that we're going to actually look up with where clauses using this a lot. Like where e-mail equals c7emashdot80u or even like clause and we might actually sort this a whole bunch and so we're telling it didn't change what we're going to store here, it's changing what we're going to expect or how we're going to use it. And then these are like hints. All of these are like hints, right? There are also rules. Primary key means it's got to be unique et cetera, et cetera, et cetera. But there also hints to say hey database while we're storing this data, later, I'm going to look at this e-mail address up a lot. I'm not going to look it up by name. I'm really going to look up in this table by e-mail address, so if you're going to make an index, go ahead and make the index on e-mail. I don't know if you're going to make it fast or slow but I'm just giving you a hint, right now, that we're going to use the where clauses with that e-mail and look strings up in there. So please be efficient. But you didn't tell it explicitly what you wanted to do. You just said later I'm like be doing stuff with this. So let's go ahead and create that table. Let me just copy this. Create table statement and go over here and let's see what's in here. Okay, we don't have a table. So I can make the table. If you did use drop table or get rid of it with a little couple of clicks. So, it's an auto increment field, this is another column and this is going to be our primary key for this row. And then we're going to add an index, a character based index, on the name field. And so ultimately, if you look at the structure of this table that I just created, it works pretty much the same as all those other ones. We still have name and email but you see the little key that's kind of telling us that's a good thing to be looked up on and primary key is that's like it knows this and it knows that it's auto increment et cetera, et cetera, et cetera. So if I for example go back in now and I insert a bunch of records. Let's go back to those inserts statements because you'll notice that I don't have to put User ID in. I just add a column called User ID and I don't have it here or here. So watch what happens. Come back. I'm going to insert four records and never specify the User ID. That's what auto increment does. As if I don't specify it, provide it for me. Oh and by the way, it's got to be unique. So I just did that. Let's take a look. Look at this. It just put them in. I can later, when we do this in PHP, I'll be able to pull this number back. I don't say what number did you give me? But it actually assigned all those things for me automatically. That's what auto increment is. Really powerful and when we get to the second major part of this. When we start talking about joints and multi-table stuff. This is going to be essential. But auto increment for now is just a way to make a column that's automatically set up. There's a lot of built-in functions like we talked about now, so you don't have to really know in PHP what the data is. You just say now and then say oh created at is a date time field and insert now into there and it works all the time. There's cool string functions etc etc. I don't teach too many of these other than now because you can go find and stack overflow. You like what's the PHP, what is the mySQL function to take the first three characters of something and you like to cut and paste. You're done. So like I mentioned, indexes, if you don't tell it to index something and you select something, it may have to scan the entire table which is like tun tun tun tun tun and it's like oh that's going to take a while. If there's hundreds of thousands of records, these things can slow down. But the indexes, like I said, a trick based on, you know, it's basically a table of contents, right? So I'll go back and draw that picture again. Right. So the data is stored here and here and here and here and you've got to move in and out and wait for the thing to come around or you store a little bit table called the Index which is a shortcut to each of these records. And this is a smaller amount of data and reads this and says oh four, I'll read the index and I'll go straight to four. Rather than going one two three four. So it's like a set of shortcuts and that's grossly oversimplifying. These are amazing techniques and people have done Ph.D. thesis on how to do this better. This picture is the simplest of them. And so there are a whole Ph.D. thesis's about how to do this on a disk drive, how to do it on multiple disk drives, how to do it on SSD drives. That's the beauty of database work is that thousands and thousands of Ph.D. thesis's in Computer Science have been written to make this software fast and you just say select. And it all works. There's a couple of different kinds of indexes. There's a Hash or Tree. The Hash is used often in primary keys for exact matches and the Trees are used for sorting and prefects matches. But the one I showed you before, I didn't even tell what kind of index. The primary key index was I showed you that and the other one, it's very little space, exact match, no duplicates. But it's super fast for integer fields and you'll notice that that primary key that I put on wasn't an integer field. The index that I used on the other one, either Hash or B-Tree, is good for prefix lookups. The B-Tree is best for prefix lookups. Most people tell you to not even tell it to use Hash or B-Tree and let it adjust sometimes based on the data. This is kind of the internal structure of the B-trees. So remember I told you that you know you have data that sort of scattered all over the place. Or another way to think about it. Let's draw this picture a little bit differently, right? So, here is data and then there's a record here and a record here and record here, it might take you some time to scan through it. So you make this little index that's like a shortcut because you can go randomly. You could go sequentially or you could go randomly. So you have little index and it sort of points to this and these are shortcuts. So you save, you skip all this and skip all this and skip all this. And so B-Trees are one form of index, not the only form of an index. But the idea is there's a small amount of data that points to the larger chunks of data. And so this is a picture of what that might look like. So you can think of this as little block of data that has a series of numbers. The data is basically sorted. And what we say is we have a little, we store a little shortcut to another block of data and anything lower than seven is stored in that block of data. And anything between seven and 16 is stored in this block of data and anything above 16 is stored in this block of data. So if I'm saying oh I would like to look up nine. It comes, it reads this. Okay. And says oh that's, then it reads this and then it reads the nine one, which is out here and so it takes three instead of you know however many it would have had to go the other way. Now when you insert data in something like this. Let's say we're going to insert four. So, we'll put four in here. So we're going insert four. So what happens? This will get a little messy. Is four goes down here. And you look in here and it belongs right there. And it's like oh drat. And so what it does is it sort of moves everything. Now it might split this, so I'll just say let's split it, right? So I'll do this. One two three. One two four and another one that's five six. And then what it will do is it'll say you'll make a five here and then everything below five goes here. Everything five and above, between five and seven goes there. So it split this block into two. Now, you don't need to know this. It sort of has to adjust these things but when it's done adjusting, then you still have a fast way to get all this stuff. And do you really want to know this? And the answer is no, I don't want to know this. Somebody in Computer Science figured this all out. And actually, this is too simple. I mean there's better ways to do this but you don't need to know that. You do need to know that B-trees are good for sorted kind of material and prefix material, especially like a string. If you're looking up names and you're looking up names that started with Ch. You still could find the right spot and then you'd scan for all the Ch's basically. And so that's what B-Trees are good for. Hashes are a little more complex to understand. They use clever math. They use this thing called a Hash function and feel free, It's the thing in Python that makes dictionaries really fast and what it does is it basically does a simple calculation based on the key, like a thing you're looking up, like an e-mail address or a name. And that produces a number. So it runs a little thing, it might look at all the characters and add them up or divide by two or do some combination. But when it's all said and done, it gives you a number say from 0-15 and then based on this calculation, you say oh this one is three and so or actually this one is two and you calculate and you go and you can store it in a slot. And so this didn't take any talking to the disk drive. You just calculate this function. So you go right to the same place. Now the problem with hashing, just like I showed you in the last one, is you can't have more than one key hash to the same place then you do things like have little extra overflows or you split these things, et cetera, et cetera, et cetera. And so, hashing sometimes takes a little more time to get it reorganized. But then it's super fast because you don't even have to read that. You do the hash and you kind of read the disk a very few times. Okay? So Hashes and B-Trees are two different kinds of indexes as we saw in the very beginning. We can indicate what kind of an index we want to put on the various columns. This is a hint. It really doesn't change the syntax that you use but it's a hint. So we've said that this is going to be sort of a Hash-based super fast integer based index. This is more of a string based sorting prefix lookup kind of index. And what's really cool about it is if you forget to put these two things on and you find out later that your application is not running well, you can all come back with an SQL command to add the index later and sometimes there's even tools that will watch your database and say you know what, you should probably put a B-Tree index on e-mail. And like I mentioned, some of my colleagues say you should never say what kind it is and let it figure out the best thing. This part about using B-Tree or using Hash, you might want to say it because if it's strings, probably you want B-Tree. Otherwise you want Hash. But you can also just let the database decide that. What we're really saying is that I'm going to be looking stuff up with where clauses a lot with this email. And so I want you to store that Mr. Database as efficiently as you possibly can. So that's a quick zoom through SQL. And we sort of set up the shape things. We set of rules about the data. We give hints about how we might use that data. Create, read, update and relate. And it's not that hard. It really isn't that hard. And so the next thing we're going to talk about is how we make more than one table and start connecting those tables together and that actually is the real performance gain that we get. Right now, we come up with a very formal way to store data. But it's not really the thing that makes it go super fast, although indexes are cool but joints are the really cool thing and we'll talk about that next.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?