

Database Models

“Introduction to Database Systems”

Fall 2020

Parke Godfrey

Versions

initial: *2020 January 7*

last modified: *2020 September 15*

Acknowledgments

Thanks to

- *Jeffrey D. Ullman*
for initial slidedeck
- *Jarek Szlichta*
for the slidedeck with significant refinements on which
this is derived
- *Wenxiao Fu*
for refinements

Printable version of talk

- Follow this link:
[Data Models \[to pdf\]](#).
- Then *print* to get a PDF.

*(This only works correctly in **Chrome** or **Chromium**.)*

Context

Information is data in context.

Parke's database

I have a database! Oh? What's it about?

Well, here it is:

[42]

That's it? Yes.

42. Yes.

But 42 what?! Kangaroos? Number of cousins? Just the number itself?!

Needs *context*

A *datum* — a piece of *data* — by itself means nothing.

A datum with *context* means something.

- The context might be a *question*, and the datum an *answer* to the question.

E.g., *How many cats does Parke have?* 42.

- The context might be provided by *where* that data is “within” a *schema* (wrt a database model).

Need to *model* context

- **data(base) model**

- This provides a logical framework for *how*

- data can be organized, and
 - can be inter-related among themselves.

- In other words, it provides a model for context.

Schema

- **schema**

- A schema is an instance of the data model that specifies how

- data *is* organized, and
 - is inter-related.

- **query language**

- This provides a way to retrieve / query data by context.

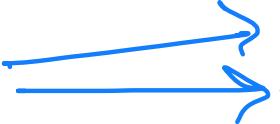
- And it may provide powerful means to query for data that are inter-related in ways we had not anticipated beforehand.

Detour!

Covered in [Preliminary Video, *What is a Data Model?*](#)

Can you think of *data models* you have encountered?

- Within *programming languages*, we use *data structures*.
 - *data structures* are precursors to this idea.
 - **map** in **Java** (& other languages), for instance, describes the “data model” of **key-value**.

Key-Value  *data model*
type of database server

Indeed, **key-value** is a *data model*, and a type of *database server*!

- One can *insert* a new *key-value* pair, *delete* a pair by *key*, overwrite the *value* associated with a *key*, and *read* the *value* paired with a *key*.
- Pretty much the API of **map**.
- But via a highly performant server.

A really simple data model, but quite useful all the same.

HTML?!

Take a look at the *source* of this “slide”, if you are viewing this (non-pdf) reveal.js slidedeck on a web browser.

- You likely recognize it. It's just HTML!
- Is HTML a *data model*? One could argue... It is a way of organizing the information of the “page”.
- And **dom** does stand for *document object model*!
- We usually don't think of HTML as a data model, though...

XML!

XML, HTML's close cousin, *is* a data model.

- In XML, we can introduce any *tag* names we want.
- We organize our information in a *tree* structure, by nested tags.
- XML is a type of “semi-structured” data model.

Take a look at the XML “document” [mondial-2015.xml](#). (Caution, it is reasonably large.)

It is a “database” of information about *countries* in the world.

How to “use” our XML information?

A *tree-structured* data model like XML is fairly intuitive.
(Although, XML's *syntax* is frightening!)

But how do we *manage* our data / information in an XML document / database?

How do we *query* for information from an XML database?

Example Question

Say I wanted to *pull* the following information:

List each *religion*; order by their names. Within each religion group, list the *countries* for which the leading religion by *percentage* is that religion. Order the countries then by percentage, country name.

How do I do it?

XQuery

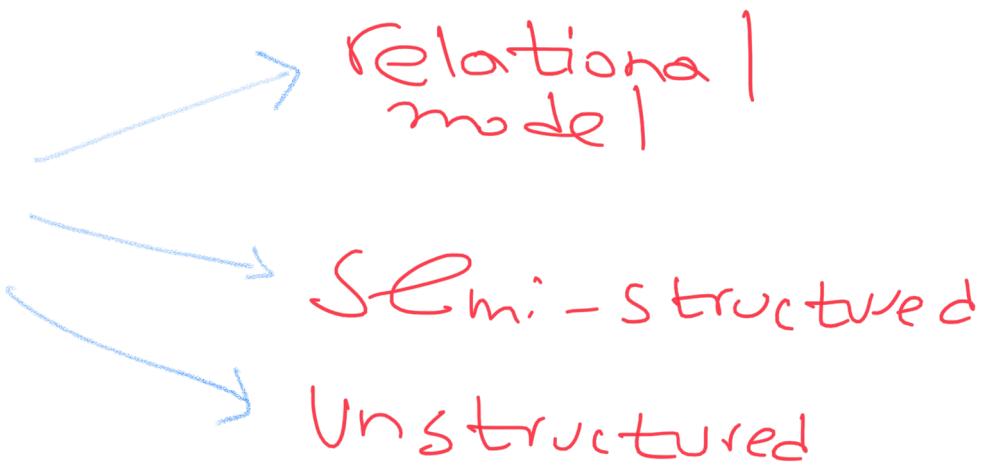
There is a standard query language named **XQuery** for XML.

- So I could write an **XQuery** query — `religions.xq` — to “answer” the question when evaluated against `mondial-2015.xml`.
- When `religions.xq` is evaluated against `mondial-2015.xml`, the *answer* produced is an XML document: `religions.xml`.

Outline

- **data models**
 - a. the *relational model*
 - b. a *semi-structured model*
 - e.g., XML
- **schema** (*wrt* the data model)
 - a. *relational schema*
 - b. e.g., DTD (*Document Type Definition*)
- **query languages** / "programming" languages
 - a. e.g., SQL
 - b. e.g., xQuery (& XPath)

Data Models



What is a data(base) model?

It is a mathematical representation for data.

1. relational model:

- data organized in *tables*
- the tables can be *related* in specific ways
- the data can be *constrained* in specific ways

2. semi-structured models:

- data organized as labeled *trees*, or
- as labeled *graphs*

Logical vs physical representation

- Data models & schema are about how the data is *logically* organized.
- How a *database system* – a *database management system* – for a given database model is
 - implemented, and
 - how the data is *physically* organizedare different matters.

This “separation of concerns” is called *data independence*.

The Relational Model

A relation is just a table

Beers		→ relation name
name	manf	→ attributes
Winterbrew	Pete's	
Bud Lite	Anheuser-busch	↑ → tuples

- **relation name:** name of the table. E.g., **Beers**
- **attributes:** the column headers. E.g., **name** & **manf**
- **tuples:** the rows.
- **cells:** individual values (given attribute, given tuple).

That's it?!

Well, yes. (Almost.)

- Kind of like a spreadsheet, eh?
But without all the cool functions!
- This extreme *simplicity* of representation will let us
 - design a powerful, *declarative* query language,
 - support the features we want — e.g., *integrity* & *transactions*, — and
 - build quite efficient database systems.
- Even while really simple, lots of data is *tabular* — that is, can be fit into tables — in nature.

Relational databases

- *Used* to be about “boring” stuff. E.g.,
 - employee records, bank records
- Today, the field covers all the largest sources of data, spanning many new ideas. E.g.,
 - web search
 - data mining
 - social networking
 - scientific & medical databases
 - integrating information

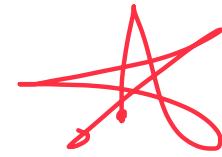
Databases everywhere

Databases are hidden behind almost everything you do on the Web or in an app.

→ has SQLite embedded in them.

- web searches (Google, Bing)
- searching at Amazon, eBay
- e-commerce
 - buying concert tickets on-line
- scrolling your favourite social-network feed
 - Instagram, Snapchat, Twitter, Facebook

Database *systems* everywhere



The functionality of database systems *solve* many complications for complex applications behind the scenes “for free”.

- Supports complex information processing.
(the query language)
- Juggles many activities simultaneously.
(concurrency)
- Ensures correctness of the results of the activities.
(transaction management)
 - E.g., two withdrawals from the same account must *each* debit the account.

Relational schema terminology

- **relation schema:** relation name and attribute list (*set*).
 - And optionally, the *types* of the attributes.
 - E.g., Beers(name, manf) or
Beers(name: string, manf: string)
- **database:** collection of relations (“tables”).
- **database schema:** set of all relation schemas in the database.



Our running-example schema

- **Beers(name, manf)**
 - **Pubs(name, addr, license)**
 - **Drinkers(name, *addr*, *phone*)**
 - **Likes(drinker, beer)**
 - **Sells(pub, beer, price)**
 - **Frequents(drinker, pub)**
-
- **bolded** attribute means it is (part of) the key

Database schemas in SQL

- SQL is primarily a *query language* for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

Creating / declaring a relation / table

The simplest form is

```
create table <name> (
    <list of elements>
);
```

To *delete* a relation:

```
DROP TABLE <name>;
```

Elements of table declarations

The most common types are

- int or integer (synonyms)
- real or float (synonyms)
- char(n): fixed-length string of n characters
- varchar(n): variable-length string of up to n characters

Example: create table

```
create table Sells (
    pub      char(20),
    beer     varchar(20),
    price    real
);
```

SQL values

Integers and reals are represented as you would expect.

Strings are too, except they require single quotes.

- Two single quotes = real quote, e.g., 'Joe's Bar'.

Any value can be *null*.

Dates and times

date and time are types in SQL. The form of a date value is:

- 'yyyy-mm-dd'
- E.g., '2007-09-30'
for September 30, 2007.

Times as values

The form of a time value is

- 'hh:mm:ss'

hour / min / second

with an optional decimal point and fractions of a second following.

- E.g., '15:30:02.5'

meaning two and a half seconds after 3:30pm.

Declaring keys

An attribute, or list (set) of attributes, may be declared as primary key or unique.

This says that no two tuples in the relation (table) may agree on *all* the attributes's values in the key's list.

There are a few distinctions to be mentioned later.

Declaring single-attribute keys

Place primary key or unique after the type in the declaration of the attribute.

E.g.,

```
create table Beers (
    name      CHAR(20) UNIQUE,
    manf      CHAR(20)
);
```

Primary key → Unique identifier(attribute)

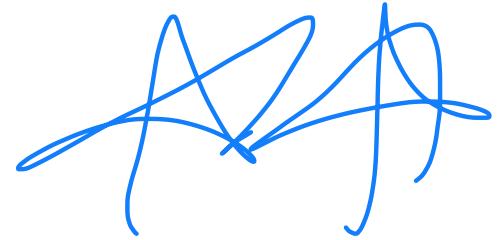
Declaring multi-attribute keys

- A key declaration can also be another element in the list of elements of a create table statement.
- This form is necessary if the key consists of more than one attribute.
- But this form may be used too for one-attribute keys.

Example: multi-attribute key

The *pub* and *beer* together are the key for **Sells**.

```
create table Sells (
    pub      char(20),
    beer     varchar(20),
    price    real,
    primary key (pub, beer)
);
```



Primary key vs Unique

1. There can be only one primary key for a relation, but any number of unique declarations.
2. No value of an attribute in the primary key can ever be null in any tuple. *(Primary key Can't be null)*

But values of attributes declared in unique may have null's! And there may be several tuples with null.

(Unique values Can be null)

AKP

Caveat: null is a messy concept

- *DB2* (IBM): any attribute used in a unique declaration must also be declared not null.
- *SQL Server* (Microsoft): at most *one* tuple may appear in the table with a given null pattern wrt s unique declaration.
- *PostgreSQL*: there may be any number of tuples appearing in the table with null values for attributes participating in a unique declaration.

Break-out Exercise

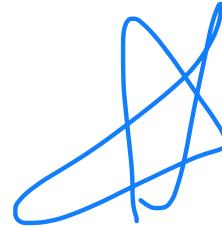
consider Wikipedia...

- Is Wikipedia a *database*?
 - If so, what is the *data model* that it likely uses?
 - Would it be *relational*?
-

Discuss in your break-out group

- whether thinking about Wikipedia as a *database* makes sense, and, if so,
 - what would be a good data model for it be?
-

Feel free to be creative!



The Semantic Web

The WWW can be thought of as a *graph data model*:

- a *web page* is a node; and
 - a *page* has *edges* (links) pointing to other pages.
-

Sir Tim Berners-Lee has lead an effort to build the *Semantic Web*.

- The links can be semantically *labelled*, so that we know *how* one page is related to the other via a link.

Knowledge Graphs

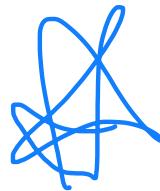
There is lots of current effort to define and build very large *knowledge graphs*. This takes a *graph data model* at its foundation.

Example

“The Google Knowledge Graph is a knowledge base used by Google and its services to enhance its search engine's results with information gathered from a variety of sources. The information is presented to users in an infobox next to the search results.”

- (*Google*) [Knowledge Graph](#) | [Wikipedia](#)

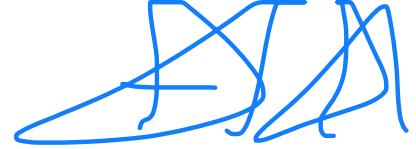
Wikidata



[Wikipedia](#) is being re-engineered as a *knowledge graph* so that applications can use it as an information source more easily.

The *behind-the-scenes* knowledge graph for [Wikipedia](#) is [Wikidata](#). E.g., Wikidata's “entry” on the *Google Knowledge Graph*: [Google Knowledge Graph - Wikidata](#).

[DBpedia](#) is another project that is a *graph-data* rendering of [Wikipedia](#)'s information.

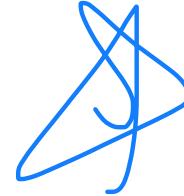


“Semi-structured” data models

Let's present another type of data model; we can base it on *trees* or on *graphs*.

Motivation

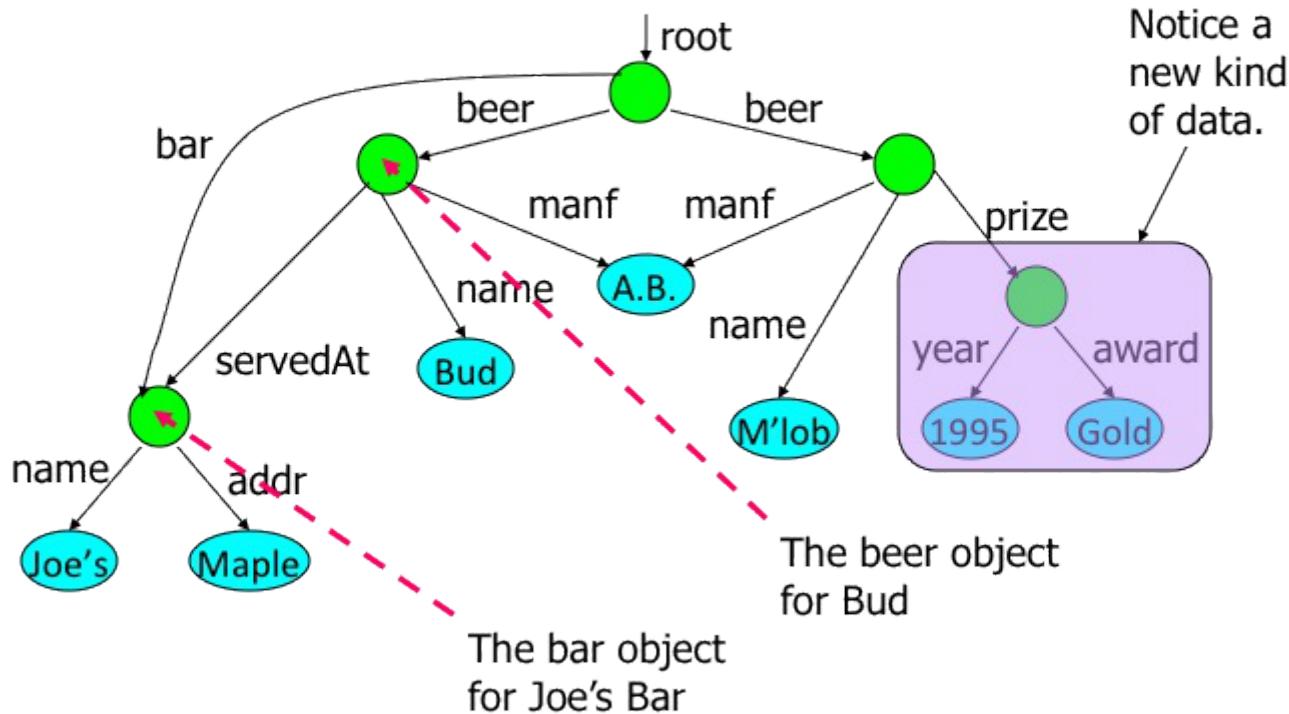
1. a more flexible (?) representation of data
2. able to go “schema-less”, or have as little or as much “schema” as needed
3. sharing of “documents” among systems and databases



Graphs of semi-structured data

- *Nodes* = objects.
- *Labels* on arcs (like attribute names).
- *Atomic values* at leaf nodes (nodes with no arcs out).
- Flexibility. No restriction on
 - labels out of a node *or*
 - number of successors with a given label

Example: Data Graph



XML

→ data model

XML = Extensible Markup Language.

While HTML uses tags for *formatting* (e.g., “italic”), XML uses tags for *semantics* (e.g., “this is an address”).

Key idea. Create tag sets for a domain (e.g., genomics), and translate all data into properly tagged XML documents.

XML documents

Start the document with a declaration, surrounded by <?xml ... ?>. E.g.,

```
<?xml version = "1.0" encoding = "utf-8" ?
```

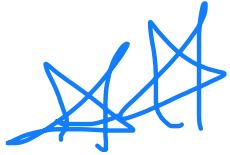
The balance of document is a *root* tag surrounding nested tags.

Tags

- Tags, as in HTML, are matched pairs, as `<foo>...`
`</foo>`.
- Optional single tag `<foo/>`, which is shorthand for
`<foo></foo>`.
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

Example: an XML document

```
<pubs>
  <pub>
    <name>Joe's Bar</name>
    <beer>
      <mark><name>Bud</name></mark>
      <price>2.50</price>
    </beer>
    <beer>
      <name>Molsons</name>
      <price>3.50</price>
    </beer>
  </pub>
  <pub>...</pub>
  ...
</pubs>
```



The difference between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
 - XML was designed to describe data, with focus on what data is
 - HTML was designed to display data, with focus on how data looks
- HTML is about displaying information, while XML is about carrying information.

HTML is an extension of XML

Attributes

Like HTML, the opening tag in XML can have attribute = value pairs.

Attributes also allow linking among elements (discussed later).

DTD (*Document Type Definition*)

A grammatical notation for describing *allowed* use of tags.

Definition form:

```
<!DOCTYPE <root tag> [
    <!ELEMENT <name>(<components>)>
    . . . more elements . . .
]>
```

This is a way to define a *schema* on an XML database / document.

DTD: Attributes

Opening tags in XML can have *attributes*.

In a DTD, `<!ATTLIST E . . . >` declares an attribute for element E, along with its datatype.

Example: DTD

```
<!DOCTYPE BARS [  
    <!ELEMENT BARS (BAR*)>  
    <!ELEMENT BAR (NAME, BEER+)>  
    <!ELEMENT NAME (#PCDATA)>  
    <!ELEMENT BEER (NAME, PRICE)>  
    <!ELEMENT PRICE (#PCDATA)>  
]>
```

Can we *query* XML?

Or JSON? Or HTML?

We would need a *query language* for it.

Well, is there one?

Yes. In fact, several. **XQuery** is a prominent *query language* for **XML**.

But more on that later. We have other things to learn first. Like querying relational (**SQL**).