Honestly, while I understand the importance of each, I personally do prefer MongoDB as a database. Idk if it cause i'm already used to Javascript's and JSON's object format but I value a lot the way I can isolate each user object away from each other instead of placing all users in the same table and apply either array properties and/or object properties to expand on each without making it look like everything is in the same soup of rows. Not to mention that whenever you make an request to an api the expected format is usually not a table but an object.

I also really like how not all have to be the same, even while using schema structure. Structured but not solidified- because in life reality is that nothing is entirely set in stone and too many extra NULLS can be a bit obnoxious when maybe Only an specific user in mention might need an extra column.

I just don't personally think it is visibly appealing to have so many otherwise unnecessary NULLs. And when I think about storage I kind of imagine it user-oriented, when I pull all information about a user I don’t want to sit down and manually select column1, column1, column1¸ column1… column1 FROM table JOIN table2 WHERE username= “blue” every single time I want all of the information of a user, nor have unnecessary columns in the outcome if “blue” is NULL for those column fields.

Like the avengers database- how many of them died and reincarnated 4 and 5 times? Not many. So the full table is more NULL than anything else.

And For Example 1 at the end.

Hence, I think that is really one of the biggest flaws of relational databases.

Most importantly, I don't want response time to lag, not even for a second, when choosing from who knows how many users just because there is too much information to query through at once.

What the comment said, or what the most recent post is about once the button is clicked on- it will be a con if it has to select from the column of 1000posts that the user has created just to take Only the last one.

Which is what SQL's order of operation does:

FROM and JOIN (join is kind of part of FROM's logic) run first in logic and collect all of the data, Then select filters the specific columns, then WHERE the specific rows and after most of the process of defining the query happens **Then** it runs LIMIT to limit it to only the one that was at the top of the left results.

If i just want the very last object in an array is as easy as array[array.length-1] in almost a “split-second”.

One of the things I consider within the most, if not the most, important about websites now-a-days is speed. There are endless statistics out there already- you don't want your user to wait even an extra second if you can avoid it. The longer your page takes to load the most likely is the user to move on to a faster UI. Which is also why is not a bad idea to avoid big heavy images and the main purpose of the modern minimalistic approach. The simplest the UI the least extra content to load and the faster the overall application runs.

This is by far the one I consider to be the most important difference from relational to non-relational although it is undermined sometimes in the posts comparing one and the other, but here is explained well how NoSQL is faster: <https://en.wikipedia.org/wiki/NoSQL>

On the other hand, tables are not known for being fast. Tables are known and were created for storing massive amounts of data in the least amount of space possible.

They are indeed the best way to store and manage humongous amounts of data- no doubt on that. But the only way they achieve that is through being strict. Which is why SQL saves information in tables. Is a lot of similar elements (rows) organized per categories that they all share (columns).

And that is why I would much rather use tables for logging history than the actual content of an application.

Things like username and profile picture for example, are not things I want to be set in stone; while certainly something i'd want to save in stone is when the password was changed- or username or email was changed per user token, Not user id. That kind of important backlog information that would seem overwhelming to the front-end oriented perspective is what I would want to keep in the relational database, because I know is not pertinent to require any flexibility or pertinent reaction any time soon: User account created date - user changed username date - user posted posts: as id of posts created in column- user commented on post with id blank: column with ids of posts the user commented- basically to keep track of user activity in general. Who What Where and When – with no whys or hows.

So there’s example 2.

Example 2: If I want a to keep track of a log of all transactions on a bank account- that is another story. In a formal transaction neither FROM nor TO will ever be NULL, nor will transaction date or amount. This is where tables shine most where NULL values per column are less. And when any flexibility not only is NOT required **but by all means should be avoided.**

For example 1:

Maybe all my users up to this point had one first name and one last name. One column for first name, another for last name.

But then, all of a sudden, two of our users notify us that they have a middle name or we just notice they entered first and middle name in the same field. Once again, not everyone has a middle name. Why would you want to give everyone a column of middle name just to make it NULL when only specific cases have a field of middle name? people don’t just grow a middle name from one day to the next, hence that NULL cell is likely to never change unless the user leaves for whatever reason. And if this user leaves- and is the only one with a middle name, then you’ll have a full column of NULL because no one else has a middle name and unless a new person who joins has one- this column will remain useless while waiting for it. Stealing storage and processing capacity while it does.

But that’s not it- Then I had a new user who came from another country and has not one but 2 last names, and I decide I want to include the room for this user's last name to be properly recognized and saved as two lastnames instead of one.

With mongoose in this case I can easily just analyze the entered data with an array method(maybe split will do) and IF there is a second last name divided by a sign of - or a space in the lastname input field Then to add an extra object property accordingly without having to have all other users who don't have more than one name to stand with an unnecessary field.