Scylla DB analysis and comparaison with MySQL / MariaDB

For a long time we have been interested in NoSQL databases in order to improve performances, have a flexible data model and scalability.
MongoDB was our first choice (towards the end of 2016 – at the beginning of our adventure), but migration was not made due to huge changes to do, our lack of free time at this period, no further studies about it and we had instead put a MySQL replication.
Recently, we are looking again at a migration to a NoSQL database, for the same reasons and because it is better suited for organization features that we are developing.

Cassandra seems good for us with its ability to handle large amounts of data across many servers, its performance compared to other NoSQL databases and its high availability with no single point of failure. CQL is close to SQL, it will be easier for us editing the code for migration.

An important point : users must have a performance gain in uploading and downloading files.

We will use DataStax PHP Driver, implemented like an extension and which uses C/C++ driver.
This choice is motivated by the fact that DataStax is a main contributor of Cassandra project and drivers are maintened.

Not long after, we discovered Scylla which is compatible and similar to Cassandra but written in C++ instead of Java. It offers significantly higher throughputs and lower latencies and we will verify this in our tests.

Data consistency

MySQL

https://stackoverflow.com/questions/25690976/does-mysql-replication-have-immediate-data-consistency

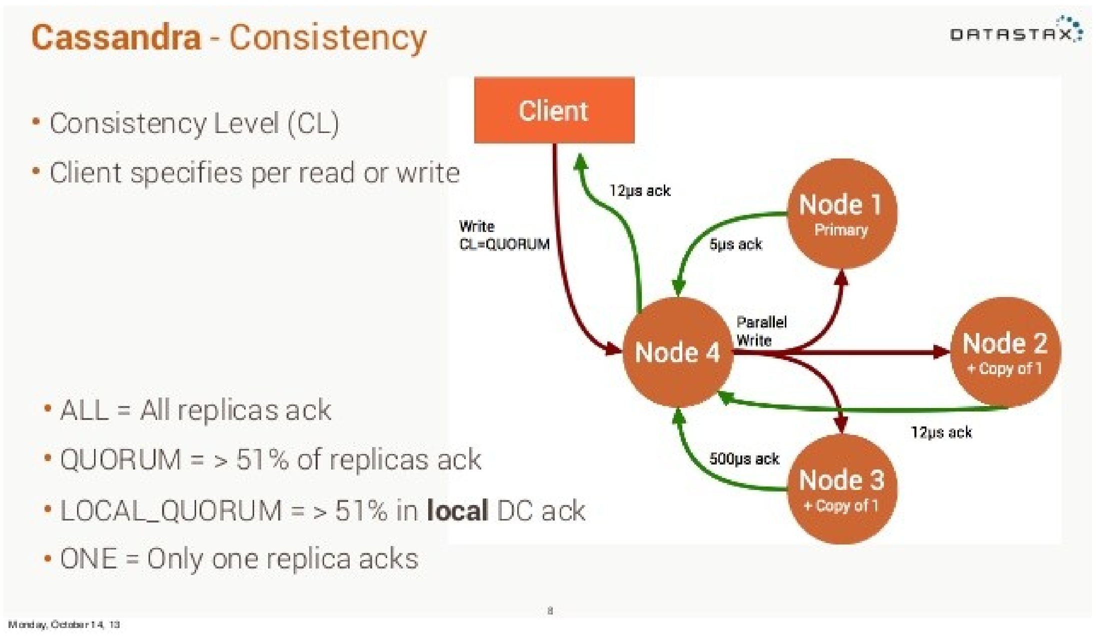
« Replication in MySQL is asynchronous by default, or "semi-synchronous" at best. Certainly it does lag in either case. In fact, the replication slave is always lagging behind at least a fraction of a second, because the master doesn't write changes to its binary log until the transaction commits, then the slave has to download the binary log and relay the event.

But the changes are still atomic. You can't read data that is partially changed. You either read committed changes, in which case all constraints are satisfied, or else the changes haven't been committed yet, in which case you see the state of data from before the transaction began.
So you might temporarily read *old* data in a replication system that lags, but you won't read *inconsistent* data.
Whereas in an "eventually consistent" system, you might read data that is partially updated, where the one account has been debited but the second account has not yet been credited. So you *can*see inconsistent data. »

Cassandra

There is in the docs a section about data consistency on Cassandra: https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlDataConsistencyTOC.html

To sum up, consistency on Cassandra is tunable : the client can decide the appropriate consistency level (zero, any, one, quoram or all). The consistency level controls both read and write behavior based on your replication factor. In a single node cluster the consistency levels any, one, quorom and all are equivalent.



Cassandra - Consistency

- Consistency Level (CL)
- Client specifies per read or write

- ALL = All replicas ack
- QUORUM = > 51% of replicas ack
- LOCAL_QUORUM = > 51% in **local** DC ack
- ONE = Only one replica acks

Figure 1 – Cassandra consistency - jeremiahdjordan - https://www.slideshare.net/jeremiahdjordan/cassandra-intro-27171544

Tests

We have set up VMs with Ubuntu 16.04, 4 GB of RAM, 16 GB storage on HDD, 1 CPU core per machine (i5 4690K). PHP 7.0 and Apache2, Last stable version of MariaDB / Cassandra / Scylla according to the machine. Caches are disabled.

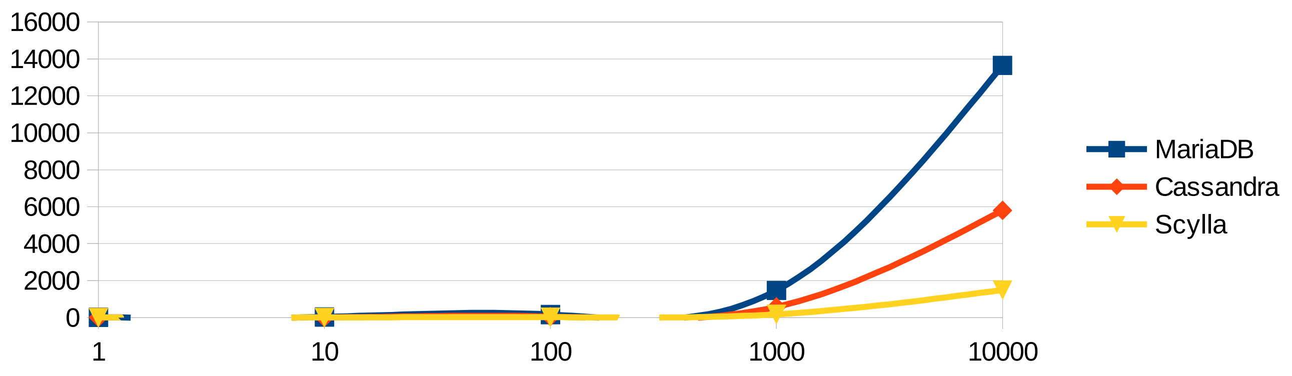
Tests consist in writting 10K rows, reading them one per one, reading them with a prepared statement, updating them and deleting them. We could have done with more rows but it already shows clearly the differences.
Cassandra and Scylla are on a single node.
We did several measures for the same operation in order to check the reliability of our results.

We used the same table (song) with song_id as primary key (int in MariaDB, uuid in Cassandra/Scylla), artist (text), title (text), album (text) and populated with realistic data.

<u>Measurements are in ms.</u>

Inserting

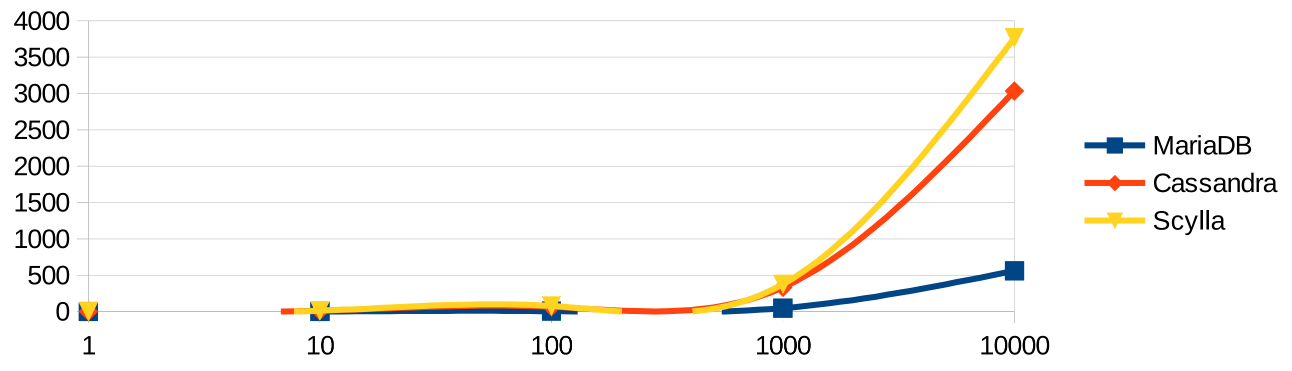
	1	*10*	*100*	*1000*	*10000*	*CPU*	*RAM*	*Operations/s*
MariaDB	17,17	34,67	165,1	1466,92	13651,2	27 %	32 %	733
Cassandra	0,74	4,3	55,73	578,43	5798,53	26 %	31 %	1725
Scylla	1,2	3,76	20,26	174,7	1493,15	26 %	31 %	6697



Scylla is the fastest (3.88 times more than Cassandra and 9.13 times more than MariaDB). Results correspond to our expectations and forecasts.

Reading (one per one)

	1	*10*	*100*	*1000*	*10000*	*CPU*	*RAM*	*Operations/s*
MariaDB	0,06	0,49	4,61	47	558,92	27 %	32 %	17891
Cassandra	0,73	11,9	64,88	333,13	3033,48	26 %	31 %	3297
Scylla	0,78	11,13	76,82	374,7	3765,63	26 %	31 %	2656



For reading MariaDB is faster and really good at that. We expected less difference but it should be noted that data are to be modeled differently : with Cassandra/Scylla, we must optimize reading operations by reducing and simplifying them (this example is a simple query) at the expense of writing. This is not a problem if the data is multiplied, as writing does not cost much. For a more complex query, the difference will not be as big and Cassandra could be faster. We can also use caching.

Reading (prepared statement, simple query, 10K rows)

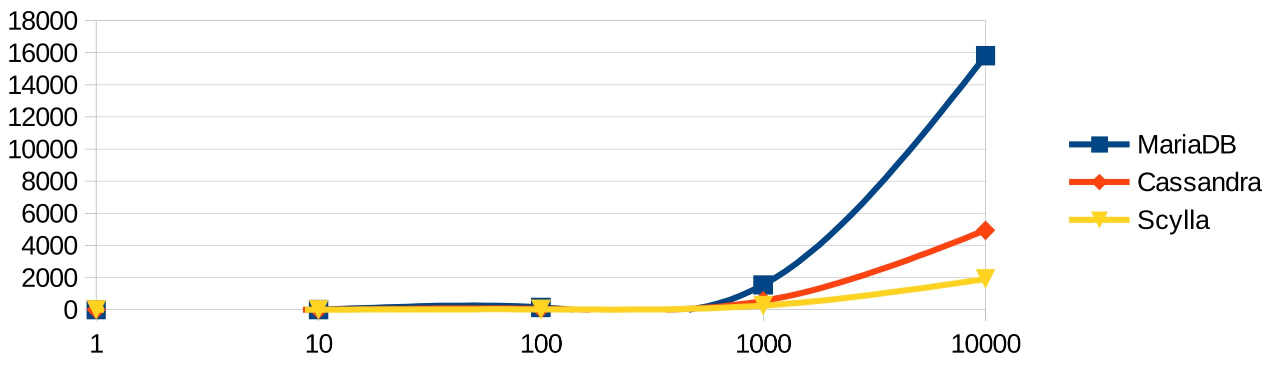
		CPU	*RAM*
MariaDB	4,58	28 %	32 %
Cassandra	174,68	28 %	32 %
Scylla	160,25	28 %	32 %

Reading (prepared statement, query with a JOIN on MySQL)

		CPU	*RAM*
MariaDB	651,13	32 %	34 %
Cassandra	224,3	28 %	32 %
Scylla	208,25	28 %	33 %

Updating

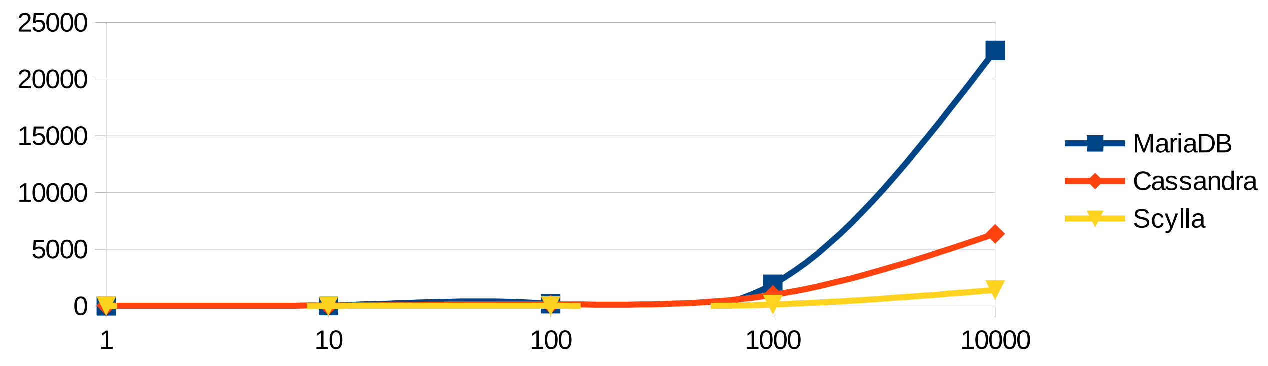
	1	*10*	*100*	*1000*	*10000*	*CPU*	*RAM*	*Operations/s*
MariaDB	2,77	14,02	143,84	1543,6	15811	32 %	32 %	632
Cassandra	1,33	6,51	56,42	551,02	4946,93	32 %	32 %	2021
Scylla	0,51	1,79	23,85	257,92	1918,92	32 %	32 %	5211



Scylla is the fastest (2.58 times more than Cassandra and 8.25 times more than MariaDB).

Deleting

	1	*10*	*100*	*1000*	*10000*	*CPU*	*RAM*	*Operations/s*
MariaDB	7,85	23,45	189,25	1905,78	22529,6	36 %	32 %	444
Cassandra	28,1	39,84	138,25	977,18	6358,24	31 %	32 %	1573
Scylla	0,66	2,97	13,84	122,2	1406,78	31 %	32 %	7108



Scylla is the fastest (4.52 times more than Cassandra and 16 times more than MariaDB).

Conclusion

Scylla and Cassandra are much faster than MariaDB/MySQL except for reading and Scylla is faster than Cassandra.

RAM and CPU usage does not really change according to our tests, deleting is a little more heavier on MariaDB.

For queries, we have currently almost the same number of reads and writes. But most of time, reads are simple queries like getting columns for a given ID.
As mentioned above, with Cassandra/Scylla, we must optimize reading operations by reducing and simplifying them at the expense of writing. Writing does not cost much.

We will have better performances for files/folders management (copy/move/create…).

On the infra-scripts (cron) side, performances are not very important as it does not impact the user. The most important is saving time on uploading/downloading phase.

File upload
incrementSizeStored method is called at every chunk, methods called at file creation or file completed are not significant. So, we will get better performances (write operation – UPDATE).

File download
getUploadFolderPath method is called, then, the file is read on the disk. Only the path is from the database but we use a cache with Redis, so it will not impact performances.

We need to pay attention to different read operations, especially about getting a folder because they are often read and refreshed.

Scylla also allows us to have a better scalability and a high availability with no single point of failure.

Scylla data modeling

We had to structure our data in order to keep best performances, so, it is different compared to relational databases. New features such as comments, contacts, share with… have been added too.

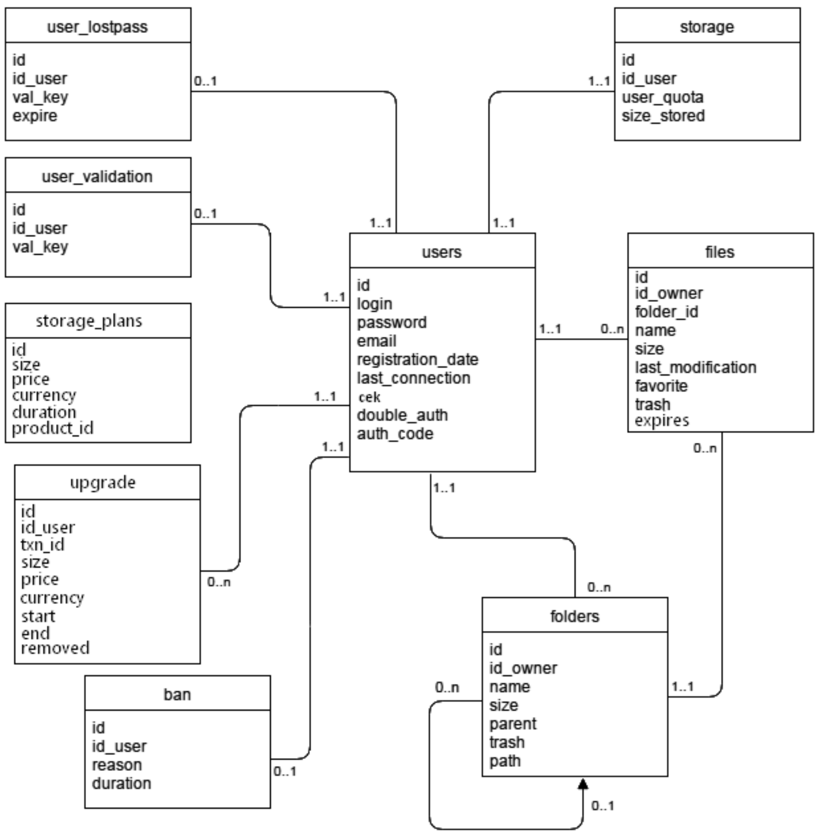


Figure 2 – MariaDB data modeling used at Muonium before migration.

Note : since we implemented ga before migrating, there is also ga_salt and a table for storing ga backup codes.

Primary Keys

http://intellidzine.blogspot.fr/2014/01/cassandra-data-modelling-primary-keys.html

This article is very interesting about primary keys. The concept is different in Cassandra/Scylla and we had to think about its utilisation. So, we will talk below about compound primary key, partition key, cluster key.

Modeling

The lack of flexibility and constraints inherent to Cassandra/Scylla have caused us difficulties for establishing a functional data structure ; indeed, we can note that :
- CQL does not support aggregation queries like max, min, avg
- CQL does not support group by, having queries.
- CQL does not support joins. separate
- CQL does not support OR queries.
- CQL does not support wildcard queries.
- CQL does not support Union, Intersection queries.
- Table columns cannot be filtered without creating the index.
- Greater than (>) and less than (<) query is only supported on clustering column.

as well as a reduced number of functions...

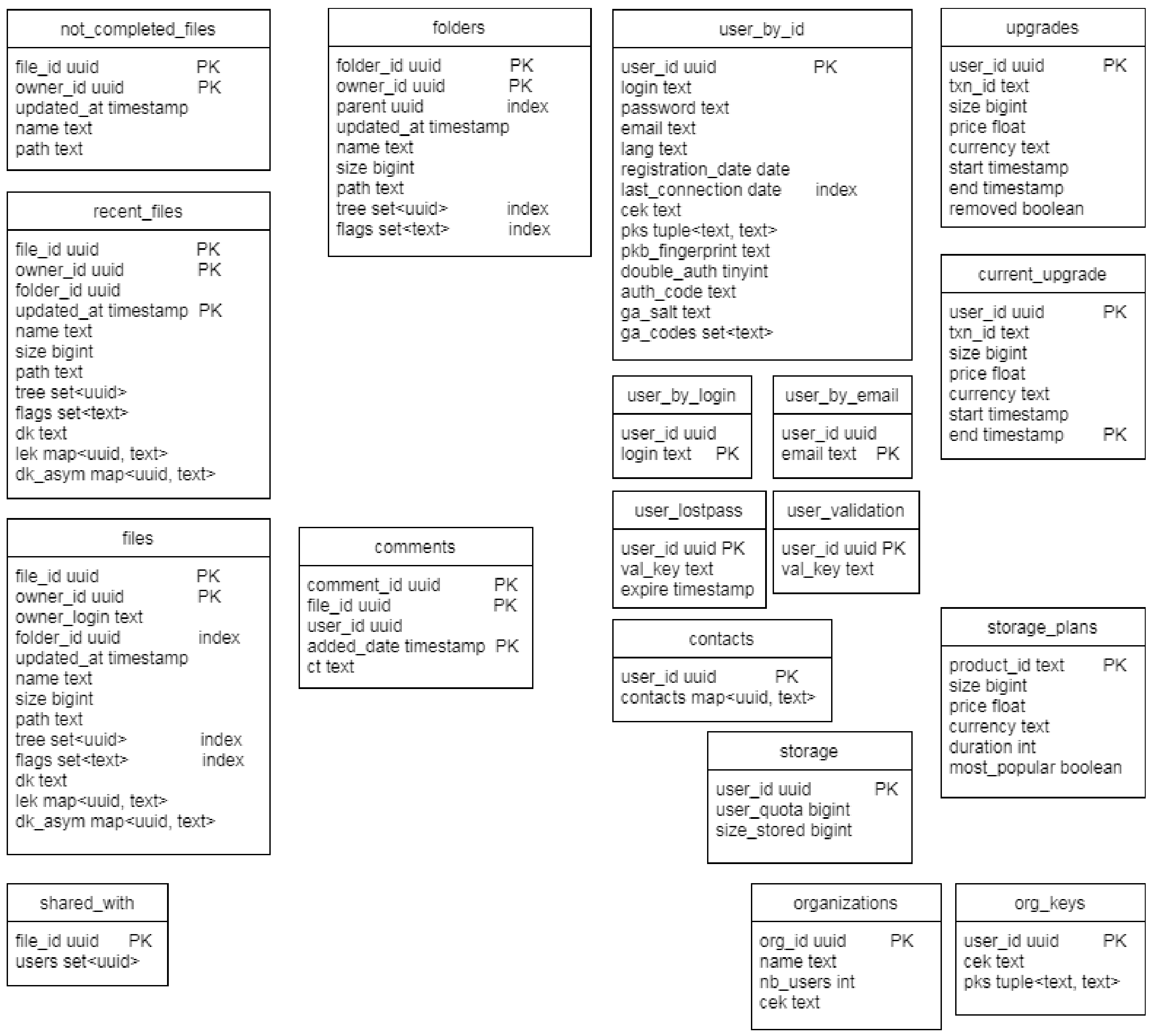


Figure 3 – Scylla data modeling

So, we analized every query needed and how this could be organized considering constraints, what is selected and what is passed in the WHERE clause.

Foremost, we could put *user_lostpass*, *user_validation* and *storage* into users as they involve users and a single row per user ; but we decided to keep them separed because we do not use them at the same time and they have a different application.
They all have a simple primary key (user_id).

ID’s type will be uuid on Scylla and timestamp for dates. Tinyint(1) columns will be stored as boolean type.

For users table (*user_by_id*), we added « pks » which is used to store public and private asymmetric keys as well as pkb_fingerprint and ga_codes which contains backup codes (instead of storing them on another table). There is an index on last_connection column in order to query users by last_connection (for deleting inactive accounts).
user_by_login and *user_by_email* are used only for retrieving user id by login or email.

This user can upgrade his storage (*current_upgrade*). Former upgrades must be stored, so, all upgrades (removed or not) are stored on *upgrades* table. There is a distinction due to the fact that otherwise, removed should have been indexed or set as PK and we don't have the same utilisation with valid or expired. Indexing a boolean type is not recommended. end is the clustering key because we need to filter them if we want to remove expired upgrades. When a upgrade is renewed, row is deleted and inserted again with new expiration date. end PK for *upgrades* is not necessary because we don't need to filter from this table.

Files and folders are a little similar.
Concerning files ; owner_login (used at file sharing) and path are stored instead doing joins to get them.
« lek » and « dk_asym » are keys used for file comments and internally shared files.

When a file/folder is moved or renamed, we need to update path for children and this path is stored as text, so, a LIKE is used (+ other functions) and we can't do the same with Scylla.
We found a solution : store tree as a collection of folders ID in order to select files and folders with folder ID requested in the collection. tree is indexed.
Once files and folders selected, we just need replacing text path and tree. Updates are fast with Scylla.
flags is a collection for settings file/folder as favorite or trashed and is indexed. It allows us to not create multiple tables and complex process for managing favorites/trash and offers flexibility if we need a new flag. It can be indexed because this is a low cardinality column but not a boolean.
folder_id/parent are also indexed to get children for a given folder.

We have still some issues with files : we need to get recent files for recent feature and select not completed files in order to delete them.

For the first issue, we created a materialized view (*recent_files*) which is sorted by updated_at column.

For not completed files, we created a table *(not_completed_files)*, a row is inserted when a file is created and removed only at the end of uploding process. So, we keep only currently uploading files and not completed. updated_at is stored for filtering currently uploading files/recent files which must not be removed.