(W4) - Modifying and Analyzing Data

- Concatenate
- Trimming
- Changing Case
- Substring Functions
- Date and Time String

String Functions:
- Concatenate
- Substring
- Trim
- Upper
- Lower

Concatenate

```
SELECT Company_name,
       Contact_name,
       Company_name || `(` || Cotact_name || `)’
FROM Customers
```

In order to concatenate two strings use pipe (vertical bar key).

Trim

TRIM
RTRIM
LTRIM

```
SELECT TRIM (“    You the Best.      “) AS TrimmedString
```

Substring

Substring is a useful function that allows you to pull apart just a portion of the string that you're looking at.

SUBSTR (string name, string position, number of characters to be returned)

```
SELECT first_name, SUBSTR(first_name, 2, 3)
FROM Employees
WHERE department_id = 60
```

Applying the above function on Alexander, Bruce and Valli would return lex, ruc and all respectively.

Applying the following to Nancy would return ncy. It just gives me whatever it can fill in with it.

```
SELECT first_name, SUBSTR(first_name, 3, 4   )
FROM Employees
WHERE department_id = 60
```

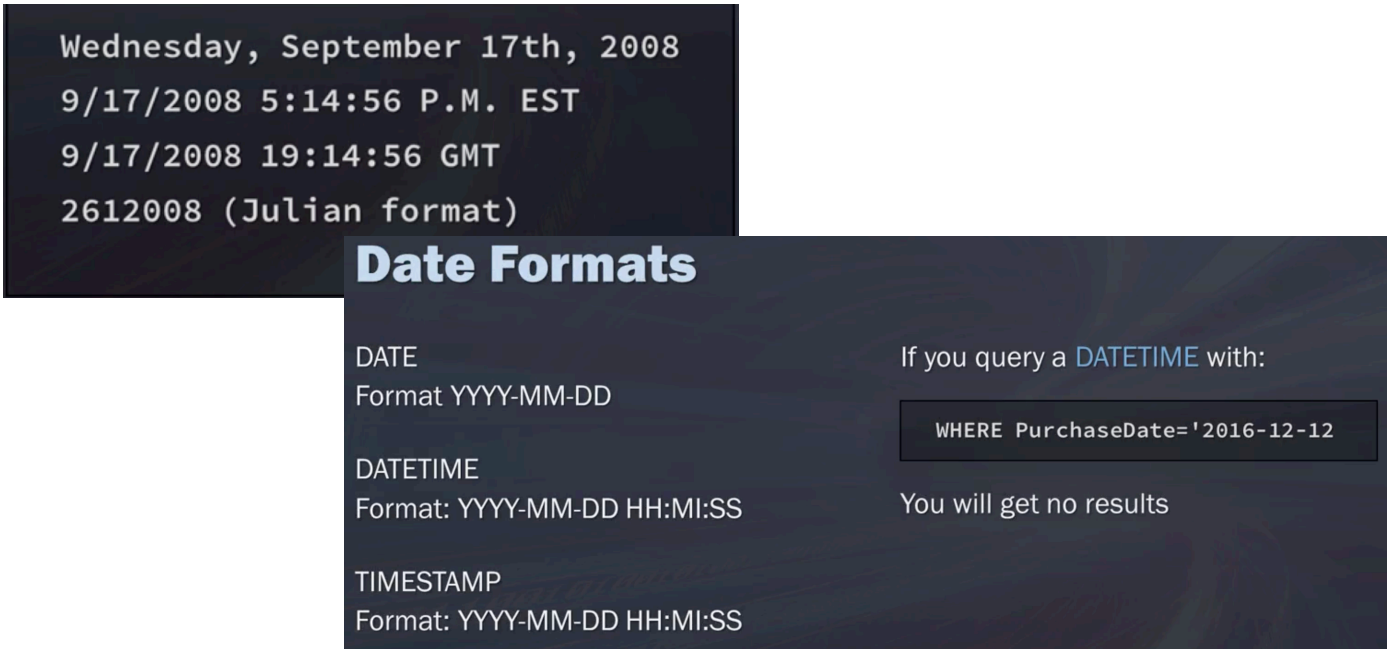
UPPER and LOWER

```
SELECT UPPER (column_name) FROM table_name
SELECT LOWER (column_name) FROM table_name
SELECT UCASE (column_name) FROM table_name
```

UCASE and UPPER are the same.

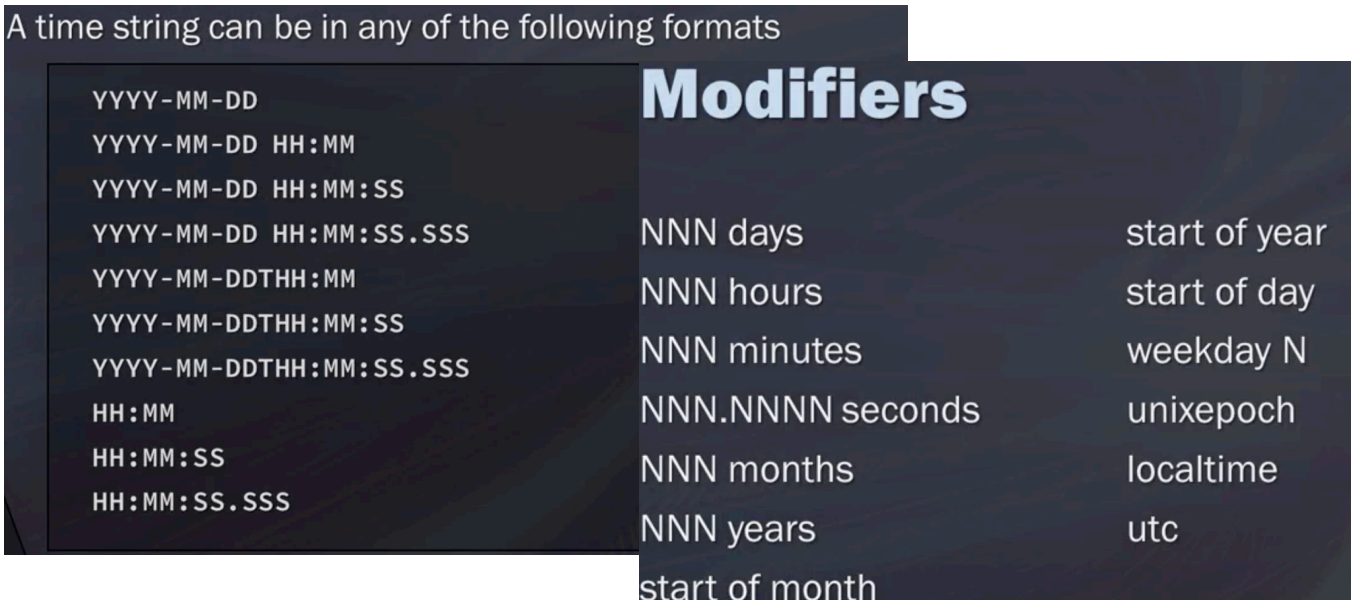
Date and Time

There are two many different formats. Look what database you are working with and what type of format it uses.



SQLite supports 5 different functions to manipulate time and dates:

- DATE(timestring, modifier, modifier, …)
- TIME(timestring, modifier, modifier, …)
- DATETIME(timestring, modifier, modifier, …)
- JULIANDAY(timestring, modifier, modifier, …)
- STRFTIME(format, timestring, modifier, modifier, …)



A time string can be followed by zero modifier to multiple modifiers. Each modifier transforms, that is applied to the time value is applied from left to right. The order is important.

- **STRFTIME**
- **Compute current date and compare it to a recorded date in your data**
- **Use the NOW function**
- **Combine several date and time functions together to manipulate data**

Example:

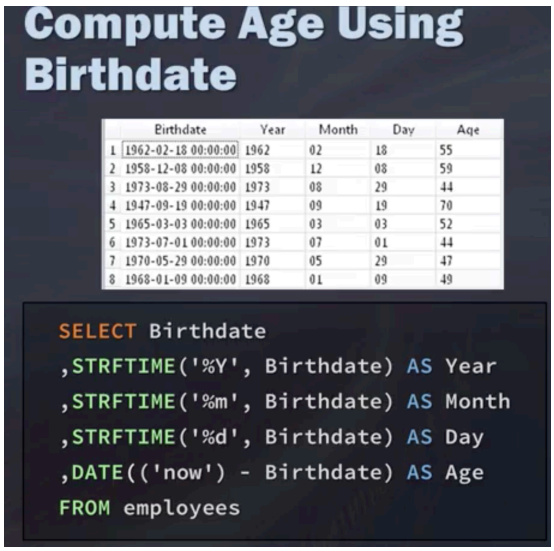
```
SELECT Birthdate,
        STRFTIME ('%Y', Birthdate)  AS Year,
        STRFTIME ('%m', Birthdate) AS Month,
        STRFTIME ('%d', Birthdate)  AS Day,
FROM employee
```

	Birthdate	Year	Month	Day
1	1962-02-18 00:00:00	1962	02	18
2	1958-12-08 00:00:00	1958	12	08
3	1973-08-29 00:00:00	1973	08	29
4	1947-09-19 00:00:00	1947	09	19
5	1965-03-03 00:00:00	1965	03	03
6	1973-07-01 00:00:00	1973	07	01
7	1970-05-29 00:00:00	1970	05	29
8	1968-01-09 00:00:00	1968	01	09

Another function is the function to find current time:
SELECT DATE(‘now’)
SELECT STRFTIME(‘%Y %m %d’ , ’now’) <— ‘now’ is a modifier
SELECT STRFTIME(‘%H %M %S %s’ , ’now’)

Example:

```
SELECT Birthdate,
        STRFTIME (‘%Y’, Birthdate)  AS Year,
        STRFTIME (‘%m’, Birthdate) AS Month,
        STRFTIME (‘%d’, Birthdate)  AS Day,
        DATE(‘now’) - Birthdate      AS age
FROM employee
```



CASE STATEMENT

```
CASE
WHEN C1 THEN E1
WHEN C2 THEN E2
…
ELSE [RESULT else]
END
```

```
CASE input_expression
        WHEN when_expression THEN result_expression […N]
        [ELSE else_result_expression]
END
```

Example:

```
SELECT employee_id,
        firstName,
        last_name,
        city,
        CASE city
            WHEN 'Calgary' THEN 'Calgary'
        ELSE 'Other'
         END AS Calgary
FROM Employee
ORDER BY LastName, FirstName
```

CASE can be used as a search:

```
CASE WHEN Boolean_expression
THEN result_expression [… n]
[ELSE else_result_expression]
END
```

For this example, I'm going to show you how you could add a couple of the cases together. Here, what I'm looking at in the Chinook database is how I'm going to classify my tracks. I want to classify them based on the number of bytes they have. Again, we have discussed a little bit earlier in the course how we do this when we're doing predictive modeling or forecasting. So here, we may want to bin all of our small sales customers into one and predict their future sales, or large scale customers into another, and so on and so forth. In this case, I'm going to be looking at the size of the tracks, and so I want to bin the bytes.

```
SELECT trackId,
        name,
        bytes,
        CASE
          WHEN bytes <30000 THEN 'SMALL'
          WHEN bytes >=30001 AND bytes<=50000 THEN 'Medium'
          WHEN bytes >= 50001 THEN 'Large'
          ELSE 'other'
        END (AS) bytes_category
FROM Tracks
```

VIEWS

we're always combining data from multiple sources or transforming it in some way. As you know, sometimes things like the order of operations can get a little tricky. Instead of creating a whole new table, sometimes we can create the illusion of a table by using a view. A view is essentially a stored query, and it helps us clean up our queries and simplify when we have to write. In a view, you can add or remove columns without changing the schema. You're not actually writing the query to the database or anything, what you're doing is you're kind of storing it for the time being. This is really helpful and pays off when we use it to encapsulate queries.

```
CREATE [TEMP] VIEW [IF NOT EXIST]
view_name (column_name_list)
AS
select_statement
```

Example:

Let's say I want to get a count of how many territories each employee has. If you look at our Diagram, this information is separated out from each other. I'm going to create a view, so that on that view, I can just run a simple count on the number of territories. So here I will create my view. Then for my view, I'm just going to call it as my_view.

```
CREATE  VIEW my_view
AS
SELECT r.region_description,
        t.territory_description,
        e.Last_Name,
        e.First_name,
        e.hire_date,
        e.Reports_to,
FROM Region r
INNER JOIN Territories t ON r.region_id = t.region_id
INNER JOIN Employee_territories et ON t.territory_id = et.Territory_id
INNER JOIN Employees e ON et.employee_id = e.Employee_id
```

To actually see the data, we need to write a select statement:

```
SELECT *
FROM my_view
```

	regiondescription	territorydescription	Lastname	Firstname	Hiredate	Reportsto
1	Eastern	Wilton	Davolio	Nancy	5/1/1992 12:00:00 AM	2
2	Eastern	Neward	Davolio	Nancy	5/1/1992 12:00:00 AM	2
3	Eastern	Westboro	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
4	Eastern	Bedford	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
5	Eastern	Georgetow	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
6	Eastern	Boston	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
7	Eastern	Cambridge	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
8	Eastern	Braintree	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
9	Eastern	Louisville	Fuller	Andrew	8/14/1992 12:00:00 AM	NULL
10	Southern	Atlanta	Leverling	Janet	4/1/1992 12:00:00 AM	2
11	Southern	Savannah	Leverling	Janet	4/1/1992 12:00:00 AM	2
12	Southern	Orlando	Leverling	Janet	4/1/1992 12:00:00 AM	2
13	Southern	Tampa	Leverling	Janet	4/1/1992 12:00:00 AM	2
14	Eastern	Rockville	Peacock	Margaret	5/3/1993 12:00:00 AM	2
15	Eastern	Greensboro	Peacock	Margaret	5/3/1993 12:00:00 AM	2
16	Eastern	Cary	Peacock	Margaret	5/3/1993 12:00:00 AM	2
17	Eastern	Providence	Buchanan	Steven	10/17/1993 12:00:00 AM	2
18	Eastern	Morristown	Buchanan	Steven	10/17/1993 12:00:00 AM	2
19	Eastern	Edison	Buchanan	Steven	10/17/1993 12:00:00 AM	2
20	Eastern	New York	Buchanan	Steven	10/17/1993 12:00:00 AM	2
21	Eastern	New York	Buchanan	Steven	10/17/1993 12:00:00 AM	2
22	Eastern	Mellvile	Buchanan	Steven	10/17/1993 12:00:00 AM	2
23	Eastern	Fairport	Buchanan	Steven	10/17/1993 12:00:00 AM	2
24	Western	Phoenix	Suyama	Michael	10/17/1993 12:00:00 AM	5
25	Western	Scottsdale	Suyama	Michael	10/17/1993 12:00:00 AM	5

To get rid of the view:
DROP VIEW my_view

Now that I have my_view out there, I can actually perform even more queries on top of that. I can now take that view and I can select the counts in the territory descriptions. For example, this will give me an idea of the counts of how many territories that each employee has. I can then group it by the employee's last name and first name. Now I can see the total count for each territory of what each employee has. This would've been a little bit more complex to do if I tried to do it all at once. But creating view just made things really simple. The beauty of the view is that it can be used like a table. But it's unlike a table in that you don't have to have ETL or run ETL on any of the data. This helps a lot by encapsulating complex queries or complex calculations that you're trying to write. It can really help simplify it. It can also be used in pretty much any database, except for stored procedures.

Now that I have my_view out there, I can actually perform even more queries on top of that. I can now take that view and I can select the counts in the territory descriptions. For example, this will give me an idea of the counts of how many territories that each employee has. I can then group it by the employee's last name and first name. Now I can see the total count for each territory of what each employee has. This would've been a little bit more complex to do if I tried to do it all at once.

This helps a lot by encapsulating complex queries or complex calculations that you're trying to write. It can really help simplify it. It can also be used in pretty much any database, except for stored procedures.

Views are really most helpful if you need to join a set of tables and you're having trouble getting calculations. Particularly those complex ones dealing with the order of operations in the right order to get the output you're looking for. Another benefit of views includes different securities or write capabilities. We talked about not being able to write data to an environment or to a particular database. Views are helpful because you're creating a view of a table but not actually writing data to that table. This is a way to get around some of those database writing limitations.

Another thing that views are helpful for is to create a stepping stone in multilevel queries. For example, let's say you create a query that counts the number of cells that each person has made. You could then write a query that groups the salespeople into a particular group. Then you can count the sales of that group as well. It just creates this multilevel dimension that you

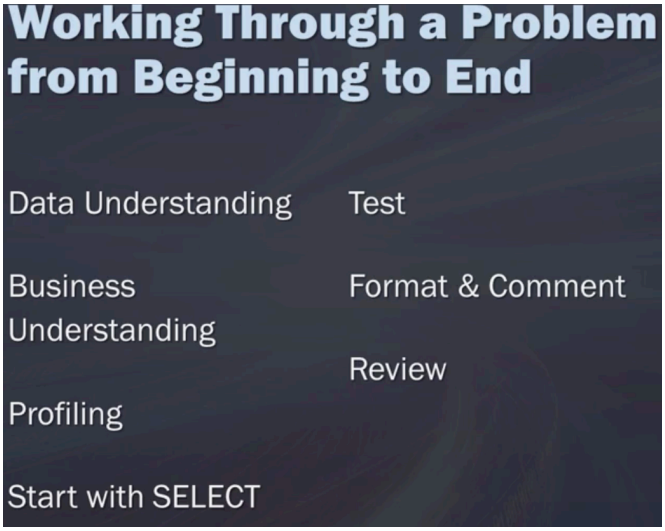
wouldn't have been able to do elsewhere. And then, it also helps so that you're not transferring any data through and ETL process.

Data Governance and Profiling

Profiling your data is where you're looking at either descriptive statistics or different information on the data. And the reason why this is important is because we've talked about how you need to understand your data first, before you start querying it. So profiling is a great step to begin to understand your data. It's really simple things that you can do to start to profile your data.
Some of the things you can do is start with just understanding how many rows are in the table. You can also look at when was the object last updated? Meaning when did the data get refreshed and reloaded? Because this may change some of your results. This may change the data that you need to limit it by.
If it's getting refreshed nightly, or in real time, do you need to set a certain date parameter or are you okay with a consistent flow of data?
So understanding these things is essential for understanding how you might want to pare down your data to get what you're looking for.
You can also do some column data profiling. So for this example, just start off with looking at what is the actual column data type? Is it a date? Is it a timestamp? Is it a date stamp? Is it a string or an integer?

Using SQL for DS

These are just a few things that I use when starting a new problem. I think it's really important to know how to work through a problem from beginning to end. And using SQL for data science, what we're often doing, is extracting the data from some storage system, analyzing it, and then maybe writing back a



prediction to the database. This is usually centered around a question or a type of analysis that we're doing. A problem that we want to solve. I think there's a few principles that you can use in making sure that the SQL piece of your work is going through a problem from beginning to end and is successful.
All of this starts with the data understanding. This is the most important step. This is why we spent so much time understanding and explaining modeling and Diagrams, and discussing the relationships in your data. Because understanding your data is key to being able to write successful queries. What I mean by understanding your data is really kind of a combination of data understanding and business understanding.

It's definitely data understanding, and asking yourself things like, are there lots of NULLs value in this? Is the data made up of string values that were just free form or entered?

Or is it concatenated dates and times? But then, there's also this concept of business understanding, meaning how do all these pieces and elements relate to each other. If you're new to this subject area and you've never worked with the data before, it's going to take you a little bit longer to write your queries. Because of this, it's going to take you a little bit longer to figure out, how does everything work together? How does it join or relate to each other?

But it'll always be worth taking the time to understand your data as much as you can before you really start to analyze it. It's important to really understand the relationships and the dependencies. That leads us to our second step, which is the business, or subject area understanding. As you start to get familiar with your data, what will happen, is that you'll run into questions about the business problem you're trying to solve, the problem or subject or area that you're looking at.

I don't know if steps one or two, data understanding and business understanding, are really separate. Usually, I'm going back and forth between looking at the data and going back to a subject matter expert or somebody who really understands the core business problem, and trying to solve the problem. And I'm asking them more and more questions. Going back to the data then going back and asking more questions. This is essential in being able to really wrap your head around the problem. But

also, then, wrapping your head around the data you're using to try and solve for it.

One of the things to be careful for is what I call, the unspoken need. You may have a business problem where they say, for example, we want to predict whether or not a customer is likely to buy our product. That seems pretty straight forward and easy, right? But as you dive into the data more, you may start to get questions like, well, what customers? What products? Some of the things that are unspoken are certain logical exclusions. For example, are there certain customers that should be excluded from the this analysis? Are there certain cases where past cells shouldn't be added into this model, or should it be counted? This is why you have to walk in between that data understanding and business understanding. Because you frequently need to look at the data to get questions. And then, you need to go back to the business to understand the problem better.

We're going to talk about those steps in this video. After this lesson, you should be able to determine and map out the data elements needed for a query. Discuss some of the strategies to employ, as you'd begin to write more complex queries. And explain some common troubleshooting techniques to try in your SQL code when it isn't giving you the results you expect.

Okay, so to really understand a problem, you really need to map out what are the exact data elements you need.

You need to know the data you're going to go after and understand some of the issues with the data from the profiling you've done. So where do you start with your data and query? If you're always extracting data, it's always going to start with the select statement. So you're going to have to use select and from.

What I do is usually write out okay, where is the data that I need? And then kind of draw out a diagram of the different tables and the pieces of information I need on paper. Basically, just creating my own data model and map.

I start with this just as sources, and then from each source, I go down and define the fields I need. And then from there, I also define how I'm going to join those different sources together.

From that point, I'm going to decide if I need to do any calculations. It's just kind of going through a logical process that I go through. But again, you're always going to start with SELECT. I mean that's the great thing about SQL, it's consistent in that way.

What I recommend is to start simple, especially if you're new to the data. Start with just one table, add in more data, add in another table, check your results and then go back from there. If you're using sub queries, remember to always start with the innermost query and then work out and build. Start small.

That leads us to our next tip which is test along the way. Don't wait to test your query until you've combined multiple sources together. And you have all your calculations done and finished. Think of this as little building blocks. If you write a calculation of the average selling price of something, look at how many values you're getting back for just that calculation from the table and make sure that seems right. Then combine this result with another table and then test that. If you know your data, you could dive in a little bit quicker. But this will really make sure that your order of operations is correct. This is key because as I have said before, it's easy to get results back but getting the right results back that you expect is a little bit harder.

Okay, let's talk about troubleshooting. When troubleshooting, it's important to always start small and simple, and slowly start to rebuild the query to see where things went awry.

It's helpful to start at the basic things first. Okay, I'm getting these fields from this table. Does that work? Yes, okay, now I'm getting these fields from this table and from another table. What's my join like? Is this working? Okay, yes it is. Slowly start to build it back up in order to figure out where things went wrong. Let's say at this point, you're working through a problem and you know your data you have profiled it, you've tested it, and you have started simple and you have your query.

Be sure that when you are writing it, the next thing to look at is to make sure you are formatting it correctly and commenting nicely.

I think that clean code says a lot about you. Make sure that it's easy to read, you're using popular indentation, you're commenting strategically where you need to, etc. You never know when you're going to need to revisit your query or you're going to need to hand it in to someone else and they need to edit to, from there. Just keep your code clean. Format your comments where necessary and strategically.

Then, you want to make sure to review what you've done. A lot of times, what happens is you'll write a query, you'll be using it for your model, and you'll be looking at different stats and things like that. Then you need to go back and edit and change that query. Always make sure you review the query to see if anything has changed. Has the data changed? Are the business rules different? Do you need to update and change the date indicators? Does anything need to be updated? And general, be really careful when you're going back and using old queries.

Okay, that really takes you through a problem from beginning to end. Again, it all starts with the data and problem understanding. Make sure you spend the time there. Make sure you are really spending the time thinking about what you are doing before you actually start writing the queries. I promise, it will save you time in the long run, then go through and really understand your data through profiling it. Make sure you're testing along the way, and keeping your code clean and commenting.

Those are just a few little tips that I can give you. You're fully equipped now to go and retrieve the data you need, which is exciting because the first step in doing data science is to be able to get your data.

You now have that in your toolbox. Reflect on these steps and framework when you look at problems and start writing your queries. All right, go and get your data and start analyzing in.