LumoSQL Single Motivation v1.0

Dan Shearer

17th May 2022

Abstract

This discussion document is for people who know at least roughly what LumoSQL is, or who have seen the LumoSQL talks. It presents a unifying theme that links the different inventions, to make it obvious what LumoSQL is for and what problem it solves. *LumoSQL is about rules enforced by data guarantees.* The LumoSQL inventions combine to move crucial decisionmaking power away from unreliable code and into data structures. Any application software handling LumoSQL-compatible data must abide by the rules and guarantees of the LumoSQL data structures. Software applications often remove power and autonomy from end users, and LumoSQL gives some of that power back. Some of these data guarantees are mathematical, others are institutional, and others are happy side-effects of unusual software patterns in SQLite and LumoSQL. It is through these guarantees that LumoSQL addresses some of the biggest unsolved problems in software.

Contents

Figure 1: LumoSQL assumes software development can never be relied on and is getting worse. (c) Randall Munroe

This cartoon is no more ridiculous than the existence of more than a million browser User Agent strings for at most a few hundred browsers. Web sites and browser developers spend a fortune reacting to each other's changes in User Agent string handling, emulating each other's *already faked* strings which *already impersonate each other*. There is no technical reason for this problem, and no way to fix it without fundamental change.

LumoSQL enforces rules by data guarantees.

Normally it is the algorithms in software that enforce rules, for example *You do not have permission to read salary information for that person.* That rule is written in just one program, written by just one developer. What if the developer made a mistake? What if it is your own salary information but the algorithm doesn't know that? This is how everyday problems occur that make people say "computers are just so stupid".

With LumoSQL, that same salary data is encrypted with several cryptographic keys. One of those keys will belong to the person whose salary it is, so they can always read it. Another one will belong to the HR group in the company, so anyone who has a key that belongs to that group can read the data and update it. And another read-only key may be issued to the tax department, and that key stops working as soon as there is a change to the salary data.

LumoSQL Addresses unsolved problems

LumoSQL is a single approach to three very big unsolved problems in software.

Complexity: The logic of society is encoded in software too complex to be understood in detail by humans.[1] Once a development team has committed to complex technologies there is often no option to reverse that decison. When there are problems, the tendancy is to abstract, extend, wrap. The intention is to simplify software development but it has the opposite effect, creating a Big Ball of Mud[2]. As an extreme example, the API company Postman[3] encourages Big Balls of Mud by curating over a million APIs for reuse and forking, claiming in their attractive graphic novel that they are solving problems of complexity. *Unlike the worsening state of software, a data structure does not inherently become more complex or fragile over time.*

Disempowered users: Software is often used for matters of personal autonomy such as banking, health, private conversations or interactions with government. No solution exists for software to guarantee it will respect the rights and wishes of users. Even in 2022 software cannot give any guarantees stronger than a message on the screen to say "we promise to look after your data and obey the law"[4]. This is often not a credible statement due to the power of organisations concerned, yet users have no alternatives. *LumoSQL guarantees that data owners*

[1] Examples include the thousands of constantly-changing cloud APIs and Node.js' millions of packages, each with maximum runtime privileges.

[2] A paper called Big Ball of Mud summarised how bad complex architectures are in 1999, and the same principle applies today

[3] recently valued at 5 billion USD

[4] Very new companies such as Privado offer automated code scanning to companies from a privacy point of view, but the end user is still required to believe that their supplier has used Privado, that Privado's AI is effective, and that their supplier has faithfully implemented everything Privado advised.

have control over access, visibility and portability. These are fundamental human rights.

Developer knowledge loss: Since approximately 1980 each generation of software developers generally understands less about computing fundamentals than the previous generation, especially on topics of efficiency and reliability. Some open source communities focus on efficient coding, but this is not mainstream or typically taught to students. After just 10 or even 5 years, most end user applications become difficult to understand and often hard to run at all due to changes in dependencies. *LumoSQL data structures are closely related to mathematics. Mathematics has transmitted knowledge across generations for 4,000 years. We can expect the mathematics of LumoSQL data to be understood in detail for much longer than 10 years.*

There are a few exceptions to the general observation that software is getting worse. All is not lost!

LumoSQL: Enforcing rules by guarantee

Data structures can enforce rules in ways software cannot. Take for example your personal chat app - who makes sure the internal logic of the app is correct through its versions and as Android/iOS make changes underneath it? We can't rely on the chat app company, because the company's interests and priorities are very different to yours. The app's logic determine how you interact with your friends, what other third parties can see your chat data, and other things that might become unexpectedly vital to you at any time. And what about your chat data backups? On one hand you want backups to be secure from prying eyes, but on the other unreadable backups are not backups at all.

The LumoSQL approach turns this upside down. We introduce a new standard secure format, which is only readable if the the chat software obeys the mathematical rules describing the format. Any LumoSQL-compatible app[5] can tell if the chat data is corrupt or not, because that is a mathematical property unrelated to the content. You, the user of the chat app, are no longer reliant on the promises of software developers working for eg WhatsApp or Telegram to protect your rights. Any one of hundreds of apps can check that your data is valid. In addition you might authorise a few of those apps to read your chat data, so you can access your history without involving any code from WhatsApp/Telegram/whoever. But if the data on your phone is copied by criminals or the police, they cannot read that data unless you give them a key... and even then, you may choose to only give the police a read-only key so you can be sure no changes have been sneakily made without your knowledge.

Software source code is never a fixed standard[6]. Unlike software, data layouts and structures are frequently fixed standards for decades. Advanced mathematics like checksums, signatures and keys have been used in data structures for decades, and now LumoSQL is adding multi-user access controls using zero-knowledge proofs.

[5] A LumoSQL-compatible app is one that does one or both of (a) understand the Lumions data format (b) uses the LumoSQL embedded database library, in a similar way that the SQLite database library is used.

[6] the TeX program is perhaps the only exception, with just a tiny handful changes in the last 30 years despite having millions of demanding 21st-century users. If you are reading this document as a PDF, it was generated using the 1992 version of TeX.

What are the guarantees?

Some of the following guarantees may initially seem outrageous or even impossible. The reason they work is that they are not all related to computer science, but arise from the collective total of the influences around LumoSQL. These influences are explained in the next section.

1. **No silent corruption**: If data has been corrupted, accidentally or deliberately, then the data becomes obviously invalid.
2. **No unauthorised updates**: If data is valid, then it has only been updated by authorised humans or computers.
3. **No unauthorised reads**: If data is valid, then only humans or computers with a valid read key can access the data. If no key is available, then the data is not recoverable.
4. **Futureproof**: Data will retain its protections regardless of future changes in operating systems, computer languages, hardware or software. This is the most multi-factor of the LumoSQL guarantees.
5. **LumoSQL data is always distinguishable**: Regardless of any other factor, valid LumoSQL data cannot be mistaken for any other kind of data.
6. **One LumoSQL data item is always uniquely identifiable**: LumoSQL data has an ID unique among all data of all kinds, which makes it difficult to claim that LumoSQL data has been lost.
7. **Non-repudiable**: If a particular author has created data, then there is no way for some other author to claim they have created the data. Even if the original author (or program) has subsequently lost their identity keys, all work produced by that author remains uniquely identifiable. (This guarantee also means that LumoSQL cannot deliver strong anonymity.)
8. **Data with the same ID will be identical**: This follows from the previous guarantee, and means "no duplicate IDs ever" although there may be many copies of data.
9. **The unique ID includes a version number**: Every write action increments a version number (there is no guarantee that the data content from a previous version is kept.)
10. **Left-over data remains secure**: Any left-overs after changes to LumoSQL data will be still be encrypted and therefore no less secure than they ever were. A practical fact of databases is that pages of obsolete data are often left on disk after changes.

No guarantee in life is absolute, including ones based on mathematics. But the LumoSQL guarantees are stronger than anything that software can offer alone. Data is where enforcement of users rights should begin, not the law and certainly not software.

What influences on LumoSQL make these guarantees possible?

The first unusual factor is the scale of SQLite. SQLite is essential to billions of people, millions of developers and thousands of companies. If any addon to SQLite appeals to even a tiny fraction of the SQLite userbase, that is what would be massive adoption for any other software.

The scale of SQLite also drives its conservatism, which is what attracts users such as Airbus. SQLite carefully avoids breaking changes because at SQLite scale they could be catastrophic. With twenty years of successfully avoiding mass breakage, SQLite's conservatism rises to a pretty strong guarantee. LumoSQL inherits this stability and keeps that same mentality while introducing optional breaking changes.

Institutional and criminal interests care about SQLite because it is overwhelmingly where the most critical personal data is held outside cloud computing. And it is the scale of SQLite which meant LumoSQL would never be successful if it forked SQLite, which lead to the invention of Not Forking. All of these things and more feed into each other, resulting in LumoSQL being able to promise both great stability and also never-before-seen security innovations.

- Institutional factors
  - US Library of Congress selected the sqlite3 file format as an official archive format, with one subformat. LumoSQL is likely to add another subformat for hidden columns.
  - UK National Archives also chose sqlite3 as a preservation format, and others around the world.
  - EU Parliament and selected other legislatures worldwide have passed laws which mandate the mathematics of privacy. LumoSQL aims to at least meet EU privacy requirements.
  - Airbus have asked for binary compatibility until 2050. This is not unusual for embedded aerospace applications, and there are likely many other companies with similar requirements.
  - Security services and policing often harvest plain text sqlite3 files (including deleted pages) to harm end users, and LumoSQL encryption will stop that. However the same institutions are likely to be very happy with LumoSQL's UUID which will allow them to track data flows between devices and clouds, even if they can't read the contents, and even though LumoSQL explicitly does not offer anonymity features.
- Mathematical factors
  - UUID hash - this is a constant method used to calculate the unique LumoSQL IDs, and will not change except and until some cryptographic breakthrough means that hash collisions become feasible with e.g. SHA3-256.
  - Signatures and checksums

  - Attribute based encryption
- Non-technical aspects of SQLite
  - Social influences and human habits
  - Commercial platforms (Android, Apple, Samsung etc)
  - Criminal interests
  - Project management
- Science aspects of SQLite and LumoSQL
  - Lumion specification and standards process
  - Source tree management
  - Software development: APIs, internal churn, testing
  - Statistics
  - Digital forensics
  - Not-forking
  - Small size of the SQLite codebase
  - Even smaller size of LumoSQL non-forked additions to SQLite

Note: Another set of technical facts which cut across all entries in this list. These include shortcomings in the current sqlite3 format, the nature of metadata in LumoSQL, and the varying levels of encryption in LumoSQL (row, table, whole-file). There is also the question of compression, solved by SQLite via the sqlar format and likely within LumoSQL by per-row dictionary compression. The items mentioned in this paragraph can be thought of as orthogonal to all of the factors in the list above.

Does the LumoSQL project have the right people?

LumoSQL can't avoid the disasterous software mistakes listed above unless we have skills and interests that are not conventional or mainstream.

So far, LumoSQL has team members who are, variously:

- **Interested in reducing complexity**: We have expertise in creating simple APIs, the Rust language, Nix and reproducibility, minimalist OS design, and OS-less computers. This range of interests helps us avoid classic software complexity traps.
- **Promote inter-generational knowledge transfer**: We have younger people, interest and expertise in the topics of equality and human rights, and a strong interest in formalised knowledge in a university context. These hopefully combine to promote an environment where younger people can apply and extend what LumoSQL is pioneering today.
- **Want to measure our work**: A feature of technological collapse is lack of knowledge about complexity. Our statistical approach aims to quantify our measurements, which may allow us to answer questions about complexity or at least see what the questions should be. How much overhead do the new security arrangements add? How do different cryptographic

solutions compare? How many SQL operations are there where the crypto substantially improves access speed?

This seems a very good start, and we can be proud of ourselves. We need more people, so if you are reading this and are intruiged do please come and say hello in #lumosql on libera.chat, or contact authors@lumosql.org.

Appendix A - Related topics

It would take months to survey the relevant literature, but I think the following are the main topics.

Collapse of Technology Systems

Papers on this general topic tend to focus on failures that are not dramatic, analysing complexity, lines of code, etc. There are vastly more papers promoting increased complexity than there are on analysing the reasons why software is getting worse. There are individual practical solutions to create systems that do not have legacy complexity, however I am not aware of them being part of a movement to do so.

Sometimes what seems a technology collapse is not. For example tragic failures of technology systems causing air crashes are not primarily about declining software quality, or overdependence on software. Humans brains can no more handle the processing demands of flying a modern jet than they were of landing on the moon in 1969. A Boeing software error caused the deaths of 346 people. But all modern jets without exception use fly by wire just like the Lunar Landing Module did half a century ago, and that is a step forwards not backwards.

Concurrent Systemic Declines

It seems like there are multiple collapses happening at once. Many papers since 1980 or so document decline, diminishing returns or rapid collapse in:

- scientific discoveries (there is constant progress but in most areas it is incrementalism. This is an entire field of study by itself.)
- technological innovation. While there is also constant progress, and one or two outstanding counter-examples, researchers in the field often comment that there are diminishing returns in innovation, and that many new systems with enhanced features do not offer anything new.
- ecosystems (i.e. the IPCC reports use the word 'collapse'.)
- democratic processes and liberties (if you are struggling to see declines in this area worldwide, there is nothing I can say that will help, but there are specialists whose work you can read on the state of journalistic freedoms, potentials for violent conflict, rising intolerance etc.)

In 2019 Jonathan Blow presenting in Russia almost summarised the LumoSQL raison d'être under the topic "Preventing the Fall of Civilisation". This URL is offset 18 minutes to where discussion of software starts, but if you are interested in the Lycurgus Cup then watch the whole thing.

Despite all the bad news, personally I have found specific reasons to feel more positive about the future of humanity than I have for a long time. And LumoSQL has the opportunity to make a signficant difference.

ENDS