Why Databases?

...

Intro to big data and databases

http://lazyprogrammer.me

Contemporary Issues

In the current environment you are probably mostly concerned with "big data", where both for-profit companies and the government download 1000s of TBs of data about you everyday. New and fancy technologies are popping up all the time, marketers and spammers love writing about them on LinkedIn, and gullible executives think they are must-haves.

Contemporary Issues

The talking heads at your workplace might say, "we need to build a scalable product!", or some such. So you end up creating a Hadoop cluster with a few tiny chunks of data and the overhead of your MapReduce actually takes longer than a for-loop by itself would have.

With all this fanciness you lose sight of the simple solutions - such as flat files, SQLite, and SQL. This article is a short survey of existing data solutions (both big data and small data) and at what scale they are appropriate for use.

http://lazyprogrammer.me

Why do you need data storage?

You are probably familiar with writing code in your first semester C++ class like this:

```
char* bob = "Bob";
char* jane = "Jane";
printf("Hi %s! Hi %s!\n", bob, jane);
```

In the real world, your code has to work on more cases than just Bob and Jane. Maybe you are writing an automated Twitter script that programmatically direct messages people when they start following you. If you use Twitter you've probably been annoyed at least a few times by this type of spam.

Why do you even need data storage?

Working off this example, suppose you (the spammer) decides that you're going to be somewhat nice and try not to spam people more than once.

So you would like to save the usernames you've direct messaged somewhere. Enter the flat file.

Flat Files

Flat files are great for storing small data or where you don't have to look stuff up. Just load the whole file into an array line by line, and do what you need to do.

In our case, we might load the data into a "set" datastructure so that when we want to look up a username, it's an O(1) search.

http://lazyprogrammer.me

Flat Files

Flat files are great for server configurations. As are JSON.

For scripts that automate something in your personal life, flat files are usually adequate.

A problem arises when you want to load your entire dataset into memory (like a set or a hash), and it doesn't fit. Remember, your hard drive is on the order of 1TB large. Your RAM is on the order of 8GB, much of which is used by the OS (or most if you're using Mac).

Why databases?

Enter the database. Databases are stored on disk. i.e. They are just a file or set of files.

The magic happens when you want to find something. Usually you'd have to look through the entire database if you didn't have some "index" (think like the index at the back of a large textbook) to tell you where everything was.

http://lazyprogrammer.me

Why databases?

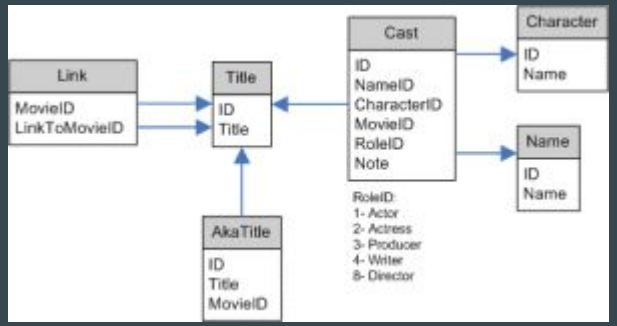
Databases index a whole bunch of metadata so that looking for stuff is really fast. You'll often see the term "balanced tree" in reference to database indexes. These are better than regular binary trees where searching is worst case O(N).

Relational Databases

Also called "RDBMS", short for "relational database management system" (they loved verbose terminology in the 80s and 90s), relational databases usually store things in tables.

Examples: MySQL, PostgreSQL.

For example, you might have one table that stores every user's ID, name, email, and password.



http://lazyprogrammer.me

Relational Databases

But you might have another table that stores friendships, so that would store the first user's ID, and the second user's ID.

Quite appropriately, relational databases keep track of "relationships", so that, suppose you deleted the user with ID = 3. That would delete all the rows from the friendships table that contain user ID = 3 also, so that in the application, there won't be any errors when it's looking for the friends of user ID = 5, who is friends with user ID = 3, when the actual user with ID = 3 has already been deleted.

Relational small data

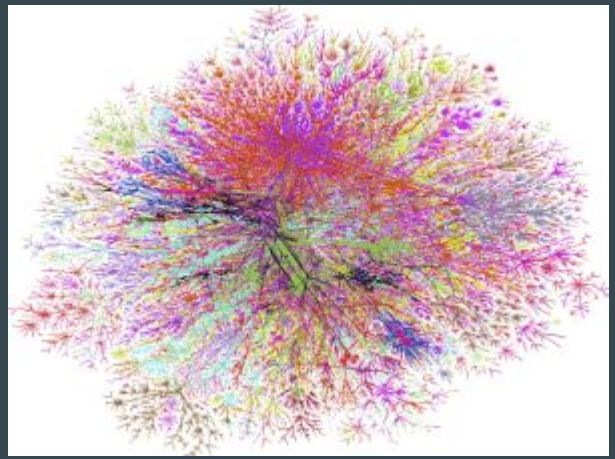
There is a special relational database called SQLite3. It works on "small data", so it's very appropriate for applications on your phone, for instance. iPhone apps on iOS use SQLite3. Many apps on your computer use SQLite3 without you even knowing it.

SQLite3 is stored locally on your machine, whereas bigger relational databases like Postgres can be stored either on your machine or on another machine over the Internet.

http://lazyprogrammer.me

Relational Big Data

Relational databases sort of hit a wall when data got too big to store in one database. Advertising companies can collect 1TB of data per day. In effect, you'd fill up an entire database in that one day. What do you do the next day? And the next?



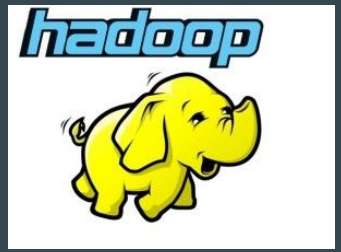
http://lazyprogrammer.me

Big Data - Hadoop

Hadoop is the open source version of Google's "Google File System" (GFS) and MapReduce framework.

Suppose for instance that your hard drives have a 1% chance of failing on any given day, and that your data is stored on 1000 hard drives. That means every day, 10 hard drives will fail. How do you make sure you don't lose this data? You replicate it.

Some very smart people have determined how many copies of your data must be stored so that, even though hard drives are basically guaranteed to fail, you will never lose your data.



http://lazyprogrammer.me

Big Data - Hadoop

In addition to data replication, the data is also spread across multiple "chunks". So multiple chunks (really files) make up one original data file.

MapReduce is a framework (a.k.a. a fancy way of writing a for loop), that distributes copies of the same program onto multiple machines, where each machine works on different chunks than the other machines.

Ideally, if you use N machines your running time would be reduced by 1/N, but there is lots of overhead that comes with coordinating the work that is done by each machine and merging it all together at the end.

Spark

Spark is seen as the "successor" to Hadoop MapReduce. I find that in general Spark jobs are a little easier to write. Note that it's a framework, NOT a database, but I list it here to ease the confusion.

We will return to Hadoop later, but first, more "big data" generation technologies.

http://lazyprogrammer.me

MongoDB

One database that became popular when startups started acquiring lots of data is MongoDB. MongoDB, unlike the other databases we've talked about, is not relational. In MongoDB, we don't have "tables", we have "collections". In MongoDB, we don't have "rows", we have "documents".

Documents are JSON documents. The nice thing about MongoDB is that you use Javascript to interact with it.

Startups started using the MEAN stack, which is made up of: MongoDB, ExpressJS, AngularJS, and NodeJS, for an all-Javascript environment.