Module 2: Information Science for Chemists (Stuart Chalk)

Learning Objectives

By the end of this module students will:

- Understand how computers represent letters, numbers, and symbols
- Be able to identify different information types
- Appreciate the difference between binary and text file types
- Be able to identify different computer languages used on computers to develop applications and construct webpages
- Understand what a relational database is and the difference between an SQL and noSQL database
- Appreciate data websites and the concepts behind how an application programming interface (API) can be developed to access such sites

Introduction

Chemical Informatics (CI), the application of information science to chemical data, is an interdisciplinary field of growing importance. The reason; there are lots of chemists, lots of chemicals, and chemists have lots of data about the chemicals. The result; it is increasingly more difficult to find the information you need, or even find out if the information you need exists, because the amount and global distribution of data is so vast. Scientists in the area of CI are therefore trying to find ways to identify, organize, characterize and integrate data from many different sources, stored in multiple ways using open and easily implemented techniques and technologies.

This module is focused on teaching you, a budding chemist, the basics behind information, how it's stored, identified and manipulated – so that you can understand how to use it effectively and efficiently.

Basics of Computer Systems

For a course about chemical information we start at the lowest level – how computers deal with information internally. As a start, imagine you have a Word document that contains the following:

904-620-1938

Humans of course can very easily identify that this is a telephone number, but to a computer it looks like the following in its basic format – binary (more information on this later…)

```
00111001, 00110000, 00110100, 00101101,
00110110, 00110010, 00110000, 00101101,
 00110001, 00110001, 00110011, 00111000
```

Whether it be on a regular hard disk drive (magnetic coating on a disk) or a flash/SSD (memory chips) all data on a computer is stored as 1's and 0's (see http://www.pcmag.com/article2/0,2817,2404258,00.asp). As a result, all data on a computer must be represented in binary notation. Each recorded 1 or 0 is a 'bit' and eight bits make a 'byte'. A byte is the basis of presenting data because eight bits, being either 1 or 0, can together represent numbers up to 255. This is because for each bit there are two possible permutations and eight bits gives 2^8 combinations - 256 values, or 0 thru 255.

In the early days of computer systems, a single byte was used as the way to represent text characters, and was defined by the American Standard Code for Information Interchange (ASCII – see http://www.ascii-code.com/). Initially, only the numbers 0-127 where used to represent letters, numbers, punctuation marks and symbols (32-127), and non-printed characters (0-31). Subsequently, extended ASCII was introduced which added accented characters, other punctuation marks, and symbols (128-255).

Looking back on the example of the telephone number above, we can now translate the binary into the telephone number

Binary	Decimal	ASCII Character
00110000	48	'0'
00110001	49	'1'
00110010	50	'2'
00110011	51	'3'
00110100	52	'4'
00110101	53	'5'
00110110	54	'6'
00110111	55	'7'
00111000	56	'8'
00111001	57	'9'
00101101	45	'-'

Although we still 'use' ASCII today, in reality we use something called UTF-8. This is easier to say than how it is derived - ***U***niversal Coded Character Set + ***T***ransformation ***F***ormat - ***8***-bit. Unicode (see http://unicode.org) started in 1987 as an effort to create a universal character set that would encompass characters from all languages and defined 16-bits, two bytes -> 2^{16} -> 256 x 256 = 65536 possible characters – or code points. Today, the first 65536 characters are considered the "Basic Multilingual Plane", and in addition there are sixteen other planes for representing characters giving a total of 1,114,112 code points. Thankfully, we don't need to worry because if something is UTF-8 encoded it is backward compatible with the first 128 ASCII characters.

It's worth pointing out at this stage that the development of Unicode is a good thing for science. We speak our own language and have special symbols that we use in many different situations (how about the equilibrium symbol? $\rightleftharpoons$) and so publishers in science and technology have developed fonts for reporting scientific research. Check out and install STIX fonts (http://www.stixfonts.org) which would not be possible without Unicode.

Information Data Types

Looking back at the telephone number in the previous section our identification of the characters is based off of two things – the fact that there are only digits and hyphens, and the pattern of digits and hyphens. Again though the computer knows nothing of this, it just knows that there is a string of 12 characters. So, in the context of humans representing information in computers it is not just important for us to use characters, we also need to know generically what they represent. This is critical to everything we do on a computer because we need to know how to process the characters.

In a broad sense characters represent either; text, numbers, or special formats. In the context of discussing the common 'data types' we are going to reference those that are used in the relational database software 'MySQL'. More about what MySQL actually is later, for now we will look at the ways in which the program represents information that it stores (see http://dev.mysql.com/doc/refman/5.6/en/data-types.html for more details).

Numeric types

Representation of different numeric values is important because it is dependent on how they are used. Choosing the right type is critical to making sure there are no inconsistencies, especially when making comparisons or rounding in calculations.

- *Integers* – You would think that there is only one type of integer (i.e. an exact number), but it turns out that we have to think about how much space we use up storing the number on a computer not just its type. MySQL defines TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT. The definitions of these are based on how big the number is they can represent which in turn is based off how many bytes are used to store the number. For each of these (like all numeric values) you also have to specify if the first bit of the leading byte indicates ± or not.
- *Fixed Point Numbers* – DECIMAL (or NUMERIC) in MySQL are used to store values with a defined precision (a set number of decimal places) and thus are non-integer exact numbers. Using the syntax DECIMAL(X,Y) you can define a decimal with X total digits (max 65) and Y decimal digits (max 30), so DECIMAL(4,2) allows you to store from -99.99 through +99.99.
- *Floating Point Numbers* – Very important in computers, and chemistry, floating point numbers FLOAT and DOUBLE are used to represent non-exact decimal numbers that range from very small to very large. The difference between FLOAT and DOUBLE is FLOAT uses four bytes to store the number (accurate to seven decimal places) and DOUBLE uses eight bytes (accurate to 14 decimal places). FLOAT is normally good enough for scientific calculations as it can store -3.402823466E+38 to -1.175494351E-38 and 1.175494351E-38 to 3.402823466E+38. These definitions work with the common IEEE standard of floating point arithmetic (see https://dx.doi.org/10.1109%2FIEEESTD.2008.4610935)

String Types

The differences between string types is primarily based on how much space they take up and how easy they are to search. Thinking about the application of the string is important especially deciding if a string should be exclusive or inclusive. Also, realize that there are situations where you might want to store a number as a string.

- *Fixed Length Strings* – CHAR allows you to define a string of characters (up to 30) of an exact length. If the string length is lower than the number of characters it is padded with spaces after the text. TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT store 255, 65,535, 16,777,215, and 4,294,967,296 characters respectively (no extra spaces).
- *Variable Length Strings* - VARCHAR allows you to define a string up to 65,535 characters in length – it therefore has a variable maximum length. No spaces are added.
- *Predefined String Types* – There are times where you want to store strings but not just anything. Take for example a field that stores the size of a shirt – small, medium, or large. Using ENUM (short for enumerated) you define the list of possible strings that the field can take. If you try and store any other value nothing gets stored. You can also expand this idea to use more than one value from a defined list using SET. You might recognize these string types as they are commonly used to populate dropdown menus and multiple select lists in web pages.

Other Types

There are many other types of information that are stored some common ones are

- *Dates, Times and Timestamps* – Common formats exist for representation of dates and exact points in time. These of course are just special format text strings that are standardized so that computers can interpret them (you can do the same for telephone numbers, social security numbers, etc.). DATE format is 'YYYY-MM-DD', TIME format is 'HH:MM:SS', and DATETIME is 'YYYY-MM-DD HH:MM:SS'. TIMESTAMP looks the same as DATETIME except it stores the data internal factoring in the local time zone using the Coordinated Universal Time (UTC) format (see http://www.cl.cam.ac.uk/~mgk25/iso-time.html)
- *Boolean* – True or False is stored as a TINYINT(1) where 1 is true and 0 is false
- *Binary* – Everything so far has be stored in character strings but there are situations where you might want to store byte strings – or binary. This is more technical than we need so we don't have to worry about it. It is fun to point out though that Binary Large Objects (BLOBs) can be used to store binary files like images, audio, etc.

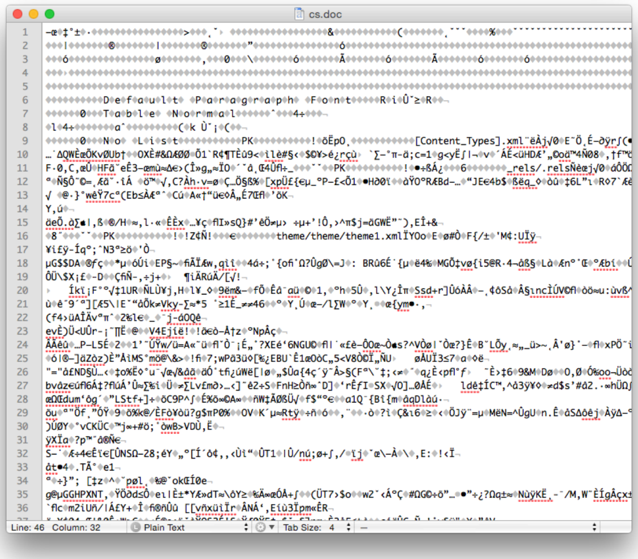
Common Computer Files and Formats

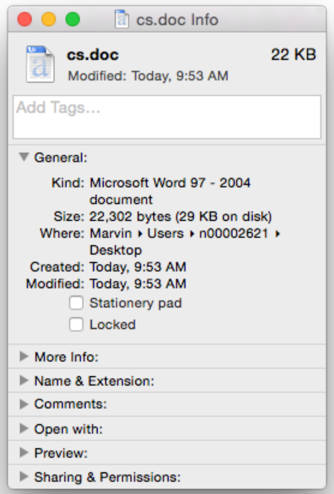
The mention of binary files in the previous sections highlights that there are differences in how data is stored by computers. Traditionally, programs or applications are stored in binary format because they contain code that is used to allow the program to run and thus should not be displayed as text (it's not readable anyway). In addition, there are many non-text based files, images, audio, video etc. that need to store their content in a format that is efficient – a place where UTF-8 (ASCII) is not needed. Historically though the most important reason for applications and files being stored as binary is that it gave the developer a way to protect their work and make money off of it. If the application they have written stores the files in a proprietary format (one that only the application can read and write) then no one else can work with the file without licensing the software used to create it.

In the early years, without the presence of the Internet this model worked well. But once we all became connected and more and more armchair software developers started developing free software, a significant push back started with this model. People wanted to be able to share documents and not have to pay high licensing fees just to be able to read a letter they sent out to their relatives. Although a lot of progress has been made in making file formats open and based on community standards, there are still a lot of issues will proprietary file formats, especially in the sciences. If you think about it, any data you get off of an instrument is stored in a proprietary format – if you want to get the data into Microsoft Excel for instance you have to export it. In a lot of cases you are not getting all the information that was gathered – but more about that later.

Microsoft is a good example of a company that used a proprietary binary file format for saving documents from all its applications, that has transitioned to an open standard text based format. In 2000 Microsoft started producing versions of its files (starting with Excel) in what was to become the Open Office eXtensible Markup Language (OOXML), a standard released by ECMA in December 2006 (see http://www.ecma-international.org/publications/standards/Ecma-376.htm). XML is a text based markup language published by the World Wide Web consortium (http://www.w3.org/XML/) that is used to annotate text strings using tags. Hypertext Markup Language (HTML) used in web browsers is a limited version of XML primarily used for presentation.

Comparing the binary Word document format (.doc file) with the new OOXML (.docx file) format and plain text (.txt file) we can see the differences (available for download online).





cs.doc Info

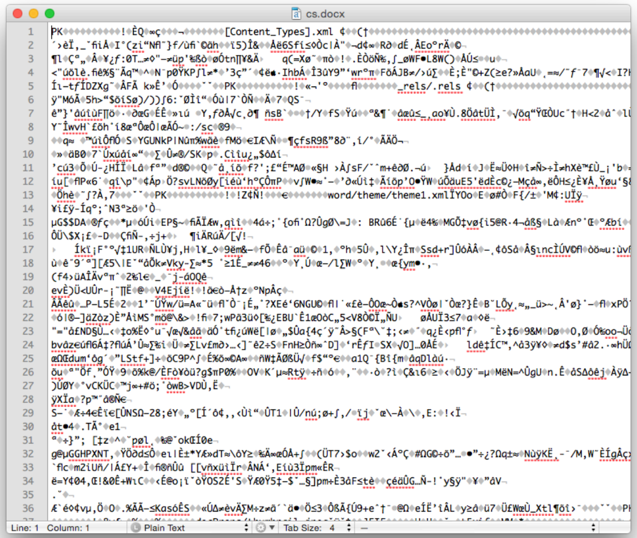
cs.doc 22 KB
Modified: Today, 9:53 AM

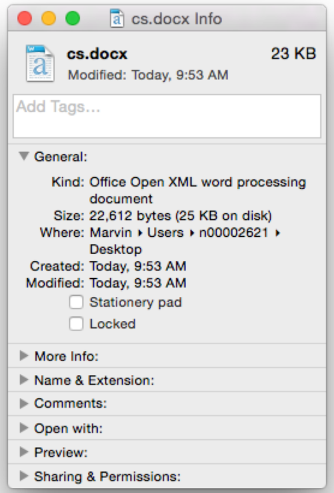
Add Tags...

General:
Kind: Microsoft Word 97 - 2004 document
Size: 22,302 bytes (29 KB on disk)
Where: Marvin ▸ Users ▸ n00002621 ▸ Desktop
Created: Today, 9:53 AM
Modified: Today, 9:53 AM
☐ Stationery pad
☐ Locked

More Info:
Name & Extension:
Comments:
Open with:
Preview:
Sharing & Permissions:

The Word .doc File Format





cs.docx Info

cs.docx 23 KB
Modified: Today, 9:53 AM

Add Tags…

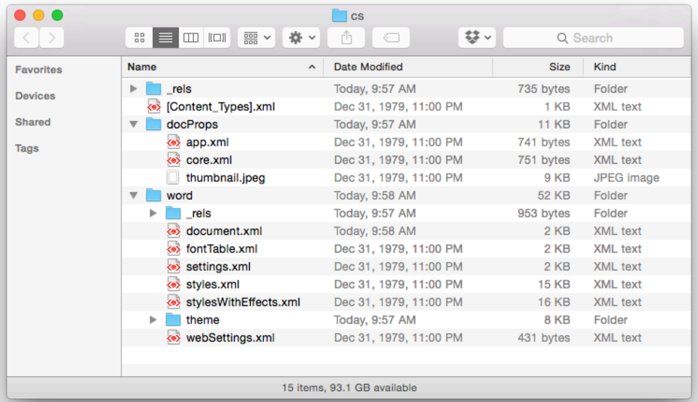
General:

Kind: Office Open XML word processing document
Size: 22,612 bytes (25 KB on disk)
Where: Marvin ▸ Users ▸ n00002621 ▸ Desktop
Created: Today, 9:53 AM
Modified: Today, 9:53 AM
☐ Stationery pad
☐ Locked

More Info:
Name & Extension:
Comments:
Open with:
Preview:
Sharing & Permissions:

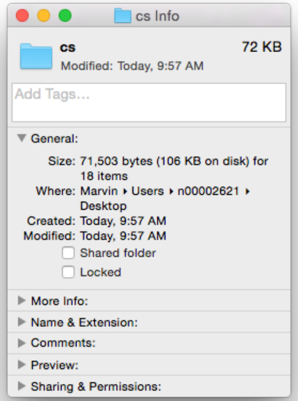
The Word .docx File Format

We see that the .doc file is unintelligible in a text reader and has over 22K characters – large considering the file contains only the text 'Chemical Informatics'. The .docx file is also unreadable and slightly larger in size – but wasn't this supposed to be in the OOXML text based format? It actually is, however the .docx file is actually a folder of XML files that is ZIP compressed into a single file. If you change the extension of a .docx file to .zip and un-compress it you see the folder of XML documents (that take up a lot of space – a problem with XML).



Name	Date Modified	Size	Kind
_rels	Today, 9:57 AM	735 bytes	Folder
[Content_Types].xml	Dec 31, 1979, 11:00 PM	1 KB	XML text
docProps	Today, 9:57 AM	11 KB	Folder
app.xml	Dec 31, 1979, 11:00 PM	741 bytes	XML text
core.xml	Dec 31, 1979, 11:00 PM	751 bytes	XML text
thumbnail.jpeg	Dec 31, 1979, 11:00 PM	9 KB	JPEG image
word	Today, 9:58 AM	52 KB	Folder
_rels	Today, 9:57 AM	953 bytes	Folder
document.xml	Today, 9:58 AM	2 KB	XML text
fontTable.xml	Dec 31, 1979, 11:00 PM	2 KB	XML text
settings.xml	Dec 31, 1979, 11:00 PM	2 KB	XML text
styles.xml	Dec 31, 1979, 11:00 PM	15 KB	XML text
stylesWithEffects.xml	Dec 31, 1979, 11:00 PM	16 KB	XML text
theme	Today, 9:57 AM	8 KB	Folder
webSettings.xml	Dec 31, 1979, 11:00 PM	431 bytes	XML text

15 items, 93.1 GB available



cs Info

cs 72 KB
Modified: Today, 9:57 AM

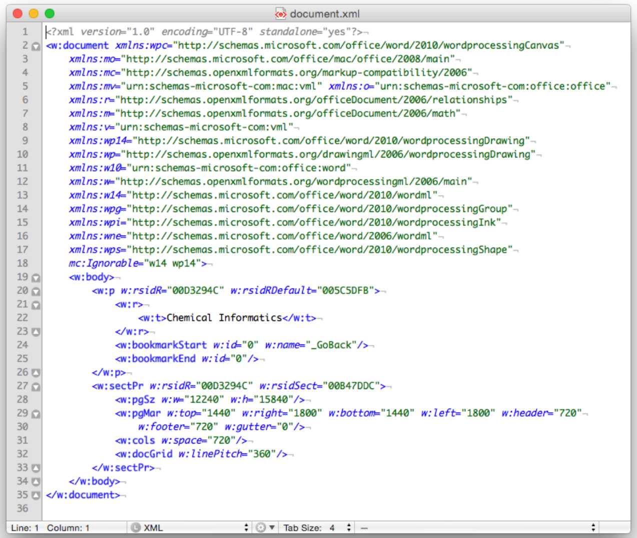
Add Tags…

General:

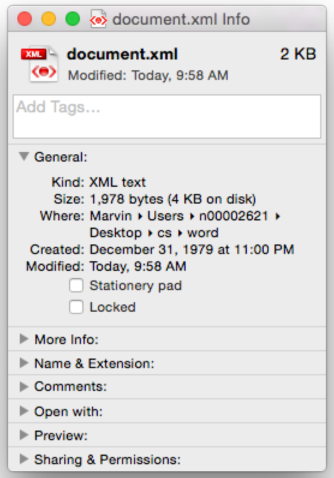
Size: 71,503 bytes (106 KB on disk) for 18 items
Where: Marvin ▸ Users ▸ n00002621 ▸ Desktop
Created: Today, 9:57 AM
Modified: Today, 9:57 AM
☐ Shared folder
☐ Locked

More Info:
Name & Extension:
Comments:
Preview:
Sharing & Permissions:

Finally, if you look inside the 'document.xml' file you find the text content. The other files in the folder are used to store information about the font used, the Word settings, the styles in use and an image of the file used by the operating system to display a thumbnail of the file content. The 'document.xml' contains the text of the file and other 'markup' – the XML – that provides the structure of the file, the layout of the page, and other important information


document.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    xmlns:mo="http://schemas.microsoft.com/office/mac/office/2008/main"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:mv="urn:schemas-microsoft-com:mac:vml" xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
    xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape"
    mc:Ignorable="w14 wp14">
    <w:body>
        <w:p w:rsidR="00D3294C" w:rsidRDefault="005C5DFB">
            <w:r>
                <w:t>Chemical Informatics</w:t>
            </w:r>
            <w:bookmarkStart w:id="0" w:name="_GoBack"/>
            <w:bookmarkEnd w:id="0"/>
        </w:p>
        <w:sectPr w:rsidR="00D3294C" w:rsidSect="00B47DDC">
            <w:pgSz w:w="12240" w:h="15840"/>
            <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800" w:header="720"
                w:footer="720" w:gutter="0"/>
            <w:cols w:space="720"/>
            <w:docGrid w:linePitch="360"/>
        </w:sectPr>
    </w:body>
</w:document>
```

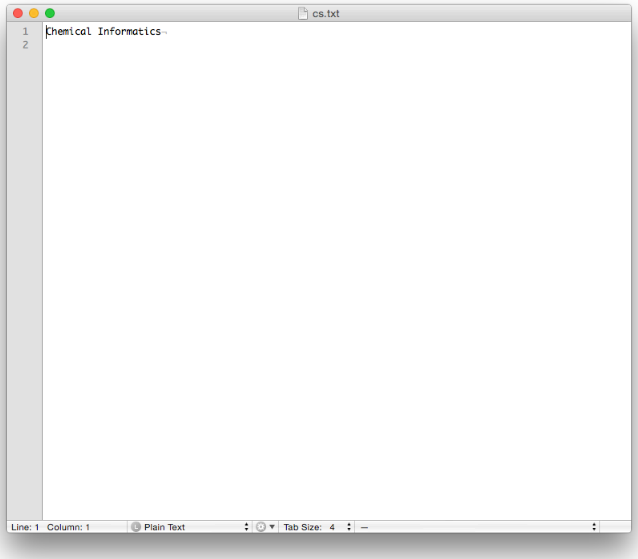

document.xml Info

document.xml 2 KB
Modified: Today, 9:58 AM

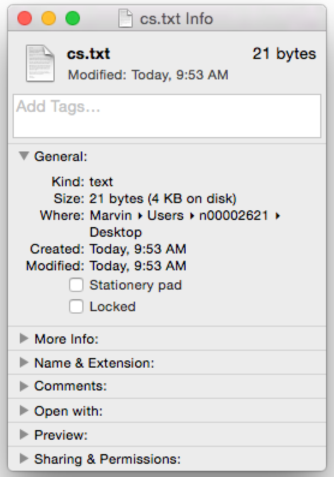
General:

Kind: XML text
Size: 1,978 bytes (4 KB on disk)
Where: Marvin ▸ Users ▸ n00002621 ▸ Desktop ▸ cs ▸ word
Created: December 31, 1979 at 11:00 PM
Modified: Today, 9:58 AM
☐ Stationery pad
☐ Locked

Of course if you just need the text (i.e. no formatting etc.) then you could use the .txt format. It's got the same size 21 bytes, as the length of the text (including the carriage return as the end of the sentence). Sometimes less, is more...


cs.txt

```
Chemical Informatics
```


cs.txt Info

cs.txt 21 bytes
Modified: Today, 9:53 AM

General:

Kind: text
Size: 21 bytes (4 KB on disk)
Where: Marvin ▸ Users ▸ n00002621 ▸ Desktop
Created: Today, 9:53 AM
Modified: Today, 9:53 AM
☐ Stationery pad
☐ Locked

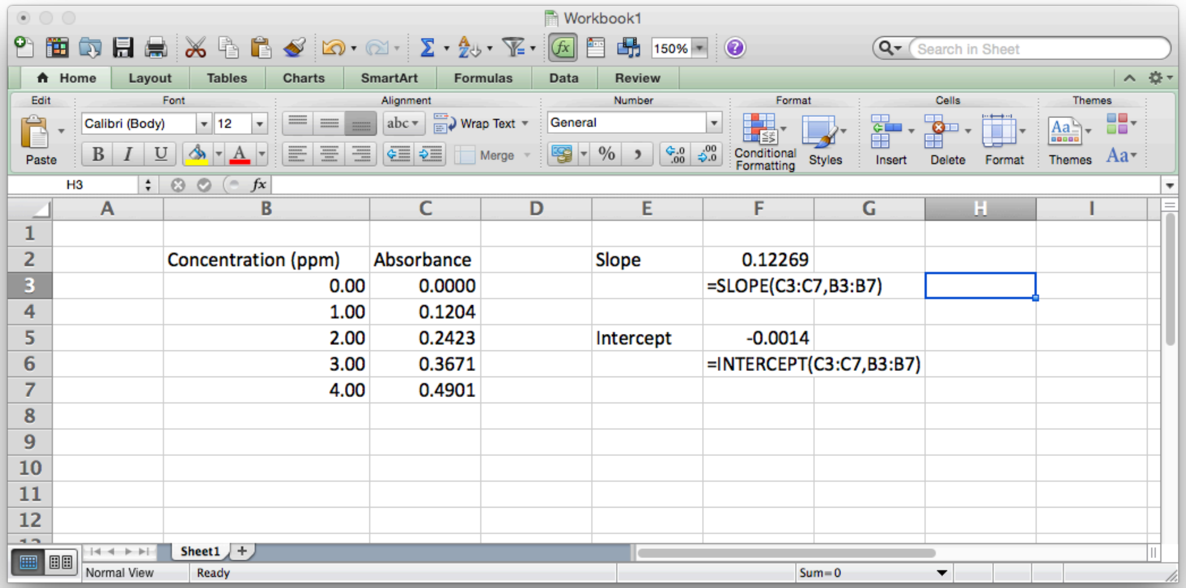
Computer Languages for Information Science

While this class is not a course in programming, it is useful to know what it is out there that can be used to manipulate information, perform calculations, and organize data. As this is an introduction we will just touch on three that scientists/web developers use, but there are many more – each with different advantages. This discussion does not include the hard code application development platforms like C++, C#, etc.

Desktop Application (Windows, Mac) – Microsoft Excel

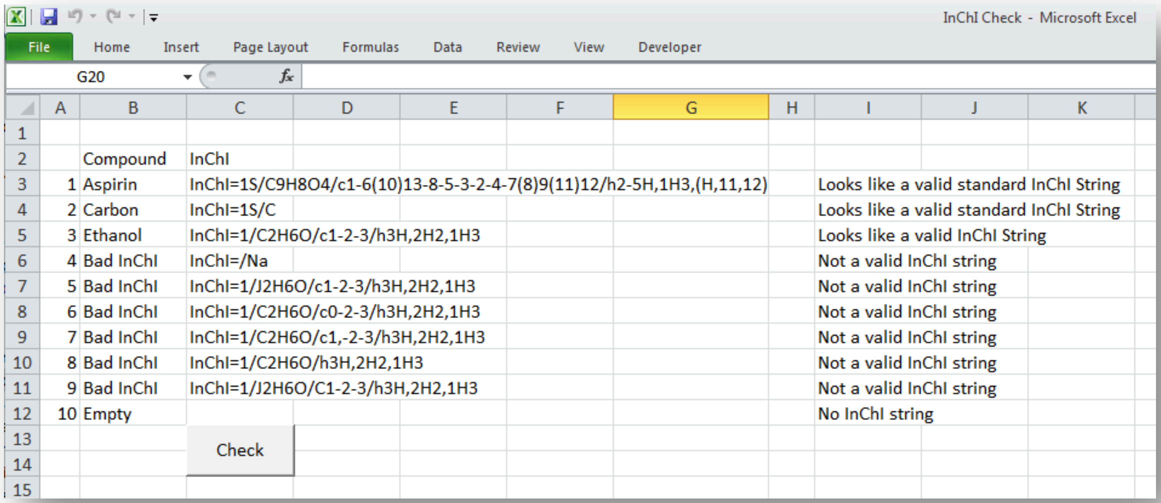
A lot of people think that Microsoft Excel is really for business people and not scientists. Certainly, Microsoft caters to the business community in terms of the table formats, graphics, and cell functions. Nonetheless, Excel has functionality that can be used in the context of informatics and, as you will see later in the course, can talk to webservers (websites) to retrieve data.

Excel has programming built-in in two ways. First, there are the in-cell functions. These are useful for text manipulation and basic math operations (see http://www.dummies.com/how-to/content/excel-formulas-and-functions-for-dummies-cheat-she.html). There are also some statistical functions like mean (AVERAGE), mode (MODE), median (MEDIAN) standard deviation (STDEV), as well as some more sophisticated functions related to linear regression (SLOPE and INTERCEPT).

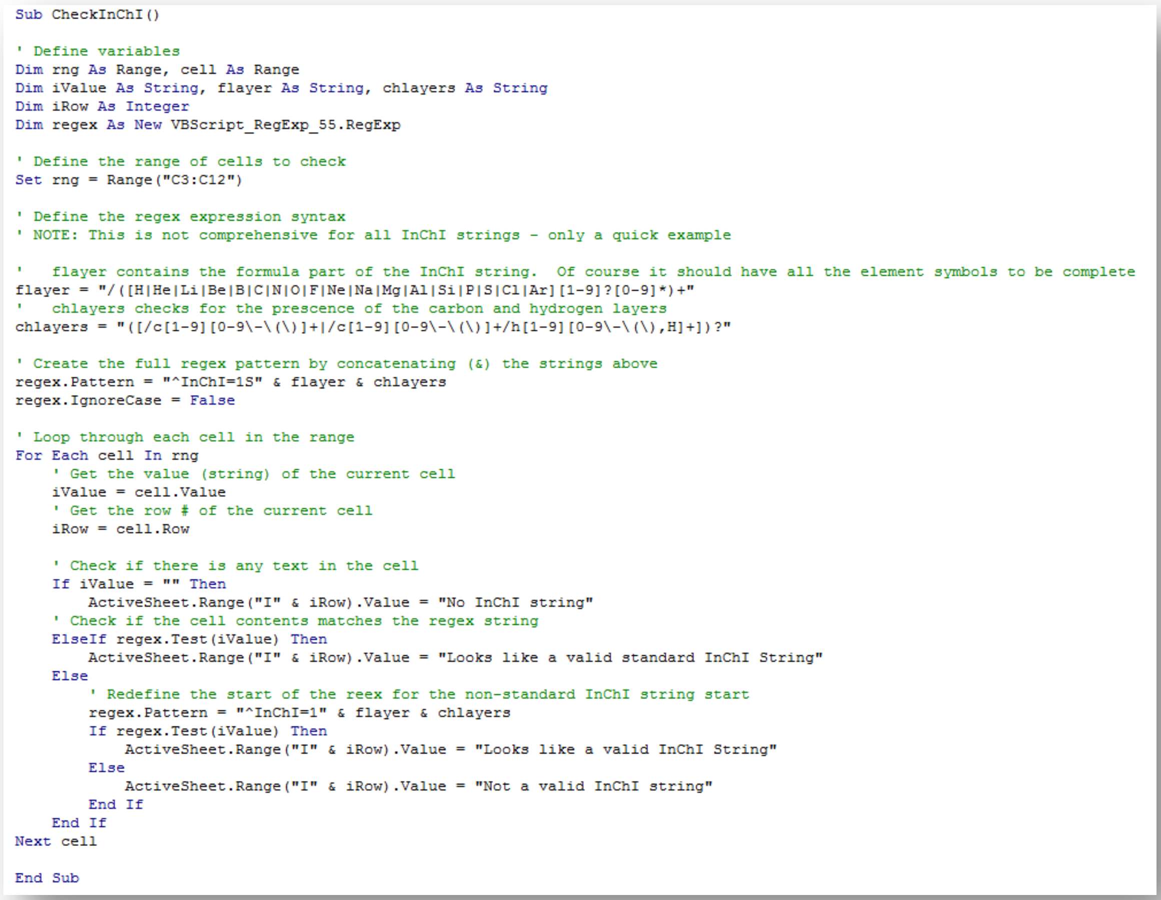


	A	B	C	D	E	F	G	H	I
1									
2		Concentration (ppm)	Absorbance		Slope	0.12269			
3		0.00	0.0000			=SLOPE(C3:C7,B3:B7)			
4		1.00	0.1204						
5		2.00	0.2423		Intercept	-0.0014			
6		3.00	0.3671			=INTERCEPT(C3:C7,B3:B7)			
7		4.00	0.4901						

Then there is the much more sophisticated Visual Basic for Applications, (VBA), which sits behind Excel and is used to record the Macro's run in Excel. VBA is much more a true programming language allowing for declaration of variables, loop structures, if-then-else conditionals and user defined functions. The script below is a quick example of how to check for strings to be a particular format – in this case the International Chemical Identifier, or InChI string (more later on this). This script tries to match the pattern that is defined to the text in each cell to tell if the string 'looks' like a valid InChI string. Whether it is would have to be tested in other ways…



	A	B	C	D	E	F	G	H	I	J	K
1											
2		Compound	InChI								
3	1	Aspirin	InChI=1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12)						Looks like a valid standard InChI String		
4	2	Carbon	InChI=1S/C						Looks like a valid standard InChI String		
5	3	Ethanol	InChI=1/C2H6O/c1-2-3/h3H,2H2,1H3						Looks like a valid InChI String		
6	4	Bad InChI	InChI=/Na						Not a valid InChI string		
7	5	Bad InChI	InChI=1/J2H6O/c1-2-3/h3H,2H2,1H3						Not a valid InChI string		
8	6	Bad InChI	InChI=1/C2H6O/c0-2-3/h3H,2H2,1H3						Not a valid InChI string		
9	7	Bad InChI	InChI=1/C2H6O/c1,-2-3/h3H,2H2,1H3						Not a valid InChI string		
10	8	Bad InChI	InChI=1/C2H6O/h3H,2H2,1H3						Not a valid InChI string		
11	9	Bad InChI	InChI=1/J2H6O/C1-2-3/h3H,2H2,1H3						Not a valid InChI string		
12	10	Empty							No InChI string		
13											
14											
15											



```
Sub CheckInChI()

' Define variables
Dim rng As Range, cell As Range
Dim iValue As String, flayer As String, chlayers As String
Dim iRow As Integer
Dim regex As New VBScript_RegExp_55.RegExp

' Define the range of cells to check
Set rng = Range("C3:C12")

' Define the regex expression syntax
' NOTE: This is not comprehensive for all InChI strings - only a quick example

'   flayer contains the formula part of the InChI string.  Of course it should have all the element symbols to be complete
flayer = "/([H|He|Li|Be|B|C|N|O|F|Ne|Na|Mg|Al|Si|P|S|Cl|Ar][1-9]?[0-9]*)+"
'   chlayers checks for the prescence of the carbon and hydrogen layers
chlayers = "([/c[1-9][0-9\-\(\)]+|/c[1-9][0-9\-\(\)]+/h[1-9][0-9\-\(\),H]+])?"

' Create the full regex pattern by concatenating (&) the strings above
regex.Pattern = "^InChI=1S" & flayer & chlayers
regex.IgnoreCase = False

' Loop through each cell in the range
For Each cell In rng
    ' Get the value (string) of the current cell
    iValue = cell.Value
    ' Get the row # of the current cell
    iRow = cell.Row

    ' Check if there is any text in the cell
    If iValue = "" Then
        ActiveSheet.Range("I" & iRow).Value = "No InChI string"
    ' Check if the cell contents matches the regex string
    ElseIf regex.Test(iValue) Then
        ActiveSheet.Range("I" & iRow).Value = "Looks like a valid standard InChI String"
    Else
        ' Redefine the start of the reex for the non-standard InChI string start
        regex.Pattern = "^InChI=1" & flayer & chlayers
        If regex.Test(iValue) Then
            ActiveSheet.Range("I" & iRow).Value = "Looks like a valid InChI String"
        Else
            ActiveSheet.Range("I" & iRow).Value = "Not a valid InChI string"
        End If
    End If
Next cell

End Sub
```

Command Line Application (Any platform) – R

R (yes just the letter R) is a very popular statistical computing and graphics environment (see https://www.r-project.org/). R is sophisticated and extremely powerful so there is no way to show off its capabilities in a brief introduction, but to give you a flavor here are some things you can do. R is natively a command line application, so it does not have a graphical user interface (GUI) built-in that means it launches like other applications – you run it from the command line in the terminal. There are however many GUI's for R – for example RStudio (https://www.rstudio.com).

In order to get data into R to do something with it you must either enter it or import it from a text file. If we look back at the linear regression example from Excel we see we have two arrays of data, the x data points (concentration) and the y data points (absorbance). If we wanted to enter these into R we would do it like this.

```
sh-3.2# R (run R)
> xpoints <- c(0.00,1.00,2.00,3.00,4.00)
> ypoints <- c(0.0000,0.1204,0.2423,0.3671,0.4901)
> lsfit(xpoints,ypoints)
$coefficients
Intercept         X
 -0.00140   0.12269

$residuals
[1]  0.00140 -0.00089 -0.00168  0.00043  0.00074
…
```

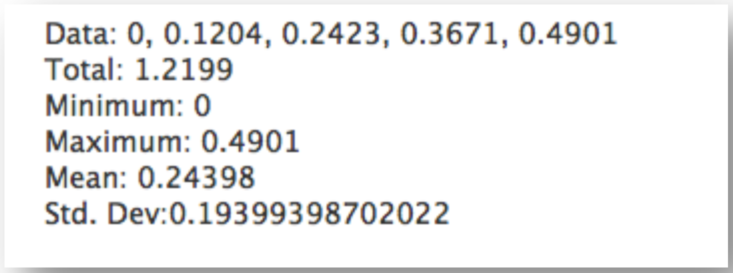
We enter the x and y data points as matrixes (you can also think of them as arrays) and then use them as inputs to the lsfit (least squares fit) function that is defined in R. It immediately spits out the slope, intercept and residuals (the differences between the y values and the calculated y values from the equation) as well as a lot of other details not shown above.

Browser Application (Any platform) – PHP

In the early days of the Internet every webpage you accessed was static, that is, it was a page of text in a file that had to be updated manually to change it. Today, the majority of websites are built using scripts that dynamically created the page when it is requested. This is because we expect sophistication (features), information (content), and immediacy (its up to date). Therefore, the web server needs to things to create the web pages: content (typically stored in a database – see the next section) and a scripting language that can logically build the page based on a set of commands. Current the most popular language to do this is called Pre-Hypertext Processor, or PHP (see http://php.net/).

Just like R, it is impossible to show how capable PHP is in generating web pages on the fly, but a brief example script below shows how to calculate some statistics in a web page, with the output shown below it.

```
<?php
$data = [0.0000,0.1204,0.2423,0.3671,0.4901];
echo "Data: ".implode(", ",$data)."<br />";
echo "Total: ".array_sum($data)."<br />";
echo "Minimum: ".min($data)."<br />";
echo "Maximum: ".max($data)."<br />";
$mean=array_sum($data)/count($data);
echo "Mean: ".$mean."<br />";
$sumsquares=0;
foreach($data as $point) {
    $sumsquares+=pow($point-$mean,2);
}
echo "Std. Dev: ".sqrt($sumsquares/(count($data)-1));
?>
```



```
Data: 0, 0.1204, 0.2423, 0.3671, 0.4901
Total: 1.2199
Minimum: 0
Maximum: 0.4901
Mean: 0.24398
Std. Dev:0.19399398702022
```

Storing Information in Databases

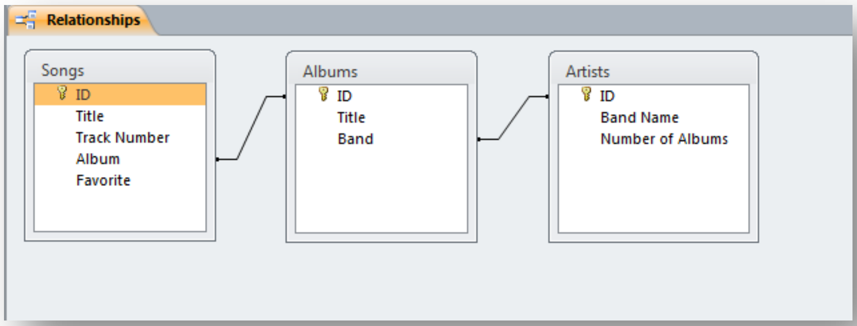
A database is a system that stores information in a logical, organized format. Many applications on your computer (contacts, calendar, mail reader) rely on databases as the places to store the information that needs to be displayed in a way that allows the application to easily go grab the data.

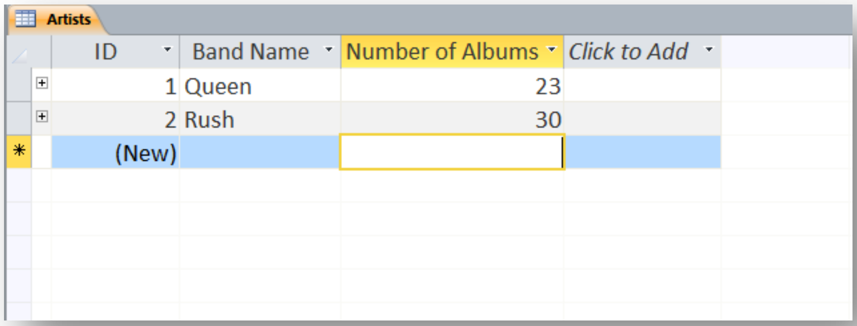
While there are a number of styles of database the most common currently are relational. This means that data in different 'tables' is related together by the use of a unique id field, also called a foreign key. Each row of a table is given a unique value and this is then entered in a field in another table for any rows of data that are related to the information in the first table. Sounds complicated but it's easy to see it in action in the upcoming examples.

Desktop Database

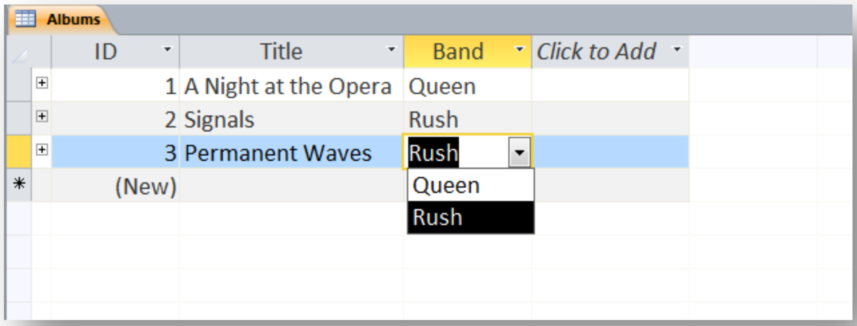
Microsoft Access has been a staple part of the Microsoft Office suite of applications since 1992. It often goes overlooked as there are a wider variety of uses for Word, Excel, and PowerPoint, but it is a powerful relational database that has found great use in business.

Users can define multiple tables of data (one entry is a row with multiple fields (columns)) and then link them together using unique ids that are automatically created on each row. Columns can be different data types (text, numeric, date/time, currency, etc.) or relational (lookup). The example below shows a small set of artists, albums, and songs that are related together via fields defined in the tables. The relationships can be see in the last image with the 'Album' field of the 'Songs' table being linked to the ID field of the 'Albums' table, and the 'Band' field of the 'Albums' table being linked to the ID field of the 'Artists' table. As can be seen in the figures, entering data in the linked fields is done through dropdown menu's (to control what text is in the field) making it easier to create the table and providing the relationships.

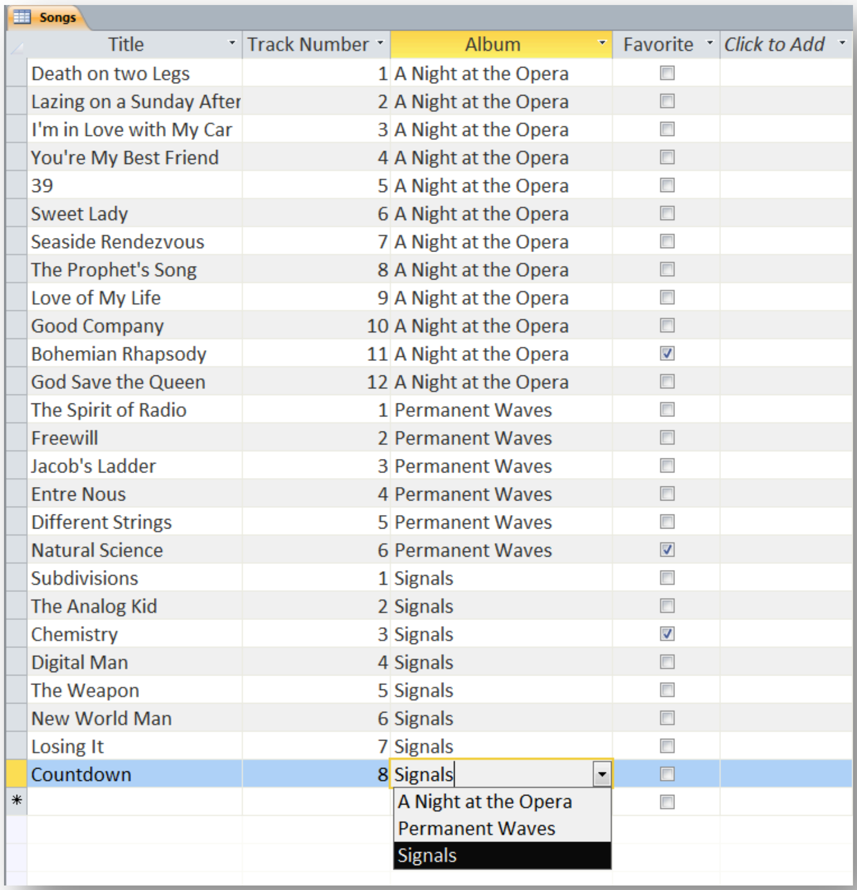



Artists

ID	Band Name	Number of Albums	Click to Add
1	Queen	23	
2	Rush	30	
(New)			


Albums

ID	Title	Band	Click to Add
1	A Night at the Opera	Queen	
2	Signals	Rush	
3	Permanent Waves	Rush	
(New)			

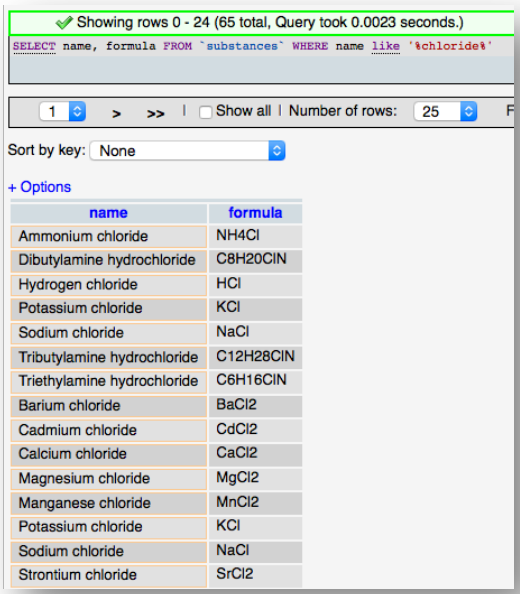


Songs

Title	Track Number	Album	Favorite	Click to Add
Death on two Legs	1	A Night at the Opera	☐	
Lazing on a Sunday After	2	A Night at the Opera	☐	
I'm in Love with My Car	3	A Night at the Opera	☐	
You're My Best Friend	4	A Night at the Opera	☐	
39	5	A Night at the Opera	☐	
Sweet Lady	6	A Night at the Opera	☐	
Seaside Rendezvous	7	A Night at the Opera	☐	
The Prophet's Song	8	A Night at the Opera	☐	
Love of My Life	9	A Night at the Opera	☐	
Good Company	10	A Night at the Opera	☐	
Bohemian Rhapsody	11	A Night at the Opera	✓	
God Save the Queen	12	A Night at the Opera	☐	
The Spirit of Radio	1	Permanent Waves	☐	
Freewill	2	Permanent Waves	☐	
Jacob's Ladder	3	Permanent Waves	☐	
Entre Nous	4	Permanent Waves	☐	
Different Strings	5	Permanent Waves	☐	
Natural Science	6	Permanent Waves	✓	
Subdivisions	1	Signals	☐	
The Analog Kid	2	Signals	☐	
Chemistry	3	Signals	✓	
Digital Man	4	Signals	☐	
The Weapon	5	Signals	☐	
New World Man	6	Signals	☐	
Losing It	7	Signals	☐	
Countdown	8	Signals	☐	
*			☐	

<u>Web-based Databases</u>
Currently, the world of relational database is primarily geared toward the use of the Structured Query Language (SQL). SQL is a special programming language for retrieving data out of databases originally developed at IBM in the early 1970s. Relational Software (now called Oracle) created the first commercial software to implement SQL in the late 70s – called Oracle V2, and has subsequently become a major player in the database industry. Currently, there are many implementations of SQL databases, both commercial and open source (free), and most of them are accessible from scripting languages so they can be considered 'Web-based' databases.

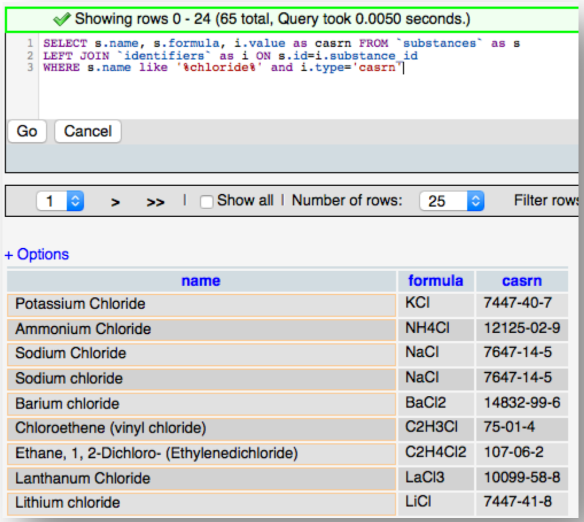
Although it grew out of the relational database model, SQL has deviated somewhat to include features and provide functionality needed for database applications. The basic structure of SQL is an English language sentence comprised of multiple parts using defined keywords. 'SELECT' is used to indicate that data is to be retrieved, 'FROM' is used to identify the table(s) from which the data is to be obtained, and 'WHERE' defines conditions on which the command searches. So the following SQL query (left) selects all the substance that have chloride ('%' is used as a wildcard) in the their name and returns the names and formulas of those entries found (rather than all the fields (columns) in the substances table).



Showing rows 0 - 24 (65 total, Query took 0.0023 seconds.)

```
SELECT name, formula FROM `substances` WHERE name like '%chloride%'
```

name	formula
Ammonium chloride	NH4Cl
Dibutylamine hydrochloride	C8H20ClN
Hydrogen chloride	HCl
Potassium chloride	KCl
Sodium chloride	NaCl
Tributylamine hydrochloride	C12H28ClN
Triethylamine hydrochloride	C6H16ClN
Barium chloride	BaCl2
Cadmium chloride	CdCl2
Calcium chloride	CaCl2
Magnesium chloride	MgCl2
Manganese chloride	MnCl2
Potassium chloride	KCl
Sodium chloride	NaCl
Strontium chloride	SrCl2



Showing rows 0 - 24 (65 total, Query took 0.0050 seconds.)

```
SELECT s.name, s.formula, i.value as casrn FROM `substances` as s
LEFT JOIN `identifiers` as i ON s.id=i.substance_id
WHERE s.name like '%chloride%' and i.type='casrn'
```

name	formula	casrn
Potassium Chloride	KCl	7447-40-7
Ammonium Chloride	NH4Cl	12125-02-9
Sodium Chloride	NaCl	7647-14-5
Sodium chloride	NaCl	7647-14-5
Barium chloride	BaCl2	14832-99-6
Chloroethene (vinyl chloride)	C2H3Cl	75-01-4
Ethane, 1, 2-Dichloro- (Ethylenedichloride)	C2H4Cl2	107-06-2
Lanthanum Chloride	LaCl3	10099-58-8
Lithium chloride	LiCl	7447-41-8

The more complicated query (right) adds the 'JOIN' syntax that is used to create relationships between tables on the fly so that related data can be retrieved in a single query. The designation LEFT JOIN means that all the data from the 'LEFT' (first) table is including in the results even if there is no related data in the 'RIGHT' (second) table. The keyword 'AS' is used to create an alias to a table to make the query shorter, and the keyword 'ON' is used to define the fields to join between the tables.

One current trend of note is the movement toward 'NoSQL' databases. This is somewhat of a misnomer as the class of databases this commonly refers to are really non-relational in nature and as a result there is no need to use SQL queries (although some do). The database model types lumped into the NoSQL classification are: Column, Document, Key-Value, Graph, and Multi-model based. Many of these have only been developed in the last few years as a response to interest in big data – applications where very fast processing of large (distributed) datasets is required. Graph based databases are of interest in many situations due to their lack of a data model and representation of 'semantic' data or subject-predicate-object triples. These databases have their own query language, SPARQL which is the interestingly recursive acronym that stands for 'SPARQL Protocol and RDF Query Language'.

Accessing Data on Websites and Application Programming Interfaces (APIs)

To bring this module to a close it is important to discuss the ramifications of all that we have covered in the previous sections. By now you will hopefully have come to the realization that i) there is a lot of information/data in the world, ii) organizing it and identifying it is a important and huge undertaking, and iii) accessing chemistry related data in this environment cannot be done by a Google search. A lot of what chemical informatics is about is providing ways to deal with lots of data, describing best practices of how to identify and store information, and developing tools that allow scientists to find and retrieve quality information that can be used for teaching and research.

One of the fundamental parts of this is understanding where good chemistry data is, working out how to efficiently search it – based on chemical terms – and how to download it so that you can use it. In the previous sections you have seen examples of how data can be processed and stored in databases, but how do you get the data? It turns out that bringing SQL databases and scripting languages together to build webpages is only one facet of their usefulness. Increasingly websites are being built to serve up data, that is, they are designed to allow the user to search databases via well-defined standards and return the results for both human and computer use.

Over the last few years the design of websites that follow the Representational State Transfer (REST) paradigm have become popular as their design allows websites to become web services. What that means is if you know how to construct the URLs for pages on these websites you can anticipate the information that you are going to see on the page that is returned. This concept is so much easier to see in practice.

One website that has a large amount of chemical metadata (information about chemicals – not chemical data like melting points etc.) is the NIH Chemical Identifier Resolver (CIR) website (see http://cactus.nci.nih.gov/chemical/structure). The website allows you to get information by writing it in the general format below.

```
http://cactus.nci.nih.gov/chemical/structure/"structure identifier"/"representation"
```

In this context "Structure Identifier" means any of the following: chemical names, SMILES, InChI Strings and Keys, and NIH identifiers like FICTS and FICuS. What you get back from a search is determined by the "representation" part of the URL and includes all of those above and 'sdf' (otherwise known as MOL file), 'formula' and 'image'. So, using the URL below:

```
http://cactus.nci.nih.gov/chemical/structure/arsinic acid/names
```

prints out the names of this compound that are in the CIR system – for humans to read. If you use a computer to access the site (i.e. via a scripting language like PHP) you can also request the data be sent to you as XML using.

```
http://cactus.nci.nih.gov/chemical/structure/arsinic acid/names/xml
```

which returns the following document.

```
<request string="arsinic acid" representation="names">
  <data id="1" resolver="name_by_opsin" notation="arsinic acid">
    <item id="1" classification="pubchem_iupac_name">arsinic acid</item>
    <item id="2" classification="pubchem_substance_synonym">CHEBI:29840</item>
    <item id="3" classification="pubchem_substance_synonym">HAsH2O2</item>
    <item id="4" classification="pubchem_substance_synonym">[AsH2O(OH)]</item>
    <item id="5" classification="pubchem_substance_synonym">arsinic acid</item>
    <item id="6" classification="pubchem_substance_synonym">dihydridohydroxidooxidoarsenic</item>
  </data>
</request>
```

This is useful because it contains additional information (metadata) about the type of name that an entry is and it makes it easy for a script to take the data and do something with it – like put it in another database.

There are many sites that chemists can use to get chemical data, ChemSpider (http://www.chemspider.com/), PubChem (https://pubchem.ncbi.nlm.nih.gov/), and the NIST Webbook (http://webbook.nist.gov/chemistry/). Each has a different set of data but all have a standard way to interact with their website. To encourage users to take full advantage of the search features many sites will publish the Application Programming Interface (API) that delineates what you can search for and how to do it. Facebook has an API, Twitter has an API, Instagram… the list goes on. API's are the way in which websites can ''talk" to other websites to integrate data, or present 'widgets' that allow you to see what's happening on one site when you are on another. A good example of a well documented and sophisticated API for chemistry data is the one at PubChem (see https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html) which has a nice tutorial at https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html. With over 68 million compounds to search you are sure to find something!