An Introduction to Data Models and Data Retrieval

Scott Hoover

January 30, 2014

- Data Models

- Data Models
- The Concept of Relational Tables

- Data Models
- The Concept of Relational Tables
- Relational Tables as a Preferred Data Store

A data model is basically a way of defining the way data is stored. While physical hard drives actually store bytes of data, a person had to think about the best *way* to store that data—the structure.

There are a number of different ways to store data:

There are a number of different ways to store data:

- **Flat models** will typically store data in one or more arrays and look like a table. Data in flat files are usually delimited by a value (*e.g.*, comma- or tab-separated)

There are a number of different ways to store data:

- **Flat models** will typically store data in one or more arrays and look like a table. Data in flat files are usually delimited by a value (*e.g.*, comma- or tab-separated)
- **Hierarchical models** are tree-like systems of storing data, much like how files are stored in your computer. A file (child) exists in a folder (parent), which itself can exist and other folders or directories.

There are a number of different ways to store data:

- **Flat models** will typically store data in one or more arrays and look like a table. Data in flat files are usually delimited by a value (*e.g.*, comma- or tab-separated)
- **Hierarchical models** are tree-like systems of storing data, much like how files are stored in your computer. A file (child) exists in a folder (parent), which itself can exist and other folders or directories.
- **Key-value models** are simple, loose data structures. Keys are unique identifiers that allow one to refer to the associated value. Values can assume any value or document.

There are a number of different ways to store data:

- **Flat models** will typically store data in one or more arrays and look like a table. Data in flat files are usually delimited by a value (*e.g.*, comma- or tab-separated)
- **Hierarchical models** are tree-like systems of storing data, much like how files are stored in your computer. A file (child) exists in a folder (parent), which itself can exist and other folders or directories.
- **Key-value models** are simple, loose data structures. Keys are unique identifiers that allow one to refer to the associated value. Values can assume any value or document.
- **Relational models** store tables of rows and columns that relate to one another through keys. This will be our primary focus.

Here's an example of a flat model:

```
first_name  last_name   age street_address  city    zip
scott   hoover  30  "305 Downtown Street"   "Santa Cruz"    95060
mike    xu  26  "East Side Avenue"  "Capitola"  95010
nathaniel   pickens 25  "001 Downtown Street"   "Santa Cruz"    95060
```

Here's an example of a hierarchical model:

```xml
<person>

    <firstName>scott</firstName>

    <lastName>hoover</lastName>

    <age>30</age>

    <address>

        <streetAddress>"305 Downtown Street"</streetAddress>

        <city>"Santa Cruz"</city>

        <state>CA</state>

        <postalCode>95060</postalCode>

    </address>

</person>
```

Here's an example of a key-value model:

```
{
    "firstName": "Scott",
    "lastName": "Hoover",
    "age": 30,
    "address": {
        "streetAddress": "304 Downtown Street",
        "city": "Santa Cruz",
        "state": "CA",
        "postalCode": 95060
    }
}
```

Irrespective of the data model, the solution should address a few key points:

Irrespective of the data model, the solution should address a few key points:

- Is the solution flexible enough to use data in unforeseen ways?

Irrespective of the data model, the solution should address a few key points:

- Is the solution flexible enough to use data in unforeseen ways?
- Can multiple users/consumers of the data access and alter the data concurrently?

Irrespective of the data model, the solution should address a few key points:

- Is the solution flexible enough to use data in unforeseen ways?
- Can multiple users/consumers of the data access and alter the data concurrently?
- Is the solution practical enough to handle data that's too big to read into memory (*e.g.*, web data)?

In a relational table, each row represents a unique entity or event (*e.g.*, a Facebook user, a purchase from Amazon). For each row, there are columns (I like to refer to them as attributes) that tell us about the entity or event (*e.g.*, the date and time that the Facebook user created her account or the amount associated with the Amazon order).

Here's an example of a table...

Likes table

like_id	like_created_time	comment_text	comment_created_time	like_profile_id	comment_profile_id
1	2012-09-19 12:22:01	"This is my first Facebook comment"	2012-09-19 12:47:59	3	1
2	2012-11-01 08:01:04	"Mondays...Am I right?"	2012-11-01 07:59:42	4	2
3	2012-09-19 13:30:00	"My name is Jonas!"	2012-09-19 12:55:02	5	6
4	2012-09-19 12:47:59	"Driving home from LA today!"	2012-09-19 12:40:01	1	3
5	2012-09-19 13:24:11	"This is my first Facebook comment"	2012-09-19 12:47:59	4	1

In a relational model, primary and foreign keys link tables to one another.

like_id	like_created_time	comment_text	comment_created_time	like_profile_id	comment_profile_id
1	2012-09-19 12:22:01	"This is my first Facebook comment"	2012-09-19 12:47:59	3	1
2	2012-11-01 08:01:04	"Mondays...Am I right?"	2012-11-01 07:59:42	4	2
3	2012-09-19 13:30:00	"My name is Jonas!"	2012-09-19 12:55:02	5	6
4	2012-09-19 12:47:59	"Driving home from LA today!"	2012-09-19 12:40:01	1	3
5	2012-09-19 13:24:11	"This is my first Facebook comment"	2012-09-19 12:47:59	4	1

Here, *like_id* is the primary key and denotes the unique row or entry or, in our case, a "like." The foreign keys are *like_profile_id* and *comment_profile_id* which relate the like table to the profiles of the liking user and the commenting user, respectively.

Relational algebra is a way of describing how data relate to one another, particularly in the context of querying relational data.

However, relational algebra relies on formal logic and set theory and is, therefore, beyond the scope of this class. For our purposes, we will refer to a few key concepts from relational algebra: projection (`SELECT`), restriction (`WHERE`), and join (`JOIN`).

There are two main reasons relational tables are are often a preferred way to store data.

The first reason is that well-structured tables can be queried very quickly, which makes a serious difference when we're talking about tables with millions or billions of rows. This is because relational tables are structured to reduce redundant information (we'll see examples of this).

The second reason is that relational tables are typically general enough so that people can query data in all sorts of ways that others may not have thought up when the data was first being structured or stored.

Considering these two points above, we can revisit the Facebook "Likes" table and see if it can be better structured.

Likes table

like_id	like_created_time	comment_text	comment_created_time	like_profile_id	comment_profile_id
1	2012-09-19 12:22:01	"This is my first Facebook comment"	2012-09-19 12:47:59	3	1
2	2012-11-01 08:01:04	"Mondays...Am I right?"	2012-11-01 07:59:42	4	2
3	2012-09-19 13:30:00	"My name is Jonas!"	2012-09-19 12:55:02	5	6
4	2012-09-19 12:47:59	"Driving home from LA today!"	2012-09-19 12:40:01	1	3
5	2012-09-19 13:24:11	"This is my first Facebook comment"	2012-09-19 12:47:59	4	1

Likes table

like_id	like_created_time	comment_text	comment_created_time	like_profile_id	comment_profile_id
1	2012-09-19 12:22:01	"This is my first Facebook comment"	2012-09-19 12:47:59	3	1
2	2012-11-01 08:01:04	"Mondays...Am I right?"	2012-11-01 07:59:42	4	2
3	2012-09-19 13:30:00	"My name is Jonas!"	2012-09-19 12:55:02	5	6
4	2012-09-19 12:47:59	"Driving home from LA today!"	2012-09-19 12:40:01	1	3
5	2012-09-19 13:24:11	"This is my first Facebook comment"	2012-09-19 12:47:59	4	1

Notice that the first and fifth rows in this table make reference to the same comment. By including the actual comment and the comment created time, we're taking up extra space on disk.

It might be more efficient to store the comment information in its own table and simply make reference to the unique comment with a key that maps these two tables together.

Likes Table

like_id	created_time	profile_id	comment_id
1	2012-09-19 12:22:01	3	1
2	2012-11-01 08:01:04	4	2
3	2012-09-19 13:30:00	5	3
4	2012-09-19 12:47:59	1	4
5	2012-09-19 12:24:11	4	1

Comments Table

comment_id	created_time	comment_text	profile_id
1	2012-09-19 12:47:59	"This is my first Facebook comment"	1
2	2012-11-01 07:59:42	"Mondays...Am I right?"	2
3	2012-09-19 12:55:02	"My name is Jonas!"	6
4	2012-09-19 12:40:01	"Driving home from LA today!"	3

Profiles Table

profile_id	first_name	last_name	age	city	state
1	scott	hoover	30	Santa Cruz	CA
2	mike	xu	26	Fremont	CA
3	nate	pickens	26	Santa Cruz	CA
4	elena	simone	23	Santa Cruz	CA
5	margaret	rosas	37	Chicago	IL
6	lloyd	tabb	46	New York	NY

This whole set of tables and the way they relate to one another is referred to as "the schema."

By far, the most popular way to query relational tables is through SQL (a Structured Query Language), though there are others.

SQL is a declarative language. This means that the code we write in SQL tells our program what we want, but not precisely how to get it—that is, the sequence of events is not explicit like imperative programming.

If we want to print to the screen all of the ages in our profile table, our SQL statement would look something like this:

```
SELECT age
FROM profiles
```

In an imperative language we might have to specify the actual mechanism by why our program returns the results:

```
for (var i = 0; i < age.length; i++){
    print(age[i])
    }
```