

Perspectives on Data

Tradeoffs and considerations in data modeling



CHIDATA

Types of Data

Tabular Data: Tables, Spreadsheets, DataFrames

Semi-Structured Data: JSON, XML

Unstructured Data: Text, Video, Images, Raw Sensor Data

Tabular Data

A rectangular data structure where rows correspond to observations and each column corresponds to properties (attributes) of the observation.

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Tabular Data

Characteristic 1. **Named Attributes**

- The columns names define all of the relevant attributes for each observation.
- Values are accessed by name.
- Every row has the same set of attributes (if data are missing, value is explicitly NULL)

Customers					
	CustomerId	FirstName	LastName	DateCreated	Cl
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	

Tabular Data

Characteristic 2. **Atomic Values**

- Each cell (a row, column pair) is generally considered to be an atomic value---i.e., it is not readily divisible--such as, an integer, a date, a string.
- Each column has a defined data type consistent across all rows.

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Schema

An abstract definition of a tabular dataset consisting of a set of named attributes and their corresponding data types.

Car(Make: String, Model: String, Year: Int)


Person(First: String, Last: String, SSN: String)

Grade(SID: String, Class: String, Score: Float)

Data Model

Key advantage is that you can separate how users program against the data from how the data are actually stored.

```
df['name'] = df['name'].lower()
```



Has to be a string

Logical data model: describes the semantics of the data, as represented by the particular data analysis technology.

Physical data model: Describes the physical means by which data are stored in terms of bits and organization.

Semi-Structured Data

A data structure where each row has a variable set of attributes the values are possibly not atomic.

```
{'name': 'Car1', 'features': ['BackupCamera', 'CruiseControl']}  
{'name': 'Car2', 'features': ['BackupCamera', 'ParkingAssist',  
'CruiseControl']}
```

```
- <Parts>  
- <Part>  
  <Id>4478</Id>  
  <Part_Name>1000 Ohm Resistor</Part_Name>  
  <Total_Available>25000</Total_Available>  
  <Price>0.01</Price>  
</Part>  
- <Part>  
  <Id>3328</Id>  
  <Part_Name>15000 Ohm Resistor</Part_Name>  
  <Total_Available>75000</Total_Available>  
  <Price>0.02</Price>  
</Part>  
- <Part>  
  <Id>4725</Id>  
  <Part_Name>555 Timer IC</Part_Name>  
  <Total_Available>1500</Total_Available>  
  <Price>0.25</Price>  
</Part>  
</Parts>
```


Semi-Structured Data

Characteristic I. **Named Attributes**

- Attributes are still named!
- But each row may have a different set of named attributes.

```
- <Parts>
- <Part>
  <Id>4478</Id>
  <Part_Name>1000 Ohm Resistor</Part_Name>
  <Total_Available>25000</Total_Available>
  <Price>0.01</Price>
</Part>
- <Part>
  <Id>3328</Id>
  <Part_Name>15000 Ohm Resistor</Part_Name>
  <Total_Available>75000</Total_Available>
  <Price>0.02</Price>
</Part>
- <Part>
  <Id>4725</Id>
  <Part_Name>555 Timer IC</Part_Name>
  <Total_Available>1500</Total_Available>
  <Price>0.25</Price>
</Part>
</Parts>
```

Semi-Structured Data

Characteristic 2. **Nested Values**

- Values are not atomic and can be nested data structures.
- Data type is generally inferred on the fly.

```
- <Parts>
- <Part>
  <Id>4478</Id>
  <Part_Name>1000 Ohm Resistor</Part_Name>
  <Total_Available>25000</Total_Available>
  <Price>0.01</Price>
</Part>
- <Part>
  <Id>3328</Id>
  <Part_Name>15000 Ohm Resistor</Part_Name>
  <Total_Available>75000</Total_Available>
  <Price>0.02</Price>
</Part>
- <Part>
  <Id>4725</Id>
  <Part_Name>555 Timer IC</Part_Name>
  <Total_Available>1500</Total_Available>
  <Price>0.25</Price>
</Part>
</Parts>
```









Semi-Structured Schema

Can think of each row as defining its own schema and that definition is stored with each row.

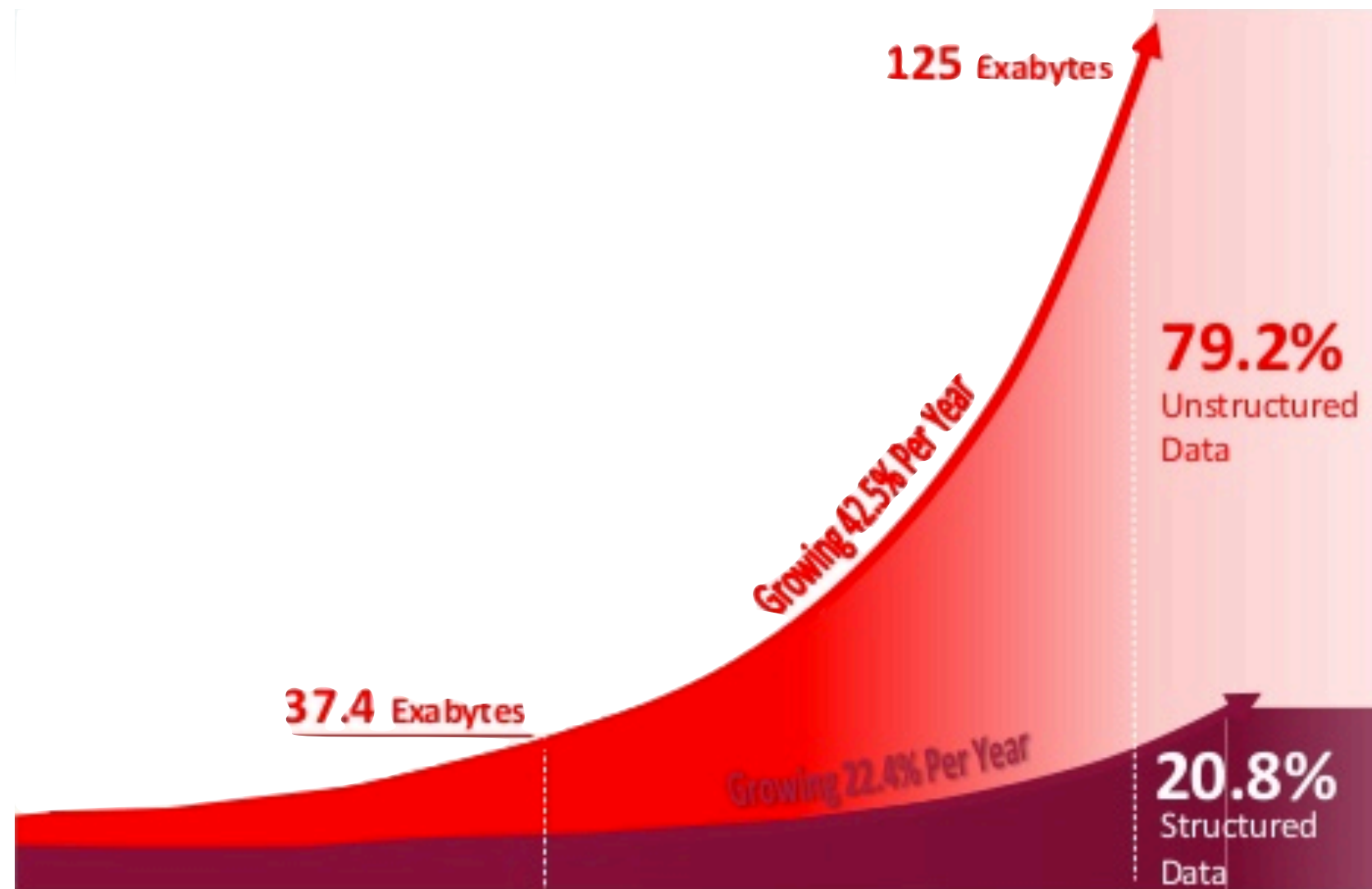
```
- <Parts>
  - <Part>
    <Id>4478</Id>
    <Part_Name>1000 Ohm Resistor</Part_Name>
    <Total_Available>25000</Total_Available>
    <Price>0.01</Price>
  </Part>
  - <Part>
    <Id>3328</Id>
    <Part_Name>15000 Ohm Resistor</Part_Name>
    <Total_Available>75000</Total_Available>
    <Price>0.02</Price>
  </Part>
  - <Part>
    <Id>4725</Id>
    <Part_Name>555 Timer IC</Part_Name>
    <Total_Available>1500</Total_Available>
    <Price>0.25</Price>
  </Part>
</Parts>
```

Unstructured Data

Everything else!

 Text files and documents	 Server, website and application logs	 Sensor data	 Images
 Video files	 Audio files	 Emails	 Social media data

Unstructured Data



Tradeoffs

Most organizations have a mix of different types of data!

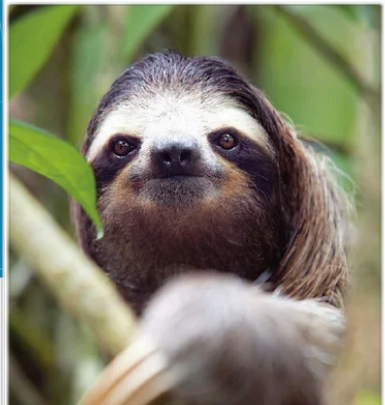
Tabular Data (structured): Tables, Spreadsheets, DataFrames

- + Simplified programming model (Pandas, SQL, Excel)
- + Efficient and scalable
- Inflexible (need to define a schema up front)
- Hard to migrate data into other systems

Semi-Structured Data: JSON, XML


- + Very flexible (on-the-fly schema design)
- + Portable (readable by all major programming languages)
- Hard to debug code with complex failure modes
- Extra data and extra code

Example

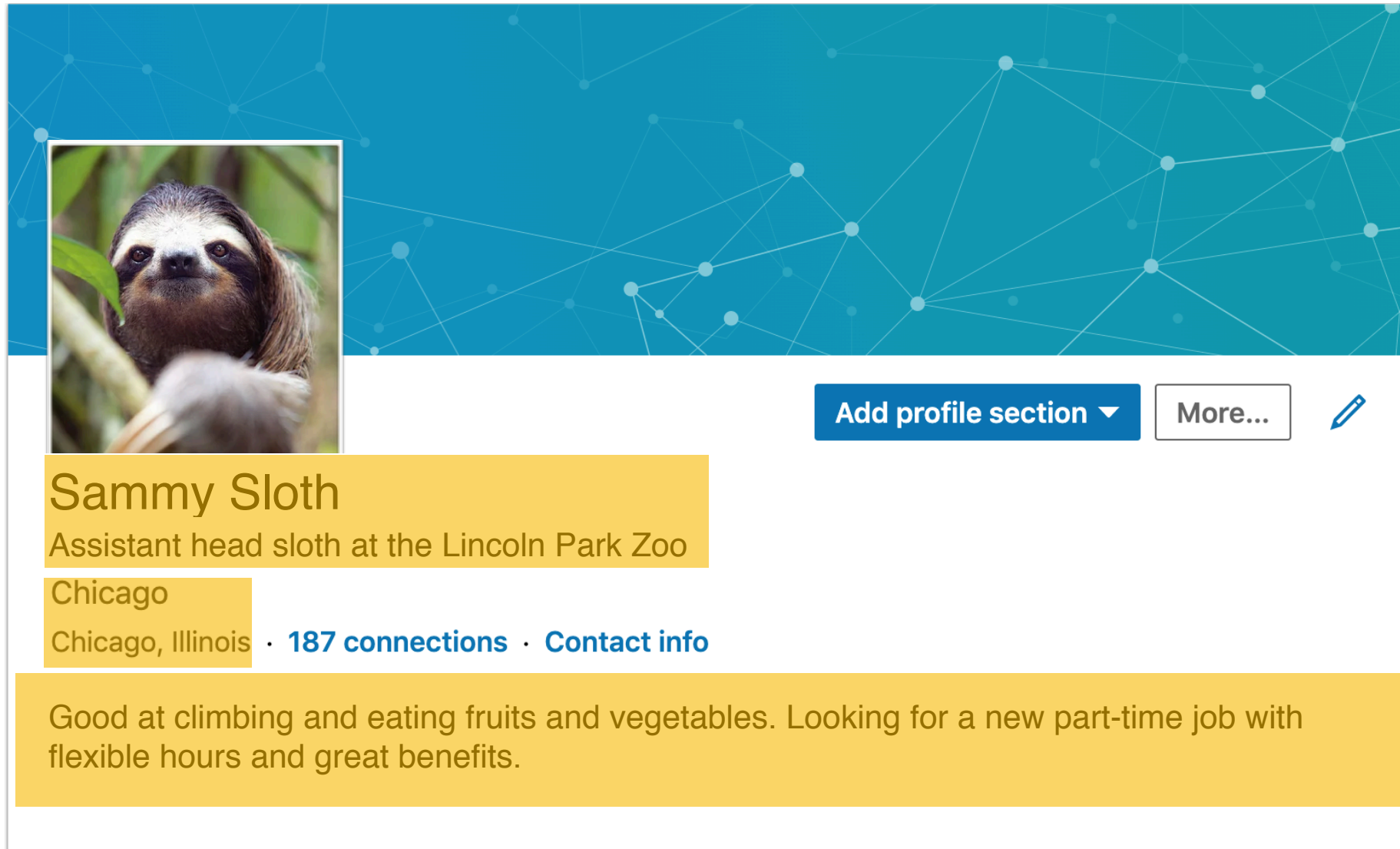


Sammy Sloth
Assistant head sloth at the Lincoln Park Zoo
Chicago
Chicago, Illinois · [187 connections](#) · [Contact info](#)

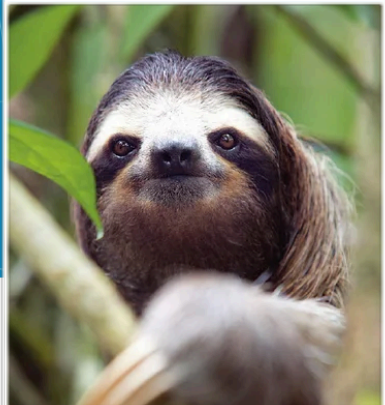
Good at climbing and eating fruits and vegetables. Looking for a new part-time job with flexible hours and great benefits.


[Add profile section ▼](#) [More...](#) 

Example



A profile card for 'Sammy Sloth' set against a blue background with a white network graph pattern. The card features a photo of a sloth on the left. To the right of the photo are two buttons: 'Add profile section' with a dropdown arrow and 'More...' with a pencil icon. Below the photo, the name 'Sammy Sloth' is displayed in a large font, followed by the title 'Assistant head sloth at the Lincoln Park Zoo'. Below this, the location 'Chicago' is shown, followed by 'Chicago, Illinois' and links for '187 connections' and 'Contact info'. At the bottom, a text box describes the sloth's skills and job preferences.



Add profile section ▼ **More...** 

Sammy Sloth
Assistant head sloth at the Lincoln Park Zoo

Chicago
Chicago, Illinois · [187 connections](#) · [Contact info](#)

Good at climbing and eating fruits and vegetables. Looking for a new part-time job with flexible hours and great benefits.

Semi-Structured Data

Example

```
{'name': 'Sammy Sloth',  
  'location_city': 'Chicago',  
  'location_state': 'Illinois',  
  'position': 'Assistant head sloth at the Lincoln Park Zoo',  
  'looking_for': 'Good at climbing and eating fruits and  
vegetables. Looking for a new part-time job with flexible  
hours and great benefits.'  
}
```

- + Flexible
- + Redesign of the website doesn't need a data change
- + Portable

Example

Count the number of climbers per city

```
cities = {}

for a in Animals:
    obj = json.loads(a)

    city = obj['location_city']
    is_climber = obj['looking_for'].contains('climber')

    if city not in cities:
        cities[city] = 0

    cities[city] += (is_climber)
```

Example

Count the number of climbers per **state**

```
states = {}

for a in Animals:
    obj = json.loads(a)

    state = obj['location_state']
    is_climber = obj['looking_for'].contains('climber')

    if state not in states:
        states[state] = 0

    states[state] += (is_climber)
```

Example

Extracting structured tables from semi-structured or unstructured data simplifies analysis later on

```
{'name': 'Sammy Sloth',  
  'location_city': 'Chicago',  
  'location_state': 'Illinois',  
  'position': 'Assistant head sloth at the Lincoln Park Zoo',  
  'looking_for': 'Good at climbing and eating fruits and  
vegetables. Looking for a new part-time job with flexible hours  
and great benefits.'  
}
```



Talent(Name: String, City: String, State:String, Climbing: Boolean)

Example

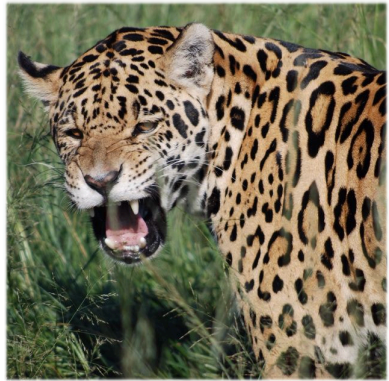
Count the number of climbers per city

Talent(Name: String, City: String, State:String, Climbing: Boolean)

```
df.groupby('city').sum('climbing')  
df.groupby('state').sum('climbing')
```


+ Cost of structuring data amortizes over all the analyses you want to do

Example



Jaime Jaguar
Jaguar at the Lincoln Park Zoo
Chicago
Chicago, Illinois · [187 connections](#) · [Contact info](#)

Just a good ole' big cat who hates climbers and looking for a good time.

[Add profile section](#) ▾ [More...](#) 

Example

Format conversions often lose or misrepresent data

```
'looking_for': 'Just a good ole' big cat who  
hates climbers and looking for a good time.'
```



Talent(Name: String, City: String, State:String, **Climbing: Boolean**)

Paradox of Structured Data

Always certain, seldom right

- Query languages (like SQL) are built on the principles of formal logic
- A *database* holds facts and allows users to make principled inferences from these facts

Paradox of Structured Data

- Mary is a citizen of France
- Jenny is a citizen of France
- All French citizens wear hats

People

Name	Citizenship
Mary	France
Jenny	France

Countries

Citizenship	Wears
France	hats

What is Mary's citizenship?



```
SELECT Citizenship
FROM People
WHERE Name='Mary';
```

```
people[people['name']='Mary']
```

Do both Mary and Jenny wear hats?



```
SELECT Name, Wears
FROM People, Countries
WHERE People.Citizenship =
       Countries.Citizenship;
```

```
people.merge(countries)
```

Paradox of Structured Data

Does Peter Wear A Hat? →

```
SELECT Name, Wears  
FROM People, Countries  
WHERE People.Citizenship =  
       Countries.Citizenship AND  
       People = 'Peter';
```

```
peter = people[people['name']='Peter']  
peter.merge(countries)
```

Closed World Assumption: Everything not currently known to be true, is false

Example

Up-to the developer to understand and account for uncertainty

```
'looking_for': 'Just a good ole' big cat who  
hates climbers and looking for a good time.'
```



Talent(Name: String, City: String, State:String, **Climbing: Boolean**)

Integrity Constraints

- Schemas not only should include attribute types but also constraints on the values they can take.

Person(First: String, Last: String, SSN: String)

Integrity Constraints

- Schemas not only should include attribute types but also constraints on the values they can take.

Person(First: String, Last: String, SSN: String)

Syntactic Constraints

- First name: must be alphabetical (upto apostrophes and dashes)
- Last name: must be alphabetical (upto apostrophes and dashes)
- SSN: must follow a format XXX-XX-XXXX

Semantic Constrains

- SSNs must be unique

Integrity Constraints

Syntactic Constraints: Formatting of a single cell of data

Semantic Constrains: Relationships between data

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Integrity Constraints

Syntactic Constraints: Formatting of a single cell of data

No cell can be empty

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Integrity Constraints

Semantic Constraints: Relationships between data

All firstname, lastname pairs are unique

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Tradeoffs

Integrity constraints are really only useful over tabular data!

Tabular Data (structured): Tables, Spreadsheets, DataFrames

- + Simplified programming model (Pandas, SQL, Excel)
- + Efficient and scalable
- + **Verifiable/Testable**
- Inflexible (need to define a schema up front)
- Hard to migrate data into other systems

Semi-Structured Data: JSON, XML

- + Very flexible (on-the-fly schema design)
- + Portable (readable by all major programming languages)
- Hard to debug code with complex failure modes
- Extra data and extra code

Uniqueness is Important!

Semantic Constraints: Relationships between data

All firstname, lastname pairs are **unique**

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Uniqueness is Important!

Keys: A set of attributes such that no two records have the same values.

{CustomerID}, {FirstName, LastName} Is this it..?

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Uniqueness is Important!

Keys: A key + any other attribute is another key

{FirstName, LastName, DateCreated}

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Uniqueness is Important!

Candidate Key: A key that doesn't contain any other keys

{CustomerID}, {FirstName, LastName}, {DateCreated}

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Uniqueness is Important!

Any one of these column sets can uniquely “index” the records.

{CustomerID}, {FirstName, LastName}, {DateCreated}

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Uniqueness is Important!

Why might the creation date be a bad key? First/Last Name?

{CustomerID}, {FirstName, LastName}, **{DateCreated}**

Customers					
	CustomerId	FirstName	LastName	DateCreated	Client
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

“Primary” Keys

- Schemas not only should include attribute types but also constraints on the values they can take.

Person(First: String, Last: String, SSN: String)

- Schemas should also describe what is the primary key for the table (uniquely identifies records)

Person(First: String, Last: String, **SSN**: String)