THE UNIVERSITY OF
MELBOURNE

COMP20008 Elements of Data Processing

Data formats: structured, unstructured and semi-structured

Announcements

- Lecture venues have changed
  - Monday 0900-1000: Redmond Barry-103 (Lowe Theatre)
  - Friday 0900-10:00: Baldwin Spencer-101 (Theatre)

- Workshop for next week (Week 2) is available

- Student representatives
  - Lawson Wang Wills (lawsonw1@student.unimelb.edu.au)
  - David Cochrane-Davis (dcochrane@student.unimelb.edu.au)

- Project dates
  - Am finalising and will announce on Monday

Data formats

- Where is the data?

- How is data stored and in what formats?
  - *Structured*: Relational databases, CSV
  - *Unstructured*: text
  - *Semi-structured*: HTML, XML, JSON

- Question: Why do we have different data formats and why do we wish to transform between different formats?

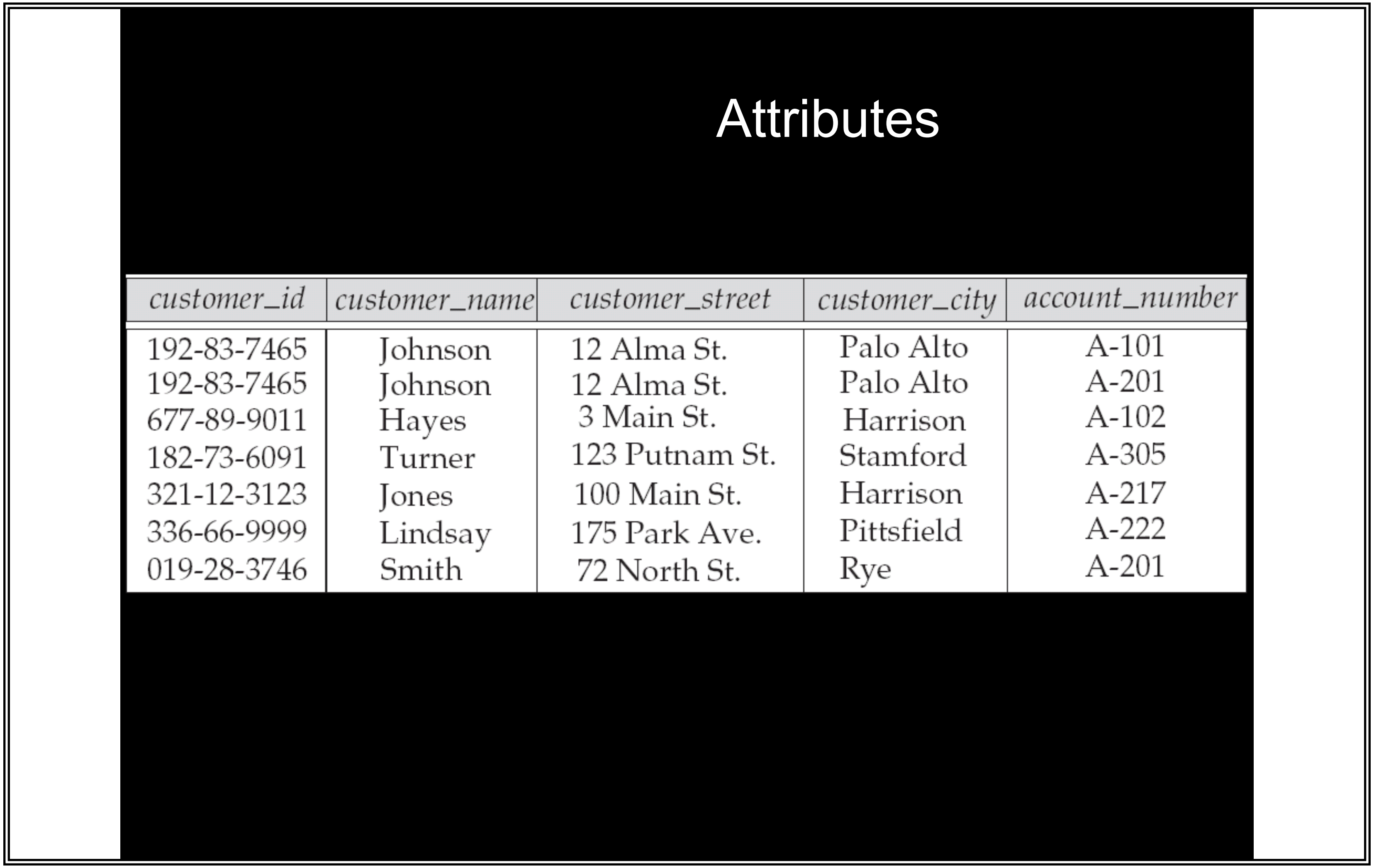
- Our purpose for next 2 lectures
  - *To understand differences between and motivation/purposes of these formats*

Relational databases (INFO20003)

- It is good to have structure for data!
  - Easier to analyse, easier to query
  - Easier to store
  - Easier to clean, maintain consistency and security, especially with multiple users

- Relational databases, the classic method of storing structured data (banking, sales, airlines …)
  - Data stored in tables, each row is a data item and columns describe attributes of the data item
  - Can query the data using a high level language such as SQL

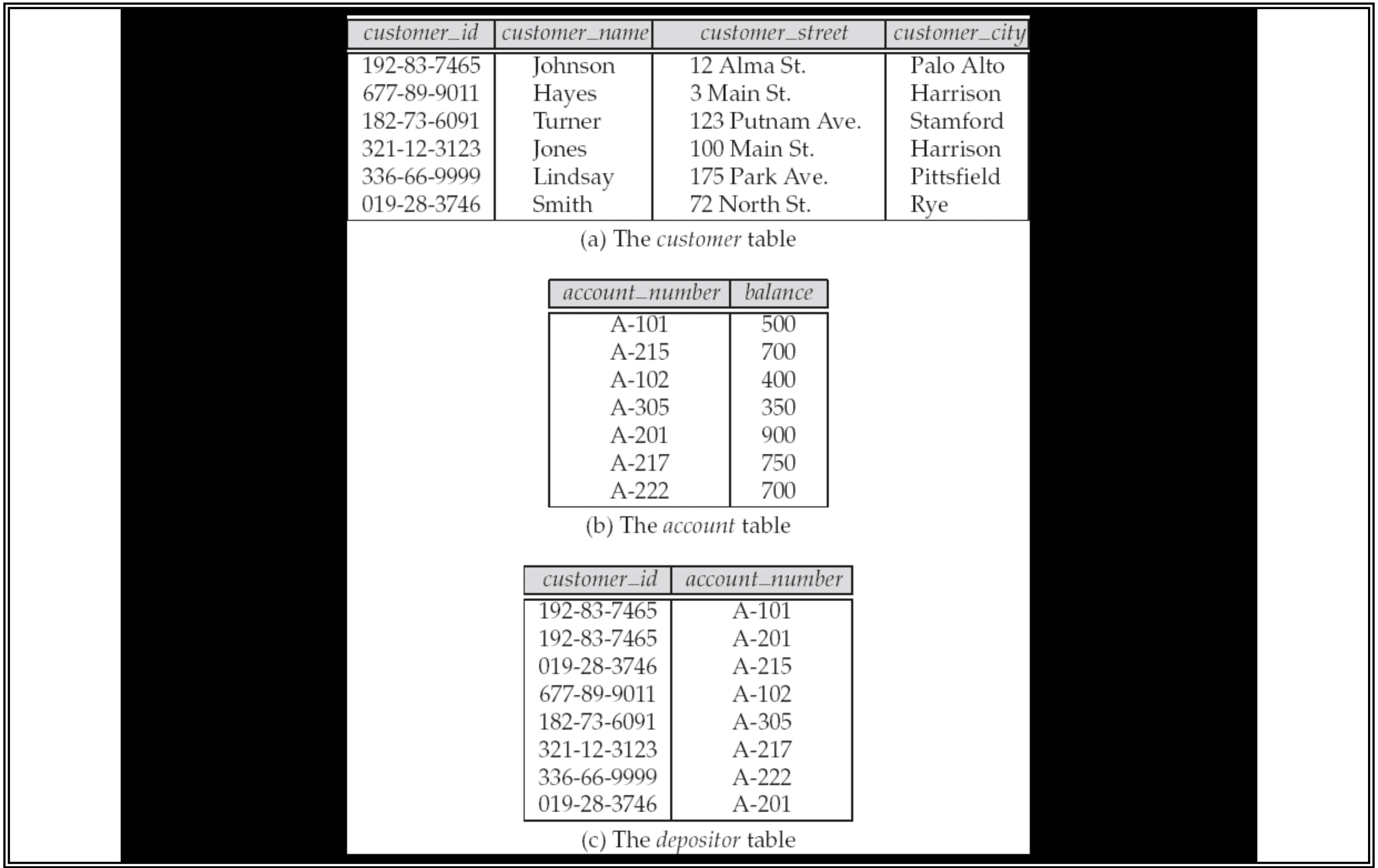
Examples from Silberschatz et al “Database System Concepts”



Attributes

customer_id	*customer_name*	*customer_street*	*customer_city*	*account_number*
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

Sample relational database



customer_id	*customer_name*	*customer_street*	*customer_city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer_id	*account_number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

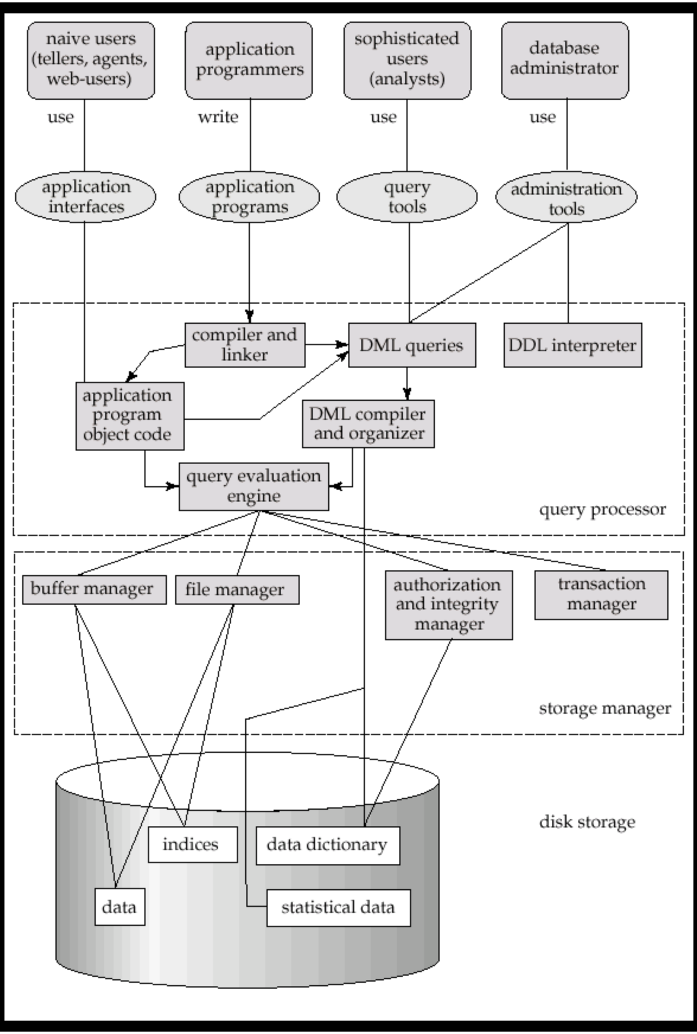
SQL

-
  ```
  create table branch
                 (branch_name   char(15) not null,
                  branch_city   char(30),
                  assets        integer,
                  primary key (branch_name))
  ```

SQL

```
select          account.balance
from            depositor, account
where           depositor.customer_id = ‘192-83-7465’
and
depositor.account_number=account.account_number
```

Database system structure



Database systems

- In INFO20003 subject you would cover topics like
  - SQL
  - Specification of integrity constraints
  - Data modelling and relational database management systems
  - Transactions and concurrency control
  - Storage management
  - Web-based databases
  - ….

- Highly relevant to data wrangling!
  - Useful to do INFO20003 as part of a data science specialisation

Challenges

- Once data is into a relational database, it is easier to wrangle.
  - But maybe hard to load it there in the first place …
    - Unstructured data: text, HTML - lack regularity

More structure - Spreadsheets: CSV

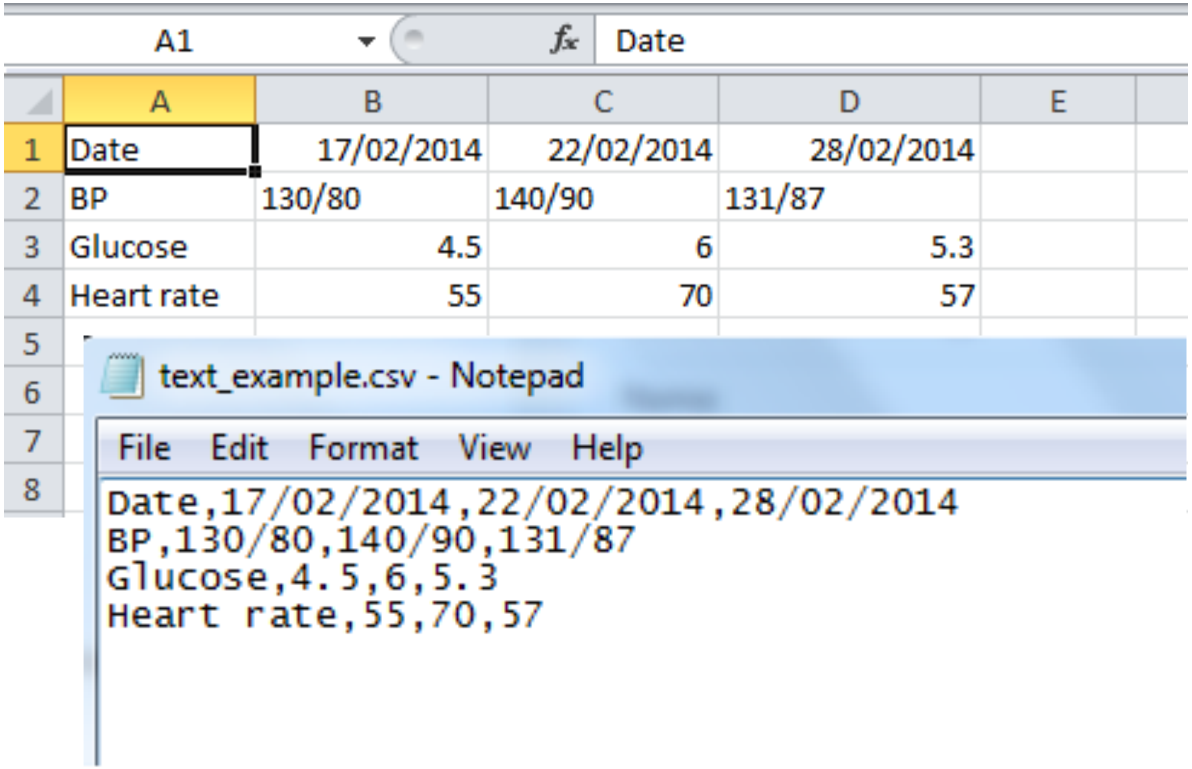
- Huge amounts of data lives in spreadsheets
  - Businesses
  - Hospitals
  - ….
- Microsoft (Excel), OpenOffice (Calc), Google docs
- CSV (comma separated values) also very popular
  - These are human readable, versus binary XLS format (Excel)
  - CSVs lack the formatting information of an XLS file
- Python libraries
  - csv
  - xlrd, openpyxl
  - pandas read_csv function

Example – Comma Separated Values (CSV)

- Spreadsheets

- Easy to use

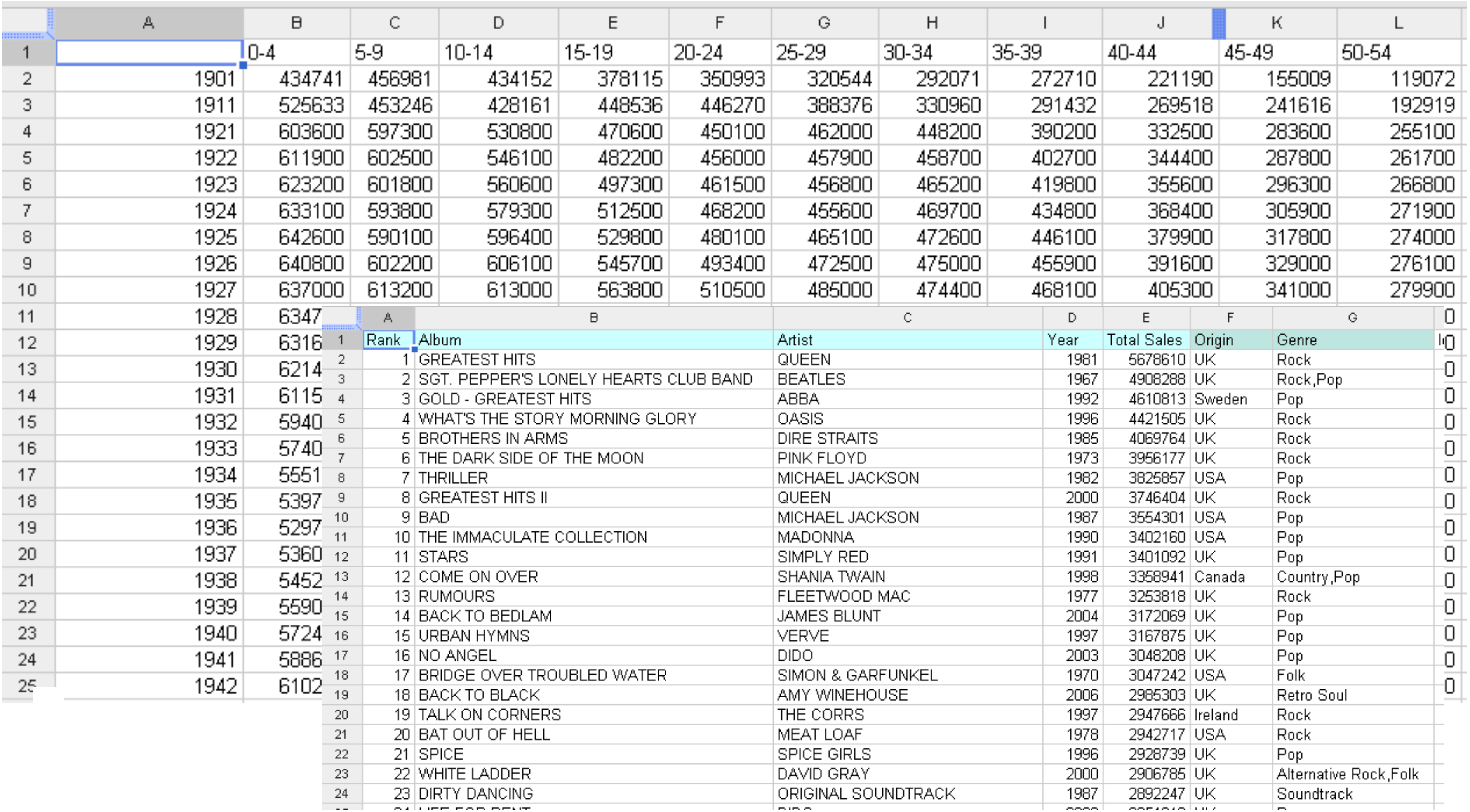
- Structured, but not like a relational DB



	A	B	C	D	E
1	Date	17/02/2014	22/02/2014	28/02/2014	
2	BP	130/80	140/90	131/87	
3	Glucose	4.5	6	5.3	
4	Heart rate	55	70	57	
5					
6					
7					
8					

```
Date,17/02/2014,22/02/2014,28/02/2014
BP,130/80,140/90,131/87
Glucose,4.5,6,5.3
Heart rate,55,70,57
```

information in tabular format (transactional, simple but many entries)

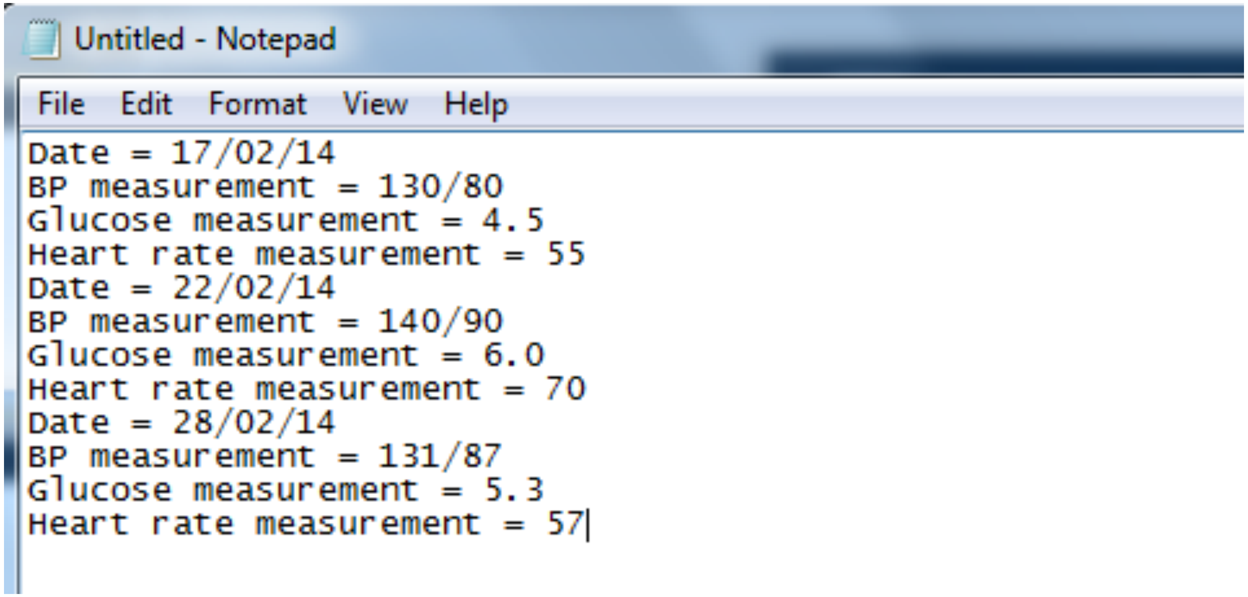


	0-4	5-9	10-14	15-19	20-24	25-29	30-34	35-39	40-44	45-49	50-54
1901	434741	456981	434152	378115	350993	320544	292071	272710	221190	155009	119072
1911	525633	453246	428161	448536	446270	388376	330960	291432	269518	241616	192919
1921	603600	597300	530800	470600	450100	462000	448200	390200	332500	283600	255100
1922	611900	602500	546100	482200	456000	457900	458700	402700	344400	287800	261700
1923	623200	601800	560600	497300	461500	456800	465200	419800	355600	296300	266800
1924	633100	593800	579300	512500	468200	455600	469700	434800	368400	305900	271900
1925	642600	590100	596400	529800	480100	465100	472600	446100	379900	317800	274000
1926	640800	602200	606100	545700	493400	472500	475000	455900	391600	329000	276100
1927	637000	613200	613000	563800	510500	485000	474400	468100	405300	341000	279900
1928	6347										
1929	6316										
1930	6214										
1931	6115										
1932	5940										
1933	5740										
1934	5551										
1935	5397										
1936	5297										
1937	5360										
1938	5452										
1939	5590										
1940	5724										
1941	5886										
1942	6102										

Rank	Album	Artist	Year	Total Sales	Origin	Genre
1	GREATEST HITS	QUEEN	1981	5678610	UK	Rock
2	SGT. PEPPER'S LONELY HEARTS CLUB BAND	BEATLES	1967	4908288	UK	Rock,Pop
3	GOLD - GREATEST HITS	ABBA	1992	4610813	Sweden	Pop
4	WHAT'S THE STORY MORNING GLORY	OASIS	1996	4421505	UK	Rock
5	BROTHERS IN ARMS	DIRE STRAITS	1985	4069764	UK	Rock
6	THE DARK SIDE OF THE MOON	PINK FLOYD	1973	3956177	UK	Rock
7	THRILLER	MICHAEL JACKSON	1982	3825857	USA	Pop
8	GREATEST HITS II	QUEEN	2000	3746404	UK	Rock
9	BAD	MICHAEL JACKSON	1987	3554301	USA	Pop
10	THE IMMACULATE COLLECTION	MADONNA	1990	3402160	USA	Pop
11	STARS	SIMPLY RED	1991	3401092	UK	Pop
12	COME ON OVER	SHANIA TWAIN	1998	3358941	Canada	Country,Pop
13	RUMOURS	FLEETWOOD MAC	1977	3253818	UK	Rock
14	BACK TO BEDLAM	JAMES BLUNT	2004	3172069	UK	Pop
15	URBAN HYMNS	VERVE	1997	3167875	UK	Pop
16	NO ANGEL	DIDO	2003	3048208	UK	Pop
17	BRIDGE OVER TROUBLED WATER	SIMON & GARFUNKEL	1970	3047242	USA	Folk
18	BACK TO BLACK	AMY WINEHOUSE	2006	2985303	UK	Retro Soul
19	TALK ON CORNERS	THE CORRS	1997	2947666	Ireland	Rock
20	BAT OUT OF HELL	MEAT LOAF	1978	2942717	USA	Rock
21	SPICE	SPICE GIRLS	1996	2928739	UK	Pop
22	WHITE LADDER	DAVID GRAY	2000	2906785	UK	Alternative Rock,Folk
23	DIRTY DANCING	ORIGINAL SOUNDTRACK	1987	2892247	UK	Soundtrack

Unstructured data - Text

- Text files…
- No structure
- Harder to index



```
Date = 17/02/14
BP measurement = 130/80
Glucose measurement = 4.5
Heart rate measurement = 55
Date = 22/02/14
BP measurement = 140/90
Glucose measurement = 6.0
Heart rate measurement = 70
Date = 28/02/14
BP measurement = 131/87
Glucose measurement = 5.3
Heart rate measurement = 57
```

- Harder to organise
- Lacks regularity and decomposable internal structure
- How can we process/search for information?

Text data (documents ..)

- Specifying patterns in text – regular expressions
  - Good for computing statistics, checking integrity, filtering, substitutions ….
- Specifying patterns in text
  - ‘.’ matches any character
  - ‘^’ matches start of string
  - ‘$’ matches end of string
  - ‘*’ zero or more repetitions
  - ‘+’ one or more repetitions
  - ‘|’ the “or” operator
  - ‘[]’ a set of characters, e.g. [abcd] or [a-zA-Z]
- https://docs.python.org/2/howto/regex.html
- regex101.com

Exercises (3 minutes)

- Write regular expressions to specify each of the following
  - Two occurrences of letter ‘e’ followed immediately by one ‘n’ and then at least one ‘t’
  - An ‘h’ or an ‘e’ or an ‘x’, followed by at least one `a’, followed by an ‘r’
  - Any 3 characters, possibly followed by a repeated sequence of the character ‘x’, followed by a ‘c’ or a ‘d’

More complex regular expression

- What do you think this pattern is for?
  - [a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+
  - Could it be improved?

HTML – Hypertext Markup language

- Marked up with *elements*, delineated by start and end *tags*. Elements correspond to logical units, such as a heading, paragraph or itemised list.
- *Tags*: Keywords contained in pairs of angle brackets.
  - Not case sensitive.
- Browser determines how to display/present the logical units
- Not all elements need both start and end tags.
- Some elements can have *attributes*. Ordering of attributes is not significant.

HTML Example

```
<div class="icon section5">
<hh2><a href="about/index.html">About the Melbourne School of Engineering</
a></h2>
<ul>
<li><a href="about/dean_welcome.html">Dean's Welcome</a></li>
<li><a href="about/staff.html">Leadership &amp; Professional
Staff</a></li>
<li><a href="about/contact.html">Contact Us</a></li>
<li><a href="http://www.ecr.unimelb.edu.au">ECR: Computer
Resources</a></li>
<li><a href="intranet/index.html">For Staff (intranet)</a></li>
<li><a href="casual_staff/index.html">For Casual Staff</a></li>
<li><a href="intranet/review/prof_staff.html">Professional Staff
Review</a></li>
<li><a href="/about/safety/index.html">Environment, Health &amp;
Safety</a></li>
<li><a href="/about/committees/index.html">Committees</a></li>
</ul>
```

Limitations of HTML

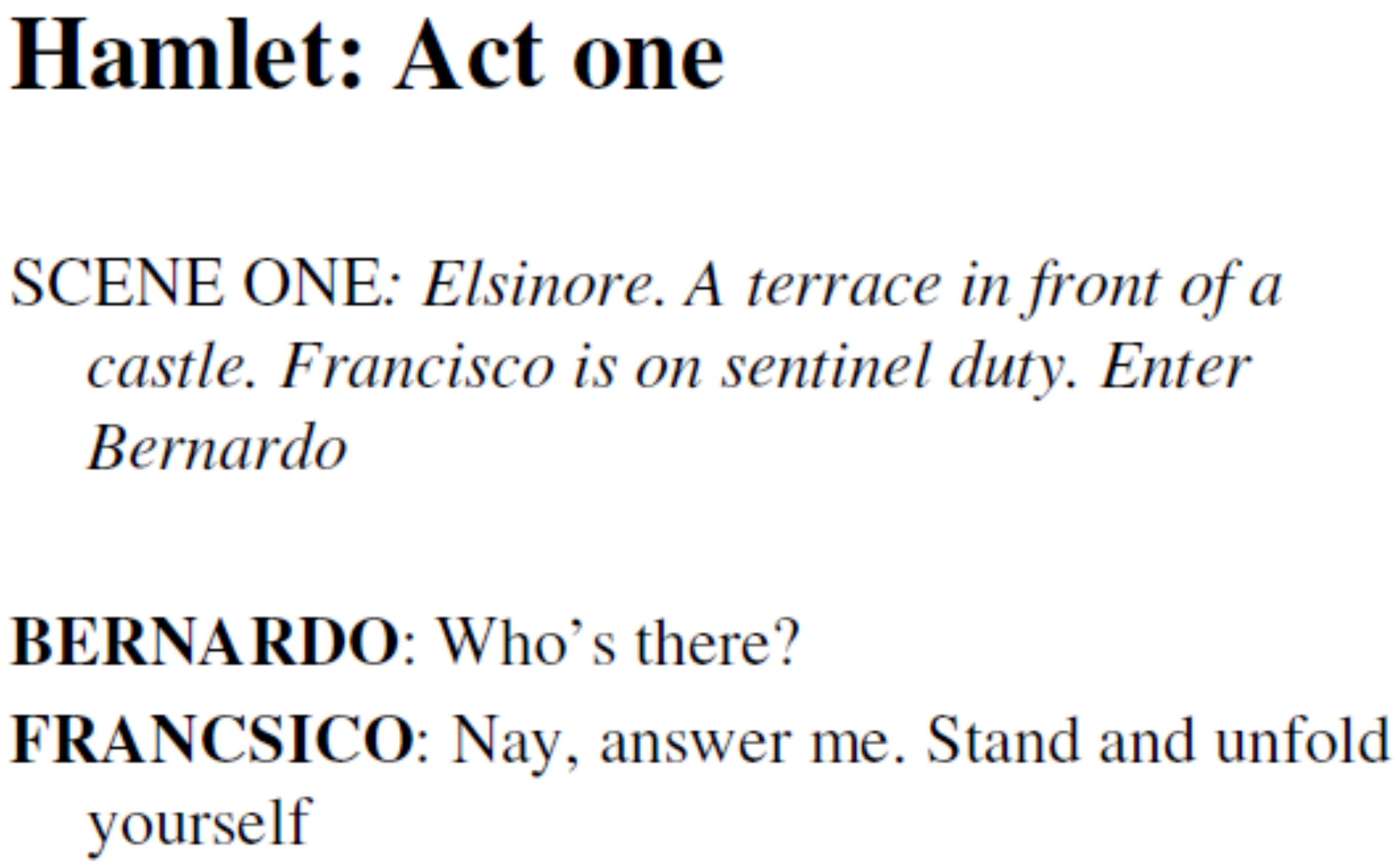
- HTML was designed for pure presentation
- **HTML is concerned with formatting not meaning**
  - it doesn’t matter what it is about, HTML will format it
- HTML is not extensible
  - can’t be modified to meet specific domain knowledge
  - browsers have developed their own tags (<*bgsound*>, <*layer*>)
- HTML can be inconsistently applied
  - almost everything is rendered somehow
    - e.g. <b>is this acceptable?</i>

XML: eXtensible Markup Language

- Developed in the mid 90’s by committee
- Derived from SGML
- A ‘meta’ mark-up language
  - used to create other mark up languages
  - Extensible, user defined tags
- Separates style and content
- Supports internationalisation (Unicode)
- Rigorous adherence to rules
- Device and system independent
- Applications may generate and process XML
- Enables data exchange between different platforms
- Facilitates better encoding of semantics
- **Both humans and machines can read it…**
- *“Transcends politics through sheer usefulness…”*
  - *“Intro to XML”*, Tim Anderson, 2004
  - http://www.itwriting.com/xmlintro.php

Hamlet in plain-text

MELBOURNE



Hamlet: Act one

SCENE ONE: *Elsinore. A terrace in front of a castle. Francisco is on sentinel duty. Enter Bernardo*

BERNARDO: Who's there?

FRANCSICO: Nay, answer me. Stand and unfold yourself

Hamlet in HTML

```
<body>
        <h1> Act One </h1>

        <p>
                <i> Scene One: Elsinore. A terrace in front of a castle.
                Francisco is on sentinel duty. Enter Bernardo. </i>
        </p>
        <p>
                <b> BERNARDO: </b> Who's there?
        </p>
        <p>
                <b> FRANCISCO: </b> Nay, answer me: stand, and
                unfold yourself
        </p>
</body>
```

Hamlet in XML

```
<?xml version="1.0"?>
<act>
        <title> Act One </title>
        <scene>
                <title> Scene One </title>
                <location> Elsinore. A terrace in front of a castle. </location>
                <stagedir> Francisco is on sentinel duty. Enter Bernardo </stagedir>

                <speech>
                        <speaker> BERNARDO </speaker>
                        <line> Who's there? </line>
                </speech>
                <speech>
                        <speaker> FRANCISCO </speaker>
                        <line> Nay, answer me: stand, and unfold yourself. </line>
                </speech>
        </scene >
</act>
```

XML Syntax Rules

- xml files must begin with declaration
  - *<?xml version=“1.0”?>*
- xml files must have one single root element
  - *E.g. <act>...</act>*
- elements are built with tags, must be properly closed
  - opening *<firstname>* and closing *</firstname>*
  - empty *<br />*
- an element may have one or more attributes, attributes must be in quotes
  - *<person title="Sir”>Richard</person>*
  - *<person title="Mr" sex="Male”>James</person>*

XML Syntax Rules

- xml code is case sensitive
  - <title> is not the same as as <Title>
- elements must be appropriately nested
  - <author><firstname>James</firstname></author>
  - <author><firstname>James</author> </firstname>
    - Wrong…
- comments

  <!-- comments do not affect the document,

  it's not part of the data that you want to represent -->

XML Syntax Rules

- some characters have special meaning
  - < and & are strictly illegal inside an element
    - <text>all books & videos are now < AUD 10</text>
      - Wrong…
    - <text>all books &amp; videos are now &lt; AUD 10</text>

- CDATA (character data) section may be used inside XML element to include large blocks of text, which may contain these special characters such as &, >
  - <![CDATA [… … …]]>
  - <![CDATA [all books & videos are now < AUD 10]]>

XML Syntax Rules (example)

```
<?xml version="1.0"?>
<catalog>
    – <book isbn="1-23456-789-0">
        • <title>Beyond the Clouds</title>
        • <author>
            – <firstname>Rebecca</firstname>
            – <surname>Skye</surname>
            – <picture source="rebecca.jpg" />
        • </author>
    – </book>
    – <book isbn="0-98765-432-1">
    – <title>The Final Straw</title>
    – <author>
        • <firstname>James</firstname>
        • <surname>Last</surname>
        • <picture source="james.jpg" />
    – </author>
    – </book>
</catalog>
```

- Declaration
- One root element
- Attributes in quotes
- Empty tag
- Opening/closing tags
- Tags correctly nested

- **“WELL FORMED”**

Exercise

- Given the following data: Yellow Balloon, $99.99
  - i) What are three possible XML encodings of the balloon ?
  - ii) What are some of the circumstances in which one encoding might be better than the others ?

XML Namespaces

- Here is some information about an HTML table

```
<table>
<tr>
<td>Dogs</td> <td>Cats</td>
</tr>
</table>
```

Here is some information about furniture

```
<table>
<name>Australian Coffee Table</name>
<width>90</width>
<length>149</length>
</table>
```

What happens if we add these together in the one document?

XML Namespaces [example adapted from w3schools.com]

- Namespace Declarations are used to qualify names with universal resource identifiers (URI's). A URI uniquely identifies a resource on the Web. The name consists of two parts
  - *namespace:local-name*
- This is achieved indirectly by using namespace declarations and associated user-specified prefixes

```
<... xmlns:tabular-info="http://www.tabularinfo.com">
<tabular-info:table>
<tr>
<td>Dogs</td> <td>Cats</td>
 </tr>
</tabular-info:table>
```

- xmlns:tabular-info attribute declares namespace with prefix tabularinfo
- URI doesn't have to refer to a real Web resource

Namespace Scope

- The scope of a namespace declaration is
  - the element that contains the namespace declaration
  - all its descendants (i.e. nested within the element)
  - The declaration may be overriden by further nested namespace declarations
- Namespaces can be used to to describe both elements and attributes. Elements/attributes without a namespace prefix are defined a default namespace.

Namespace example

```
<collection xmlns="http://www.tabularinfo.com" xmlns:furniture="http://
www.furniture.com">

<table>
<tr><td>Dogs</td> <td>Cats</td> </tr>
</table>

<furniture:table>
<furniture:name>Australian Coffee Table</furniture:name>
<furniture:width>90</furniture:width>
<furniture: length>149</furniture:length>
</furniture:table>
 </collection>
```

-collection, first table, td and tr use the default (tabularinfo namespace)

-second table, name, width and length use the furniture namespace

Namespace exercise [adapted from http://saxadapter.sourceforge.net/XMLNamespaceTutorial.html]

```
<a:Envelope
                xmlns="http://default/"
                xmlns:a="http://urla"
                xmlns:b="http://urlb"
                xmlns:c="http://urlc"
                a:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <a:Header xmlns="" xmlns:b="http://alturlb">
      <b:type>HelloWorld</b:type>
      <c:to xmlns:c=http://alturlc>John Doe</c:to>
      <from fromType="name">Jane Seymour</from>
  </a:Header>
<a:Body>
    <text xmlns="http://newdefault">Hello</text>
    <b:mood>Tired</b:mood>
    <c:day>Thursday</c:day>
    <month>March</month>
</a:Body> <
/a:Envelope>
```

- For each of the following, give its namespace URI: i) a:Envelope ii) a:Header iii) a:encodingStyle iv) b:type v) month vi) from vii) fromType viii) a:Body ix) text x) b:mood

XML Schema

- We need to ensure the integrity of our data – define its expected structure and content
  - “A book element must have as children, a title, an ISBN and at least one author.”
  - “A title is a sequence of characters”, “An ISBN is …”
- The format of the data can be specified by a *schema* and a document validated using schema checking software
  - Browsers use the HTML 5 Schema (see <!DOCTYPE html> at the start of an HTML document)
  - Schemas also used for other data formats
    - XML Schema (a W3C standard)
      - Large and complex, uses regular expression like rules
      - We will not look at the details in this subject

Validation

- An XML instance file is valid if it is consistent with a particular Schema

- Validation Tools
  - local XML editors (XMLWriter, Editix, Liquid XML …)
  - online validators: http://validator.w3.org/
  - lxml (python library)

- **Note: an XML file can be well-formed and NOT valid**

Using HTML/XML (Python)

- For HTML scraping, the Beautiful Soup library is good

- For XML, a good library is
  - http://lxml.de/
- Import the XML file into your program as a tree structure:

```
import xml.etree.ElementTree as ET
tree = ET.parse(‘yourfile.xml')
root = tree.getroot()
```

- Then loop through root with the various methods available:

```
for child in root:
  print child.tag, child.attrib
```

Using XML (Programming interfaces)

- Document Object Model (DOM)
  - Most useful way of parsing XML
  - Parsing calls load the document into a tree structure with different nodes that can be navigated by the program
- Simple API for XML (SAX)
  - Stream-based way of reading XML
  - Fast and efficient if you don’t need random-access

Further reading

- Further reading
  - Relational databases
    - Pages 403-409 of http://i.stanford.edu/~ullman/focs/ch08.pdf
  - XML
    - http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SG.html