

Xml or Sqlite, When to drop Xml for a Database? [closed]

Asked 16 years, 3 months ago Modified 2 years, 5 months ago

Viewed 42k times



57



As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Closed 12 years ago.

I really like Xml for saving data, but when does sqlite/database become the better option? eg, when the xml has more than x items or is greater than y MB?

I am coding an rss reader and I believe I made the wrong choice in using xml over a sqlite database to store a cache of *all* the feeds items. There are some feeds which have an xml file of ~1mb after a month, another has over 700 items, while most only have ~30 items and are ~50kb in size after a *several* months.

I currently have no plans to implement a cap because I like to be able to search through everything.

So, my questions are:

1. When is the overhead of sqlite/databases justified over using xml?
2. Are the **few large xml files** justification enough for the database when there are **a lot of small** ones, though even the small ones will grow over time? (a long *long* time)

updated (more info)

Every time a feed is selected in the GUI I reload all the items from that feeds xml file.

I also need to modify the read/unread status which seems really hacky when I loop through all nodes in the xml to find the item and then set it to read/unread.

xml

database

Share

Improve this question

Follow

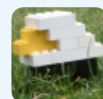
edited Aug 9, 2012 at 3:25



Bill the Lizard

405k ● 211 ● 572 ● 889


asked Sep 16, 2008 at 22:05



sieben

2,191 ● 4 ● 23 ● 31

-
- 4 I am confused by the reason for closing. I, personally, found this to be a helpful thread and am sad to see it killed. But aside from my personal feelings, can someone tell me what

"not constructive" means? It is not defined in the help center as one of the reasons for closing a thread. Is there a non-subjective definition of "not constructive" somewhere other than the help center? – [frozenjim](#) Apr 11, 2016 at 15:22 

18 Answers

Sorted by:

Highest score (default)



44



Man do I have experience with this. I work on a project where we originally stored all of our data using XML, then moved to SQLite. There are many pros and cons to each technology, but it was performance that caused the switchover. Here is what we observed.



For small databases (a few meg or smaller), XML was much faster, and easier to deal with. Our data was naturally in a tree format, which made XML much more attractive, and XPath allowed us to do many queries in one simple line rather than having to walk down an ancestry tree.

We were programming in a Win32 environment, and used the standard Microsoft DOM library. We would load all the data into memory, parse it into a DOM tree and search, add, modify on the in memory copy. We would periodically save the data, and needed to rotate copies in case the machine crashed in the middle of a write.

We also needed to build up some "indexes" by hand using C++ tree maps. This, of course would be trivial to do with SQL.

Note that the size of the data on the filesystem was a factor of 2-4 smaller than the "in memory" DOM tree.

By the time the data got to 10M-100M size, we started to have real problems. Interestingly enough, at all data sizes, XML processing was much faster than SQLite turned out to be (because it was in memory, not on the hard drive)! The problem was actually twofold- first, loadup time really started to get long. We would need to wait a minute or so before the data was in memory and the maps were built. Of course once loaded the program was very fast. The second problem was that all of this memory was tied up all the time. Systems with only a few hundred meg would be unresponsive in other apps even though we ran very fast.

We actually looking into using a filesystem based XML database. There are a couple open sourced versions XML databases, we tried them. I have never tried to use a commercial XML database, so I can't comment on them. Unfortunately, we could never get the XML databases to work well at all. Even the act of populating the database with hundreds of meg of XML took hours.... Perhaps we were using it incorrectly. Another problem was that these databases were pretty heavyweight. They required Java and had full client server architecture. We gave up on this idea.

We found SQLite then. It solved our problems, but at a price. When we initially plugged SQLite in, the memory and load time problems were gone. Unfortunately, since

all processing was now done on the harddrive, the background processing load went way up. While earlier we never even noticed the CPU load, now the processor usage was way up. We needed to optimize the code, and still needed to keep some data in memory. We also needed to rewrite many simple XPath queries as complicated multiquery algorithms.

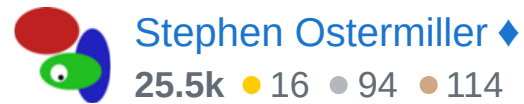
So here is a summary of what we learned.

1. For tree data, XML is much easier to query and modify using XPath.
2. For small datasets (less than 10M), XML blew away SQLite in performance.
3. For large datasets (greater than 10M-100M), XML load time and memory usage became a big problem, to the point that some computers become unusable.
4. We couldn't get any opensource XML database to fix the problems associated with large datasets.
5. SQLite doesn't have the memory problems of XML DOM, but it is generally slower in processing the data (it is on the hard drive, not in memory). (note- SQLite tables can be stored in memory, perhaps this would make it as fast.... We didn't try this because we wanted to get the data out of memory.)
6. Storing and querying tree data in a table is not enjoyable. However, managing transactions and indexing partially makes up for it.

Share Improve this answer

Follow

edited Jul 10, 2022 at 13:52



answered Sep 17, 2008 at 1:25



-
- 1 I've had a very similar experience - but from .NET. Using XML serialization, the initial XML version of the code was a breeze to write, and performed quite well initially. Since I wasn't using the DOM, I didn't get the memory blow-up problem you describe (it would have shown up later). The real problem, however, was updates - even a trivial update to a 30MB xml file (not at all large) will take seconds to write to disk. Make it a bit bigger... The sqlite version is much more involved, but it's multi-proc safe now, a feature that turned out useful once it was there. – [Eamon Nerbonne](#) May 5, 2011 at 6:50
-



24



I basically agree with [Mitchel](#), that this can be highly specific depending on what are you going to do with XML and SQLite. For your case (cache), it seems to me that using SQLite (or other embedded databases) makes more sense.



First I don't really think that SQLite will need more overhead than XML. And I mean both development time overhead and runtime overhead. Only problem is that you have a dependence on SQLite library. But since you would need some library for XML anyway it doesn't matter (I assume project is in C/C++).

Advantages of SQLite over XML:

- everything in one file,
- performance loss is lower than XML as cache gets bigger,
- you can keep feed metadata separate from cache itself (other table), but accessible in the same way,
- SQL is probably easier to work with than XPath for most people.

Disadvantages of SQLite:

- can be problematic with multiple processes accessing same database (probably not your case),
- you should know at least basic SQL. Unless there will be hundreds of thousands of items in cache, I don't think you will need to optimize it much,
- maybe in some way it can be more dangerous from security standpoint (SQL injection). On the other hand, you are not coding web app, so this should not happen.

Other things are on par for both solutions probably.

To sum it up, answers to your questions respectively:

1. You will not know, unless you test your specific application with both back ends. Otherwise it's always just a guess. Basic support for both caches should not be a problem to code. Then benchmark and compare.

2. Because of the way XML files are organized, SQLite searches should always be faster (barring some corner cases where it doesn't matter anyway because it's blazingly fast). Speeding up searches in XML would require index database anyway, in your case that would mean having cache for cache, not a particularly good idea. But with SQLite you can have indexing as part of database.

Share Improve this answer

Follow

edited Jul 10, 2022 at 13:54



Stephen Ostermiller ♦

25.5k ● 16 ● 94 ● 114

answered Sep 16, 2008 at 22:40



Stan

2,599 ● 18 ● 27



Don't forget that you have a great database at your fingertips: the filesystem!

16



Lots of programmers forget that a decent directory-file structure is/has:



1. It's fast as hell
2. It's portable
3. It has a tiny runtime footprint

People are talking about splitting up XML files into multiple XML files... I would consider splitting your XML into multiple directories and multiple plaintext files.

Give it a go. It's refreshingly fast.

Share Improve this answer

answered Sep 16, 2008 at 22:26

Follow



Oli

240k ● 65 ● 226 ● 303

6 The filesystem hierarchy is only fast in Unix derivatives. It's horrendously slow in Windows. – [apenwarr](#) Sep 16, 2008 at 22:39

2 That's not true - I've done this on windows too, and it often blows SQL db's out of the water. Even if you store *huge* number of files in a single directory (which in this myth is slow) you don't run into issues. Don't, however, browse to the directory in windows explorer - that can't deal with large directories. – [Eamon Nerbonne](#) May 5, 2011 at 6:56



7



1. Use XML for data that the application should know - configuration, logging and what not.

2. Use databases(oracle, SQL server etc) for data that the user interacts with directly or indirectly - real data

3. Use SQLite if the user data is more of a serialized collection - like huge list of files and their content or collection of email items etc. SQLite is good at that.

Depends on the kind and the size of the data.

Share Improve this answer

answered Sep 16, 2008 at 22:22

Follow



Vin

6,145 ● 4 ● 44 ● 56



I wouldn't use XML for storing RSS items. A feed reader makes constant updates as it receives data.

5



With XML, you need to load the data from file first, parse it, then store it for easy search/retrieval/update. Sounds like a database...



Also, what happens if your application crashes? if you use XML, what state is the data in the XML file versus the data in memory. At least with SQLite you get atomicity, so you are assured that your application will start with the same state as when the last database write was made.

Share Improve this answer

answered Sep 16, 2008 at 22:09

Follow



[typicalrunt](#)

671 ● 1 ● 6 ● 12

These were the things I was thinking of, but I thought I might have been making change for changes sake. – [sieben](#) Sep 16, 2008 at 22:28

The typical solution to atomicity would be to write a new version; delete the normal filename; move new version to the normal filename. Depending on your needs, throwing in an fsync may be advised. – [Eamon Nerbonne](#) May 5, 2011 at 6:52



XML is best used as an interchange format when you need to move data from your application to somewhere else or share information between applications. A

5



database should be the preferred method of storage for almost any size application.



Share Improve this answer

answered Sep 16, 2008 at 22:09

Follow



[Bradley Harris](#)

932 ● 1 ● 6 ● 12



5



When should XML be used for data persistence instead of a database? Almost never. XML is a data transport language. It is slow to parse and awkward to query. Parse the XML (don't shred it!) and convert the resulting data into domain objects. Then persist the domain objects. A major advantage of a database for persistence is SQL which means unstructured queries and access to common tools and optimization techniques.



Share Improve this answer

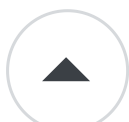
answered Sep 17, 2008 at 1:15

Follow



[David Medinets](#)

5,588 ● 3 ● 32 ● 44



5



I have made the switch to SQLite and I feel *much* better knowing it's in a database.

There are a lot of other benefits from this:

- Adding new items is really simple
- Sorting by multiple columns
- Removing duplicates with a unique index



I've created 2 views, one for unread items and one for all items, not sure if this is the best use of views, but I really wanted to try using them.

I also benchmarked the xml vs sqlite using the **StopWatch** class, and the sqlite is faster, **although it could just be that my way of parsing xml files wasn't the fastest method.**

1. Small # items and size (25 items, 30kb)

- ~1.5 ms sqlite
- ~8.0 ms xml

2. Large # of items (700 items, 350kb)

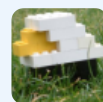
- ~20 ms sqlite
- ~25 ms xml

3. Large file size (850 items, 1024kb)

- ~45 ms sqlite
- ~60 ms xml

Share Improve this answer
Follow

answered Sep 17, 2008 at 22:30



sieben

2,191 ● 4 ● 23 ● 31

could you check that against files with 100 mb, 250 mb and 500 mb of data ? – [Erdinc Ay](#) Sep 18, 2017 at 13:32



3

To me it really depends on what you are doing with them, how many users/processes need access to them at the same time etc.



I work with large XML files all the time, but they are single process, import style items, that multi-user, or performance are not really needs.

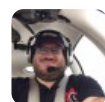


SO really it is a balance.

Share Improve this answer

Follow

answered Sep 16, 2008 at 22:07



[Mitchel Sellers](#)

63.1k ● 15 ● 114 ● 174



2

If any time you will need to scale, use databases.

Share Improve this answer

Follow

answered Sep 16, 2008 at 22:23



[Mostlyharmless](#)

2,283 ● 19 ● 28



2

XML is good for storing data which is not completely structured and you typically want to exchange it with another application. I prefer to use a SQL database for data. XML is error prone as you can cause subtle errors due to typos or omissions in the data itself. Some open





source application frameworks use too many xml files for configuration, data, etc. I prefer to have it in SQL.

Since you ask for a rule of thumb, I would say that use XML based application data, configuration, etc if you are going to set it up once and not access/search it much. For active searches and updations, its best to go with SQL.

For example, a web server stores application data in a XML file and you dont really need to perform complex search, update the file. The web server starts, reads the xml file and thats that. So XML is perfect here. Suppose you use a framework like Struts. You need to use XML and the action configurations dont change much once the application is developed and deployed. So again, the XML file is a good way. Now if your Struts developed application allows extensive searches and updations, deletions, then SQL is the optimal way.

Offcourse, you will surely meet one or two developers in your organisation who will chant XML or SQL only and proclaim XML or SQL as the only way to go. Beware of such folks and do what 'feels' right for your application. Dont just follow a 'technology religion'.

Think of things like how often you need to update the data, how often you need to search the data. Then you will have your answer on what to use - XML or SQL.

Share Improve this answer



echarcha

Follow



1



I agree with @Bradley.

XML is very slow and not particularly useful as a storage format. Why bother? Will you be editing the data by hand using a text editor? If so, XML *still* isn't a very convenient format compared to something like YAML. With something like SQLite, queries are easier to write, and there's a well defined API for getting your data in and out.

XML is fine if you need to send data around between programs. But in the name of efficiency, you should probably produce the XML at sending time, and parse it into "real data" at receive time.

All the above means that your question about "when the overhead of a database is justified" is kind of moot. XML has a way higher overhead, all the time, than SQLite does. (Full-on databases like MSSQL are heavier, especially in administrative overhead, but that's a totally different question.)

Share Improve this answer

answered Sep 16, 2008 at 22:15

Follow



[apenwarr](#)

11k ● 6 ● 50 ● 59

Yeah, sqlite is very fast. I was googling about xml vs sqlite and they were all saying that xml was better because of the database overhead. – [sieben](#) Sep 16, 2008 at 22:24

XML can be stored as text and as a binary file format.



1

If your primary goal is to let a computer read / write a file format effeciently you should work with a binary file format.



Databases are an easy to use way of storing and maintaining data. They are not the fastest way to store data that is a binary file format.



What can speed things up is using an in memory database / database type. Sqlite has this option.

And this sounds like the best way to do it for you.

Share Improve this answer

answered Sep 16, 2008 at 22:53

Follow



Mischa Kroon



1

My opinion is that you should use SQLite (or another appropriate embedded database) anytime you don't need a pure-text file format. Note, this is a pretty big exception. There are a lot of scenarios that require, or are benefited by, pure-text file formats.



As far as overhead goes, SQLite compiles to something like 250 k with normal flags. Many XML parsing libraries are larger than SQLite. You get no concurrency gains using XML. The SQLite binary file format is going to support much more efficient writes (largely because you can't append to the end of a well-formatted XML file). And even reading data, most of which I assume is fairly random access, is going to be faster using SQLite.

And to top it all off, you get access to the benefits of SQL like transactions and indexes.

Edit: Forgot to mention. One benefit of SQLite (as opposed to many databases) is that it allows any type in any row in any column. Basically, with SQLite you get the same freedom you have with XML in terms of datatypes. This also means that you don't have to worry about putting limits on text columns.

Share Improve this answer

answered Sep 16, 2008 at 22:53

Follow



Jay Stramel

3,301 ● 4 ● 29 ● 25



1

You should note that many large Relational DBs (Oracle and SQLServer) have XML datatypes to store data within a database and use XPath within the SQL statement to gain access to that data.



Also, there are native XML databases which work very much like SQLite in the sense they are one binary file holding a collection of documents (which could roughly be a table) then you can either XPath/XQuery on a single document or the whole collection. So with an XML database you can do things like store the days data as a separate XML document in the collection... so you just need to use that one document when your dealing with the data for today. But write an XQuery to figure out historical data on the collection of documents for that person. Slick.

I've used Berkeley XMLDB (now backed by Oracle). There are others if you search google for "Native XML Database". I've not seen a performance problem with storing/retrieving data in this manner.

XQuery is a different beast (but well worth learning), however you may be able to just use the XPath's you currently use with slight modifications.

Share Improve this answer

answered Sep 17, 2008 at 19:27

Follow



Nika



1

A database is great as part of your program. If querying the data is part of your business logic. XML is best as a file format, especially if you data format is:



- 1, Hierarchal
- 2, Likely to change in the future in ways you can't guess
- 3, The data is going to live longer than the program



Share Improve this answer

answered Sep 17, 2008 at 19:48

Follow



[Martin Beckett](#)

96k ● 28 ● 195 ● 268



0

I say it's not a matter of data size, but of data type. If your data is *structured*, use a relational database. If your data is *semi-structured*, use XML or - if the data amounts really grow too large - an XML database.



Share Improve this answer

answered Sep 16, 2008 at 22:09



Follow



[Sebastian Redl](#)

71.7k ● 8 ● 132 ● 167



0

If your searching go with a db. You could split the xml files up into directories to ease seeking, but the managerial overhead easily gets quite heavy. You also get a lot more than just performance with a sql db...



Share Improve this answer

answered Sep 16, 2008 at 22:12



Follow



[Andrew Taylor](#)

