SQL Overview

I'm going to introduce some SQL code to you. You do not need to be able to create this code yourself or even understand it completely. The purpose of this overview is just that, to give you a brief overview of SQL statements. We will look much closer at each type of statement and all the details of each in weeks to come.

Let's download the datawinter database. (See the activity that goes with this video to get that sql file downloaded.) Open up Workbench and click File, and Open SQL Script. Find the file you downloaded and click Open. Don't just double click the file from downloads it might open without a server connection or open by default in another editor. So use File and Open SQL Script. Once it's opened, you should see a bunch of code that might look very foreign to you. This is just a file that has all the code and data needed to create and put data into a small database. This is the same kind data that you all collected the first week of class. This is actual data from one of my Winter semester students. I've only changed their names.

Before we run this, which will create and populate the database, let's look at some of the code. Some of the statement fall into the DDL category of statements and some into the DML category.

Here's a line that creates a database called datawinter. This is an example of a DDL statement. Data Definition Language. DDL has to do with the actual structure of the database itself. So, this statement creates an empty database named datawinter. This statement is also DDL and it creates a table of the database called image. So again DDL having to do with the structure of the database—adding a table.

This INSERT statement doesn't affect the structure of the database, it's just getting data into the database. So, it is not DDL but DML which stands for Data Manipulation Language. All the SELECT statements we are going to look at next will be DML because we won't be affecting the structure of our database, just retrieving data out of the database in a result set.

Let's run this and click on the lightening bolt to run everything and if I refresh my database list it should show datawinter there.

Also it's good to know that none of these SELECT statements will change the original data in the database, just retrieve the data we want from the database.

Before we run the SELECT queries make sure you are using the correct database. You can double click the database name from the schema list or run:

```
USE datawinter;

SELECT * FROM student;
```

We get all the data from the student table. The asterisk means every column in that table. SELECT and FROM are keywords and it is best practice to capitalize them even though they don't have to be. Student is an identifier that represents the name of the table that we are getting data from. This query has two clauses each beginning with a keyword, the SELECT clause and the FROM clause.

Let's try another:

```
SELECT lname, fname, major
FROM student;
```

Instead of the asterisk, I'm replacing it with column names, I'm adding more identifiers that represent columns in the table this time. This would get only 3 of the columns from the table. Each column name is separated with a comma.

Notice how I've been ending each statement with a semi-colon. This helps differentiate one statement from another. I have a number of statements on the same SQL script file here. If I wanted to save this, I'd just click File, Save Script then I can name it and it would have a .sql extension.

It's also important to know that you should only run one query at a time. In Workbench if you click the lightening bolt, the exectution icon without the insertion point, it will run every statement on the current tab. I would instead use the icon with the insertion point to run just the statement that has the insertion point in it or close to it. Or use the keyboard shortcut, CTRL (or Command) + Enter. You could also highlight just what you want to run and click the lightening bolt. But it's important to run one query at a time, and not run everything on the page every time.

Let's add a filter to our query.

```
SELECT lname, fname, major
FROM student
WHERE major = 'CIT';
```

Now it only shows those students who have the major of CIT and no other student. The string CIT is placed in quotes because it is a constant, or a value from the database that is a string or character. String values will have quotes around them.

Let's add a sort to our query. Notice how there is no order to the results except how I entered it into the database.

```
SELECT lname, fname, major
FROM student
WHERE major = 'CIT'
```

ORDER BY lname;

This will alphabetize it by lname.

Let's look at which students had internet as their top distraction.

```
SELECT lname, fname
FROM student
WHERE dist_category = 'internet';
```

Now remember from the ERD we saw of this database, the dist_category was an ENUM datatype so the data entry was limited to one of these words so we know we got all the students with that category. But what if we change the dist to worry category and put family. So we are getting all students who worry the most about family.

```
SELECT lname, fname
FROM student
WHERE worry_category = 'family';
```

Look at the ERD again the worry category is a VARCHAR which leaves it open to the data entry person to any string 45 characters or smaller.

Let's look at everything in the table again:

```
SELECT * FROM student;
```

Now I left this data exactly as the students entered it. So look at Brandon Brown he put wife as his top worry, but really that fits in the category family right? So when we run this statement it's not going to get as exact data. Now let's run it.

```
SELECT lname, fname
FROM student
WHERE worry_category = 'family';
```

Notice Brandon doesn't show up even though we worries about this wife, which is his family. So datatypes matter if you want your result sets to be exact.

Now remember the purpose of these SELECT statements is to get the data we need to make our decisions from the database. So, think back to our Statement of Work what kinds of questions might the app developers need from this data to make an effective app for college students? They probably don't need to know names of students or groups but overall data like to answer questions like:

'What is the top distraction category for all the students?"

Or
“What is the top worry for all students?”

That code could be answered in query that looks like this. Notice the comment at the top of the query. The dash, dash space, tells the system to ignore this code when running the query. It’s just a note to myself so I remember what this query is doing. You can also use a hashtag to start comments.

```
-- Which group in section 1 had the most selfies?
SELECT dist_category, COUNT(*)
FROM student
GROUP BY dist_category
ORDER BY COUNT(*) DESC;
```

Notice there are more keywords and clauses on this one that we haven’t seen before. We’ll learn these in a later week. But it’s basically adding up every time it sees a certain distraction and putting the one counted most at the top. Notice that only the categories limited by the ENUM are there.

If we change it to the top worry:

```
SELECT worry_category, COUNT(*)
FROM student
GROUP BY worry_category
ORDER BY COUNT(*) DESC;
```

We get lots more categories because it was VARCHAR, even though I told the students to enter it in by the category on the data gathering sheets, they didn’t and that’s true of more data entry. Notice even the misspelled values create a new category. That’s the risk you take for using the VARCHAR when you know there are specific categories that they can fall into. But it works the same way, showing the biggest worry on the top.

The last few are just some queries that use more than one tables data. For example, if we wanted to know which group in class section 1 had the most selfies, we’d have to go from group to student to image (see the ERD). So this query is joining tables together to get all the data it needs.

```
SELECT group_name, COUNT(type)
FROM student_group JOIN student
  ON student_group.group_id = student.group_id
  JOIN image
  ON student.student_id = image.student_id
WHERE student_group.location LIKE "%1" AND type="selfie"
```

```
GROUP BY group_name
ORDER BY COUNT(type) DESC;
```

OK so there's a quick brief overview of SELECT statements, what we will be spending the majority of our time on here in this class. I wanted to go over them briefly this week because the next few weeks we will be going over database design and won't see a lot of SELECT statements for a few weeks, but they are a huge part of this class.