CS 240: Retrieving Data with SQL Queries Transcript

This video shows a split screen of Professor Wilkerson on the right and a PowerPoint screen on the left. Any text displayed or action performed that is not verbalized will be included in italics as visual descriptions.

[00:00:00] **JEROD WILKERSON:** Now we're going to learn how to use SQL to get data out of your database, or in other words, to ask it questions and get the answer back as a set of data.

[00:00:08] You learned how to do that in a previous lecture when I was showing you the tables.

[00:00:12] You learned how to do it manually by looking at primary and foreign keys and tracing through and seeing what data matches a certain question we might ask.

[00:00:21] We need to be able to do that programmatically as well.

[00:00:24] That's what SQL queries do for us.

[00:00:27] Again, there's a lot that I'm going to teach you about queries, but there's also a lot that I'm not going to teach you.

[00:00:36] We could spend most of the semester learning about all the details of queries.

[00:00:41] We'll defer some of the details to 452, but you will learn a lot about how to get data out, and you'll definitely learn enough to do what you need to for this class and really, for most things that you would do with SQL.

[00:00:55] The general structure of a query is a SELECT statement or a SELECT clause followed by a number of columns, comma-separated list.

[00:01:04] This is the data that we want back in our result.

[00:01:08] This is the set of columns we want to see back.

[00:01:10] Then we have a FROM clause where we specify the tables that contain all or part of that data.

[00:01:16] Then we'll have a WHERE condition again, similar to what you saw with update; we need to be able to specify which rows we're interested in.

[00:01:22] That's what the WHERE clause does.

[00:01:24] Here is an example of a query.

[00:01:28] If I have this table, the book table with all of our books in it, and I say SELECT * FROM book, that says give me all of the columns from the book table.

[00:01:38] I don't have a WHERE clause, so that says give me all the rows.

[00:01:41] That's going to give me a result that is the same as the original book table. I'll have all the data.

[00:01:48] Often, we want just some of the rows.

[00:01:52] The way we do that is with a WHERE clause.

[00:01:55] We also don't always want all the columns.

[00:01:57] If I just want to see the author and the title—I don't want all the data, I want author and title—only for non-fiction books, I could do that by saying "SELECT author, title FROM book WHERE genre = 'NonFiction'".

[00:02:11] I can submit that to the database.

[00:02:14] The database will give me back the answer of which data matches that criteria.

[00:02:20] I can list the subcategories of a particular category from the category table.

[00:02:25] If I want to know what are all of the subcategories of the top category, then I can say “SELECT id, name, parent_id FROM category WHERE parent_id = 1”.

[00:02:37] Let’s look at the table first.

[00:02:39] This is the “Top” category; it’s ID is 1, so its first level children are all of the rows that list the parent ID of 1.

[00:02:49] We have “Must Read” and “Optional”.

[00:02:52] We can see that’s what we’ll get in our result.

[00:02:54] That’s just another example of a query.

[00:02:58] I need to tell you about something that you want to be careful not to do in most cases, and that is to create what we call a Cartesian product.

The following SQL statement creates a Cartesian product:

SELECT member.name, book.title

FROM member, books_read, book

End of code.

[00:03:06] If I do this query, “SELECT member.name”.

[00:03:10] First of all, it’s possible that I have multiple tables in my FROM clause and some of the names of columns are duplicated across those tables.

[00:03:21] If I have that, the way I can say which table’s column I’m referring to is with dot notation.

[00:03:27] I can say member.name.

[00:03:29] That says give me the name column from the member table, and if there were a name column in either of these tables, I wouldn't be referring to that one.

[00:03:36] Here, I'm saying give me the name column from member, the title column from book.

[00:03:43] Here are the tables that contain data I'm interested in or that I need to use in this query, member, books_read, and book.

[00:03:50] Notice there is no WHERE clause.

[00:03:53] What this means, what I've just asked for, is a Cartesian product.

[00:03:58] What that does is it, first of all, takes every row from the member table, and that's going to be included in my result.

[00:04:04] Then it matches every row from the member table with every row from the books_read table.

[00:04:13] To see what that looks like, that means it'll take the first row from member and match it with the first row from books_read.

[00:04:20] Then you'll take the first row from member and match it with the second row from books_read, and so on until it's matched that first row with every books_read row.

[00:04:27] Then it will go to the next row of member and match it with every books row and so forth.

[00:04:31] Just this part of the query, if I have three members and six books, which is what I have in my data, that will give me 3 times 6 rows.

[00:04:40] Then it will take the result of doing that and it will match every one of these rows, the 18 rows, it'll match every one of them to each row in the book table, and I will end up with 72 rows.

[00:04:52] That's probably not what I wanted.

[00:04:54] Probably what I really wanted is to join these tables up the way you did when I was asking you to look at the tables and tell me who has read what book.

[00:05:04] It's probably what we wanted. It's not what we got.

[00:05:06] What we got instead was 72 rows.

[00:05:08] If we look at the result, it looks like we have some duplicate data.

[00:05:12] We can see we have "Ann" and "Decision Points" and then "Ann" and "Decision Points" again.

[00:05:17] The reason it looks like duplicate data is because I'm not showing all the columns.

[00:05:21] Keep in mind that it has matched every row in each table with every row in every other table.

[00:05:26] Some of the columns have the same data, but other columns would have different data if I displayed them all.

[00:05:33] It's not duplicate data, it's just parts of all of these rows.

[00:05:39] Not only is that probably not what I wanted, but you have to keep in mind, in a production system, some of these databases are huge.

[00:05:47] I've worked on databases that have a half a million rows in the table.

[00:05:50] I've even worked in databases that'll have billions of rows, billions with a B.

[00:05:55] If you think about what would be the performance impact of doing a join in Cartesian product on tables that have hundreds of thousands or millions or even billions of rows, that will take so long that you'll completely kill performance of the database with some of these queries.

[00:06:11] You have to be really careful about doing that.

[00:06:13] You might bring the whole system to a halt by doing a Cartesian product.

[00:06:18] Here is what we probably intended.

[00:06:21] We probably wanted to know what books each member had read, which is basically the question I was asking when I first started talking about relational databases. Here's how we would do that.

[00:06:30] Here's one way. There's another way that I'll show you also.

[00:06:33] One way to do that is keep the SELECT and the FROM clauses like they were before, but now I use the WHERE clause to match up the primary and foreign keys.

[00:06:43] Now, you might think about that and think, well, what if I use foreign key constraints? Doesn't that mean the database will match it up for me? It doesn't, because the database doesn't know how you want to use those foreign keys.

[00:06:54] It's not going to make any assumptions about your query.

[00:06:57] It's going to execute exactly what you asked for.

[00:07:00] If you leave off the WHERE clause, even if it has foreign key constraints, it's not going to assume that you meant to use them.

[00:07:07] It's going to assume you meant what you said and what you said was give me a Cartesian product.

[00:07:12] With the WHERE clause, I can say match up.

[00:07:15] If you can remember the table structure, member has an ID column that's the primary key and the foreign key equivalent in books_read is member ID.

[00:07:23] I can match that up by saying "WHERE member.id = books_read.member_id AND book.id = books_read.book_id".

[00:07:31] If you can remember back to when I was asking you questions from just looking at the table, that's exactly what you did visually.

[00:07:38] You would look in the member table and if I said what books has Ann read? You would say, well, what is Ann's ID? Now we'll find that ID in the books_read table.

[00:07:46] Then I will look in the books_read table and find the book_id and find that in the book table.

[00:07:52] That's exactly what we're doing here.

[00:07:54] Now you've learned how to do a join of tables where you're doing in SQL the same thing you already knew how to do manually.

[00:08:03] Now let's look at another version of this query.

[00:08:07] Here I have the same thing I had before, but now I only want nonfiction books.

[00:08:12] I can just add some WHERE clauses.

The following SQL statement is another WHERE clause option :

WHERE member.id = books_read.member_id AND book.id = books_read.book_id AND genre = "NonFiction"

End of code.

[00:08:14] WHERE clauses are separated with either AND or OR—you can use OR as well.

[00:08:19] In this case, I'm saying match up the tables so I know who's read which books, but then only show me the ones that were nonfiction.

[00:08:26] I only want to know who's reading nonfiction books.

[00:08:29] That will give me that answer.

[00:08:32] Now this is great, it works well, and a lot of people do it this way.

[00:08:36] In fact, for most of my career, I've done it this way.

[00:08:38] But one downside to this is we're intermixing the join part of the WHERE clause.

[00:08:49] These columns are just here to join tables.

[00:08:52] What part of that am I looking for? There is syntax that allows us to separate that.

[00:08:58] We can separate it by using INNER JOIN.

[00:09:01] There are different versions of this, and this is the part where I'm going to defer.

[00:09:06] some of the discussion we could go into about this to 452.

[00:09:09] In general, what I can do is I can still keep the SELECT clause the same, but now notice I'm just starting from one of the tables, so I'm just saying FROM member and then join that, INNER JOIN that to "books_read ON member.id = books_read.member_id".

The following SQL statements use INNER JOIN:

SELECT member.name, book.title

FROM member

INNER JOIN books_read ON member.id = books_read.member_id

INNER JOIN book ON books_read.book_id = book.id

WHERE genre = "NonFiction"

End of code.

[00:09:25] I'm saying join member to books_read by matching up these columns: members.id column with books_read.member_id column.

[00:09:34] I'm saying INNER JOIN.

[00:09:36] That's what I get when I do this.

[00:09:38] This is what we call an inner join.

[00:09:40] An inner join joins the tables, but I don't get values for which there's not a matching row.

[00:09:50] For example, let's say that I have my member table and I have some members who haven't read any books.

[00:09:57] When I do the query, they won't be included in the result.

[00:10:02] If they haven't read a book, they won't be included.

[00:10:04] That's an inner join. There are other kinds of joins.

[00:10:08] There are outer joins, where I could say I want every member and if they read a book, I want the title, and if they haven't read a book, I want the title to be null.

[00:10:17] That would be an outer join.

[00:10:19] In fact, that would be what we'd call a left outer join.

[00:10:21] I would write left outer join or I would just write left join and that's the same thing as writing left outer join.

[00:10:27] There are different kinds of joins.

[00:10:29] There are inner joins which are the standard ones, but if I want to include cases where there's not a relationship to the table I'm joining with, I would do that as either a left or a right outer join.

[00:10:41] There's even a full outer join, where from joining two tables, I can get rows where I can have null columns in results from both tables.

[00:10:55] We're starting to get a little deeper than I really want to with this syntax.

[00:10:59] For now, we're just going to keep it simple and say that if you want to separate the join part from the rest of the where part, you can do that with inner join.

[00:11:09] Then when you take 452, you'll learn a lot more that you can do with those types of joins.