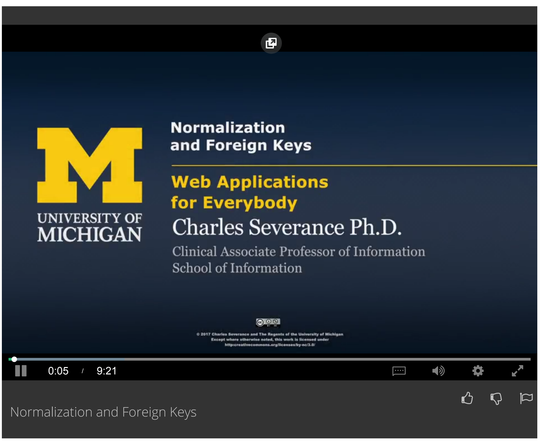
Lecture Materials

Assignment

Bonus Materials



Normalization and Foreign Keys

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

Interactive Transcript

Search Transcript English

0:08
So now we have sort of a picture. We've drank a couple of cups of coffee. We took a break and then we come back we take a look at our picture and we make sure that maybe we got everything right in the picture. And now it's time to get to work and make it so that that picture turns into columns and databases, right? So, we still need to connect all these things. We need to keep track. Like, we can't just say the tracks are in one table, the albums are in another table. We need to connect rows of track table to row the corresponding row of album Days. So we still have to keep track of this stuff. We just don't get to have this vertical duplication of string data. This is what's called database normalization. It's what we've been doing. And so database normalization, you can read as much as you want about this and it has to do with. And I told you about like why relational databases are so cool and there's like all this wonderful predicate calculus that I have no clue about, but I do know this, it's really awesome when someone else figures it out. So databases are super fast. So most of us as programmers, don't need to know the math but the math benefits us. And so, when we're doing this database normalization, we are structuring a database in that the magical underlying cool math that makes databases fast, we can take best use of it. We can write bad databases and then all this magic that it does is not available to us. So database normalization is a study of its own and you could take like a whole semester on the math that makes this work and then there's like, first normal form, second normal form, two and a half normal form or bunch of stuff and that's fun, and if you like that stuff go study it and you'll learn something from it. What I am going to do is, I'm a collapse that down into one slide. I've already told you all the rules. The rules are: Don't replicate vertically data. Don't put the same string in twice. So in a system where you're going to put the name of each of your user and the name is Charles Severance, don't put that anywhere except one place and then get a little number, integer key. We have keys so Charles Severance is equal to two. And so two then is everywhere else you need to mention my name that points to the user table as it were. We're going to use the number two and these integer keys. And so we have this special key column, so that we can like have a handle for each of the rows and you already know how to do that. The auto increment. Remember that, that makes a key column? That's what we're doing with auto increment, okay? And so, what we end up with is a key column in every table. And so this is our artist table, right? Our artist table, we have an artist name and then the key is just our bookkeeping mechanism that we add to every table. And so this is like one of those auto increments and this is just something we're going to do in every row and you've seen this before. What you haven't seen before is what we call a foreign key. And that is, if this is a table called the album table and we have a relationship between albums and artists, we add another column which is the beginning of these arrows, right? The beginning of the arrow called Artist ID that says, this corresponds to row two of the artist table. So go to row two, row two. And so this is it. That's the whole thing. I mean literally, we're going to have put in primary keys and then foreign keys. So we have some terminology. The primary key is the key that indicates the row. And it's generally an integer auto increment and I have a convention. This convention I sort of borrowed a framework called Ruby on Rails. I sort of adjusted a little bit. If I name a table, I tend to name a table and then I have the primary key be the table name underscore ID. So in this class, I make my tables be uppercase, camel case first letter and then I replicate it, but I make all my columns lowercase. Having a convention is good. If you work at a place and they have a different convention, don't tell them about my convention and don't try to tell them that your professor told you that my convention is better. Even though my convention is better, but don't tell them because they think their convention is better and they won't like you very well. So there's a primary key which is the key for the row. Which row it is. A logical key, we talked about this. This is like, your email address or your name or the title of an album. It's how you might look it up. And so, if there was some user interface that had a search, and whatever you'd be typing into that search and hitting the button, that's what we call a logical key. It's a key that the humans outside of our application. The humans should never know or see what this internal key is because it's just a sequence number one, two, three, that allows us to have a handle to grab on to a particular row. Okay? Then, when we have a column that really is pointing to a different table, the artist table in this case, then we call that a foreign key. So it's not a key to the table we're in, it's a key to another table. So a primary key handle for a row, logical key the way humans look the rows up, foreign key points to a row in a different table. Again, integer, integer, and this one is usually not an integer, this is usually a string. So the title, the foreign key is usually a string. Probably the single biggest mistake that people make in any application is they say, "Well, everybody has an email address. Why do I have to make a little number?" See, sev@umich.edu is 916 and then I can't just put these sev everywhere because wouldn't it be easy to look at my database and just see the email address. The answer is, no. You're not going to do that. You are not going to do that. Never use your logical key, the string thing, as the number thing ever, ever, ever. Never. And that's because, logical keys change. Also they take up more space. Now you think oh a numbered number and a string and by whatever. A number takes up four bytes. A string can take up 100 bytes or 200 bytes. Problem is, you don't know how long strings are and so but the numbers are always exactly four bytes. They sort really fast numbers. Computers are so much faster at numbers than strings especially when you start thinking about character sets and sort of Asian character sets and Russian characters and all these other character sets. That's hard for computers to do compared to comparing numbers. So just don't do it. Don't do it. There are some systems I won't mention. Oracle. No I won't mention Oracle. That think, "Oh, don't worry about it just use your string as the primary key. We'll take care of everything." Like, "No, I'm not going to do that." Because in MySQL, you don't want to do that. Maybe there's magic things, I don't use magic. I like to give the primary key. I like to know what it is. Sorry, I got to calm myself down, because I'm imagining some of you were thinking you might use the logic key as the primary key. Logical keys can and do change and even if you're use using Oracle and you think maybe logical keys that don't perform so horribly bad anymore, because Oracle does like useless magic that makes me really annoyed, but you at a university, you can come in and they can assign an email address and you don't like it. Maybe you get married, maybe get divorced. Maybe you just don't like the email address. There are generally at most universities a place that you can say, "Hello, I don't like my email address. I want to be awesomedude@umich.edu." Right? Now, usually they don't just do that, but maybe I did get married and they can change. And then if you have in many applications. If you think of a university, it has hundreds of applications, and perhaps 20 to 100 tables in every one of those applications. And if your email address is in every one of those tables because it's more convenient to look at it, versus a number, it's terrible. So the way it works at this university is, there are like 10 major applications. When you change your email, you kind of hold. You say, "Okay, give us a couple of days." And then what they do is they talk to each application, and if the email address is only one place in the application, bam. Then you kind of wake back up. You get your new email address and then all these applications know about you, cause if you're lucky, different applications will have a copy of your name but each application will have only one copy, so we change 10 applications and literally the entire university sees you as this new email address, okay? Okay. Strings are slow, integers are fast. Logical keys change. I think I've said enough about that. Foreign keys as I've said is a situation where you have a row, an artist. And this is row seven. And you have an album and you want it. This is an album and this can be album four and you want to connect album four to row seven, so you just put a seven in here. So this is the foreign key. That's the foreign key. Now I also name the foreign key with the same as that primary key, so that's like a little memory cues so that I know that. Now Ruby on Rails does this exactly the same way. Okay. Foreign keys are gonna be integers fast, good. Small. Store. Compare. All good. So, now that we understand the basic idea of making these numbers and putting the numbers in various places, let's go through an example of how we would do this.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

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