MariaDB 5.2: Using MariaDB as a document store and Virtual Columns for indexing

Submitted by: hingo
on Thu, 2010-10-28 22:05

MariaDB MySQL NoSQL

This is a followup to my previous post about Virtual Columns. In this post I will do a more in-depth test on using virtual columns in a use case where MariaDB is used as a

Document oriented DB basics

Relational databases store data in 2-dimensional tables, rows and columns. Document oriented databases do not store Word documents or novels, rather the "documents" are essentially serialized PHP arrays, Java-objects, etc. The most popular document format today is JSON. JSON "documents" have exactly the same syntax as a JavaScript array, originally JavaScript programmers would just `eval("data = " + jsondocument);` to get the data into a variable (unserialize). For more information about JSON and how a JSON database would work, see the Wikipedia article and for instance the MongoDB manual.

Another format that has been used for document stores is XML documents, and many programming languages can serialize objects into XML.

In my exploration I've used XML, since MySQL/MariaDB has XML XPath functions builtin since 5.1. The below approach could also be used for JSON or any other format, but then you'd need functions to parse JSON documents, and by googling I couldn't even find a User Defined Function that would do that. (And even if I had one, Virtual Columns doesn't seem to support UDF functions.)

Using MariaDB as a document store (schemaless)

MySQL has been successfully used in a document store resembling way. A couple years ago Friendfeed posted an excellent narrative of their migration to what they call a "schemaless" MySQL database. They concluded that this approached allowed them to stay on MySQL as a stable, proven and familiar database, rather than migrating to one of the new NoSQL databases.

I have personally also used this approach sometimes when quickly putting together some PHP site, since I like the flexibility I get wh new elements into

Search

Search

Recent blog posts

Accord in plain English (without greek letter notation)

resources on Paxos

Using Math and Science to cope with Parkinson's

Is it worth exercising options early?

Sakta vi gå genom stan

On coaching and managing

A scalability model for Cassandra

Secondary indexes in Cassandra

OSI State of the Source 2020: In Defense of Extreme Copyleft

RUM Conjecture for Beginners

more

Recent comments

So what you do is that you simply fall back to using MariaDB as a simple key-value store, such as:

```
CREATE TABLE xmlstore (
   id INTEGER UNSIGNED NOT NULL,
   doc TEXT,
   PRIMARY KEY (id)
);

INSERT INTO xmlstore (id, doc) VALUES (1,
'<user>
 <id>1</id>
 <username>hingo</username>
 <name>Henrik Ingo</name>
 <status time="2010-21-10 13:16">I\'m writing an example XML document</status>
 <friends>
  <friend_id>9</friend_id>
  <friend_id>91</friend_id>
  <friend_id>92</friend_id>
  <friend_id>93</friend_id>
  <friend_id>94</friend_id>
  <friend_id>95</friend_id>
  <friend_id>96</friend_id>
  <friend_id>97</friend_id>
  <friend_id>98</friend_id>
  <friend_id>99</friend_id>
 </friend>
</user>');
```

Now your actual record is completely inside the `doc` BLOB. MariaDB doesn't care about the fields in it. Different records can have different fields. If you want to change the "schema", you simply store some different data on the application level and push that into MariaDB instead.

The uptake here compared to a traditional fully normalized RDBMS approach is flexibility, and apparently also performance in FriendFeed's case. Naturally you lose the ability to do any queries on the fields in the XML document, you can just fetch by primary key, which you therefore need to know. This is why document stores are also called key-value databases.

Using Virtual Columns to create secondary indexes for the document

But suppose we also need to fetch this data based on the `username`. Thanks to virtual columns, we can actually add an automatically generated column and index it. MySQL 5.1 introduced the `ExtractValue()` function where you can use XPATH expressions to extract snippets from an XML document:

```
MariaDB [test]> DROP TABLE xmlstore;
Query OK, 0 rows affected (0.04 sec)

MariaDB [test]> CREATE TABLE xmlstore (
   id INTEGER UNSIGNED NOT NULL,
   doc TEXT,
   username CHAR(10) AS (ExtractValue(doc, '/user/username')) PERSISTENT,
   PRIMARY KEY(id),
   KEY (username) );
```


The range query could be...
3 months ago

Interesting approach. I have...
4 months ago

Try this one.
10 months ago

4 round trips
1 year 9 months ago

I tried to understand the...
1 year 9 months ago

Paxos in Cassandra
1 year 10 months ago

Thanks for sharing the pain...
1 year 10 months ago

Replacing the locks
2 years 7 months ago

10% bound on space-amp
2 years 10 months ago

All time:

Bruce Perens needs your help in re-joining the OSI board

The current and future of Free Culture... or whatever you may want to call it.

How to grow your open source project 10x and revenues 5x

Slides from Failover or not Failover, that is the question

Open Life: The Philosophy of Open

confirmed this is a bug and will be fixed in future releases. (Actually, it seems to have been excluded already in Zhakov's original implementation.)

That was kind of a bummer. While waiting for the bug to be fixed, I can still continue my testing just using `SUBSTR()`:

```
-- 10 is a magic number, it is the length of the string '<username>'

MariaDB [test]> CREATE TABLE xmlstore (
    ->   id INTEGER UNSIGNED NOT NULL,
    ->   doc TEXT,
    ->   username CHAR(10) AS (SUBSTR(doc,
    ->          LOCATE('', doc)+10,
    ->          LOCATE('', doc)-(LOCATE('', doc)+10))
    ->       ) PERSISTENT,
    ->   PRIMARY KEY (id),
    ->   KEY (username));
Query OK, 0 rows affected (0.01 sec)

-- same insert as above...

MariaDB [test]> SELECT * FROM xmlstore\G
*************************** 1. row ***************************
      id: 1
     doc: <user>
 <id>1</id>
 <username>hingo</username>
 <name>Henrik Ingo</name>
 <status time="2010-21-10 13:16">I'm writing an example XML document</status>
 <friends>
  <friend_id>9</friend_id>
  <friend_id>91</friend_id>
  <friend_id>92</friend_id>
  <friend_id>93</friend_id>
  <friend_id>94</friend_id>
  <friend_id>95</friend_id>
  <friend_id>96</friend_id>
  <friend_id>97</friend_id>
  <friend_id>98</friend_id>
  <friend_id>99</friend_id>
 </friend>
</user>

username: hingo
1 row in set (0.00 sec)

MariaDB [test]> EXPLAIN SELECT * FROM xmlstore WHERE username='hingo'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: xmlstore
         type: ref
possible_keys: username
          key: username
      key_len: 31
          ref: const
         rows: 1
        Extra: Using where
1 row in set (0.00 sec)
```

Last viewed:

- Speaking at MySQL Connect (OpenWorld)
- Running sysbench tests against a Galera cluster
- Translating reliably between XML and JSON (xml2json)
- Navigator, Explorer, Konqueror, Safari
- Authoring Impress.js presentations in Markdown
- avoinelama.fi (Finnish)

So while waiting for `ExtractValue()` to become supported, we can still use `SUBSTR()` for any non-complex XML document to extract the node that we want to index, and a SELECT on it will use the index. This is now a very useful document oriented database we have created.

Limitations: no secondary indexes for shards

The main limitation I can think of here is that the secondary indexes created with virtual columns cannot be sensibly used if your database is sharded. But now that I think about it, this is true for secondary indexes just in general.

The FriendFeed narrative has a nice example how they used a separate table to implement each of their index. (And each such table is then sharded on its own.) This has the added benefit that it nicely also works around the limitation that MySQL/InnoDB doesn't support online ALTER TABLE operations, so you cannot (easily) add or drop indexes on a large production database, but you can easily create and drop tables! (But see Facebook's Online Schema Change for a nice workaround-tool.) On the other hand, it then becomes a burden for the application layer to maintain all indexes, whereas with virtual columns this is handled by the database. Also in the FriendFeed implementation the secondary "index tables" are not guaranteed to be consistent with the actual data table (because you don't have transactions across shards) which makes it even trickier for the app layer to actually use them.

I guess MySQL Cluster again deserves a mention here: it has transparent sharding, including for secondary indexes. These are consistent across the cluster, they are updated and read within a transaction. And nowadays it even sports ADD INDEX as online operation! I suppose this is yet another technique where MySQL Cluster might just rock!

Coming up next: The benchmark

I mentioned I did some sysbench runs on the above schema. I have already written enough for one post, so I will save that for a separate post. The bet is still on :-)

Justin Swanhart (not verified)

Thu, 2010-10-28 23:44

Permalink

secondary indexes

Don't forget shard-query which can execute a query across all shards and return the result(and it support aggregate functions): http://code.google.com/p/shard-query

I'll be extending it shortly to support partition elimination either using a directory server or by a custom function. Queries that include the shard key will only go to the necessary shard, other queries will be broadcast to all shards.

This would work well with your document model and online-schema-change.

Reply

hingo

Sat, 2010-10-30 21:22

Shard query

Thanks! I had not heard of this before, but will certainly look into

If you will work on it, I suppose you will blog about it. If you blog about it, please make a short comparison against Spider engine.

Notice that the ability to just broadcast the SELECT ... WHERE username=x to all shards is not a sufficient solution. In a truly shard-supporting solution we'd want also the secondary index to be sharded on it's own. This means that the client knows where to connect to find the index position username=x and this is typically a different shard than where the actual record(s) is located.

So for instance in MySQL Cluster secondary indexes are essentially just tables of their own, partitioned on their own. When you scan the index, you get the primary key(s), from where you then know which partition/shard the actual record is. In MySQL Cluster all of this is transparent to the user, but the performance is of course different compared to a solution where someone would scan all partitions to find the right place in the index.

Reply

Justin Swanhart (not verified)

Tue, 2010-11-02 02:49

Permalink

I have been making some

I have been making some changes to Shard-Query in svn. It now supports three callback functions. The first must return a list of servers, the second must return a list of column names that can be partitioned on, and the third accepts two parameters (a column name and a column value) and it must return an index into the list of servers.

The following examples assume that 'calendar_id' is the partitioning key. I'll be blogging about this in more depth soon.

This query is pushed to all shards and then the results are aggregated:

```
select calendar_id, count(*)
from partitioned_table
group by calendar_id;
```

This query could be routed to a particular server, as long as calendar_id is a partition key. The callback will send ('calendar_id',1) which should return the shard to connect to to run the query:

```
select calendar_id, event_type, count(*)
from partitioned_table
where calendar_id = 1
group by 1, 2;
```

This query is routed to all servers and then aggregated. Optionally, it can be sent as three equality queries to each shard:

```
select calendar_id, event_type, count(*)
from partitioned_table
where calendar_id in (1,2,3)
group by 1, 2;
```

The system supports more than one sharding key.

Reply

Brian Aker Aker (not verified)

Fri, 2010-10-29 01:11

Permalink

You do know that a column store, and a covering index are two different things?
http://en.wikipedia.org/wiki/Column-oriented_DBMS

What you are showing off is a covering index, which MyISAM supports, but that is not the same thing as a column store. If you go back and look at my blog I have some comparisons of the two and the difference (and Stonebraker has an entire paper dedicated to this subject as well).

Cheers,
-Brian

Reply

hingo

Sat, 2010-10-30 21:13

Permalink

Title was wrong, corrected. (thanks)

Oops! The title was wrong. (The first paragraph was and is correct.) This article is about using MariaDB as a *document store*, not column store.

I corrected the title now. Funny mistake.

Reply

Justin Swanhart (not verified)

Fri, 2010-10-29 21:07

Permalink

Storing schemaless data in an

Storing schemaless data in an RDBMS (the E-BLOB (anti?)pattern) in the manner described is fairly routine.

The indexing that you are suggesting could easily be done with a trigger, and that would work with extractXML.

What advantage do persistent virtual columns have over a triggers except syntactic sugar?

Reply

hingo

Sat, 2010-10-30 21:11

Permalink

syntactic sugar

Storing schemaless data in an RDBMS (the E-BLOB (anti?)pattern) in the manner described is fairly routine.

Unless you work for one of the popular web2.0 companies you probably don't do it. All database developers were thought to normalize, normalize, normalize. When I talk about de-normalization to organizations in Europe, the response is usually a clear "No thanks". For you guys that do this kind of thing routinely, I would welcome more posts like what FriendFeed wrote. It's also good to educate people that you actually can do this in MySQL/MariaDB, you don't need to migrate to some NoSQL database just to get this pattern.

What advantage do persistent virtual columns have over a triggers except syntactic sugar?

plication level. Performance-wise application level is often even the better choice, preferable also to TRIGGER.

Even so, it is one feature in the list of things some proprietary databases have, and now MariaDB has it too. Now that we have it, we of course should use it since it is simpler than triggers. Stored Procedures and Triggers used to be on that list too :-)

Reply

hingo

Sun, 2010-10-31 22:07

Permalink

Seems that virtual columns are faster!

Well whad'ya know: Turns out that Virtual Columns are faster than Views and putting stuff literally in the SELECT query, and in this case might even be faster (or equal) than doing it on application level. See my next blog post for details.

Reply

Justin Swanhart (not verified)

Tue, 2010-11-02 02:31

Permalink

Persistent virtual columns

Persistent virtual columns will naturally be faster than expressions in a SELECT clause or VIEW, since they are calculated when the row changes and stored with the row (that is, they are persistent). You wouldn't be able to index them otherwise.

I have another project called Flexviews (flexviews.sourceforge.net) which lets you create materialized views. A materialized view is like a virtual column, except for an entire query is materialized and kept up to date based on row changes.

Reply

hingo

Tue, 2010-11-02 07:22

Permalink

That's what I thought too...

That's what I assumed too, but my laptop-run benchmarks suggest that the VIRTUAL columns are faster too, and equally fast to persistent columns. Remains to be confirmed also on larger server.

Reply

Add new comment

Your name *

Notify me when new comments are posted

Email

The content of this field is kept private and will not be shown publicly. Cookie & Privacy Policy

Subject

Comment *

About text formats

Plain text

No HTML tags allowed.

External and mailto links in content links have an icon.

Lines and paragraphs break automatically.

Web page addresses and email addresses turn into links automatically.

Use [fn]...[/fn] (or <fn>...</fn>) to insert automatically numbered footnotes.

Each email address will be obfuscated in a human readable fashion or, if JavaScript is enabled, replaced with a spam resistent clickable link. Email addresses will get the default web form unless specified. If replacement text (a persons name) is required a webform is also required. Separate each part with the "|" pipe symbol. Replace spaces in names with "_".

Save

Drupal spam blocked by CleanTalk.

About the book About this site Academic Accord Amazon Beginners Books BuildBot Business models bzr Cassandra Cloud cloud computing cls Community communityleadershipsummit Consistency coodiary Copyright Creative Commons css Databases datamining Datastax DevOps Distributed Consensus Drizzle Drupal Economy electron Ethics Eurovision Facebook Froscon Funny Galera GIS github Gnome Governance HandlerSocket High Availability impressionist impressjs Inkscape Internet JavaScript json KDE Kubuntu Licensing Linux Maidan Maker culture MariaDB markdown MEAN stack MepSQL Microsoft Mobile MongoDB MontyProgram Music MySQL MySQL Cluster Nerds Node NoSQL odba Open Content Open Source OpenSQLCamp Oracle OSCon PAMP Patents Percona performance Personal Philosophy PHP Pirates PlanetDrupal Politics PostgreSQL Presales presentations Press releases Programming Red Hat Replication Severalnines Silly SkySQL

Contact Privacy Policy

Designed by: Golems G.A.B.B. OÜ