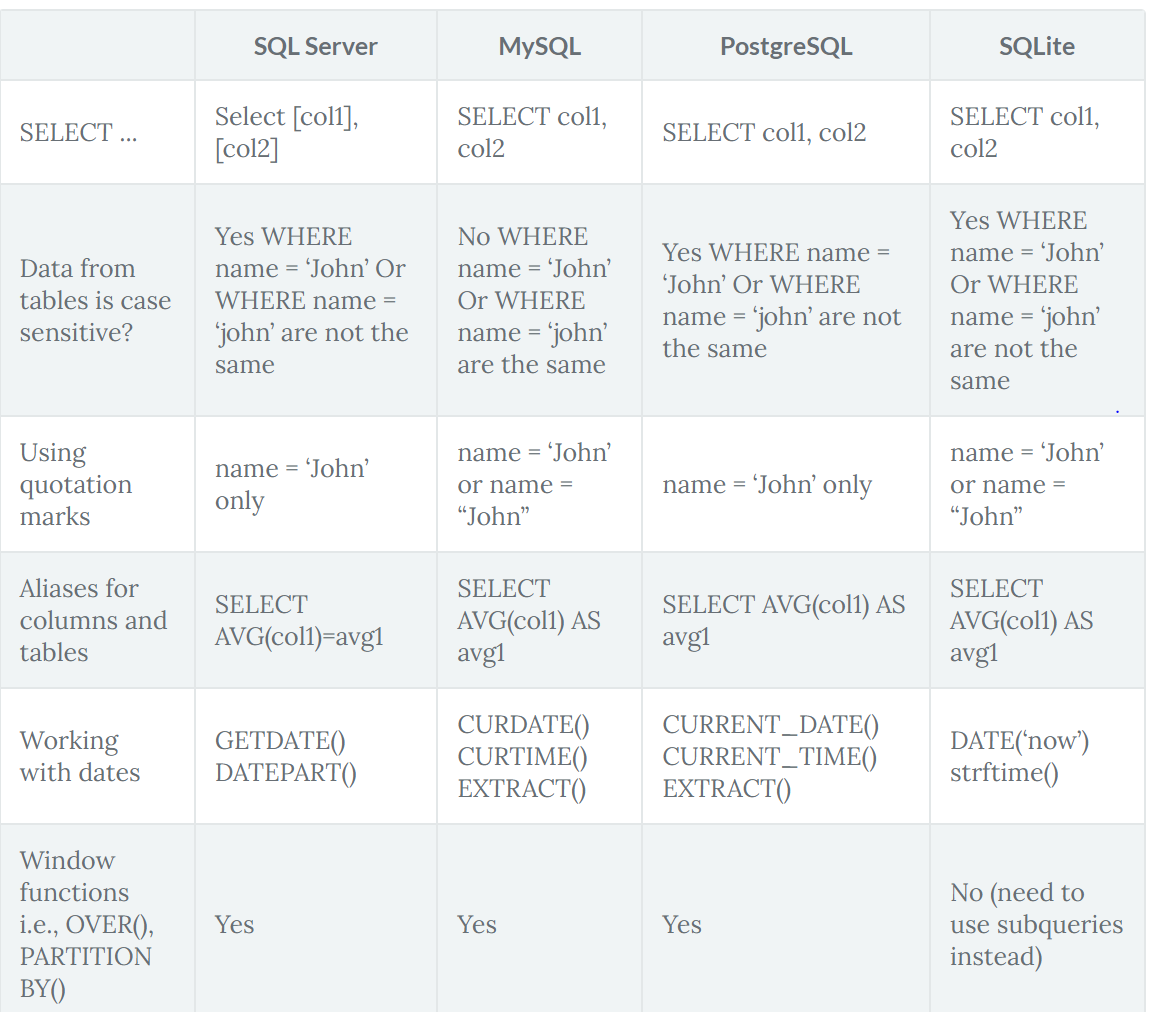
**MySQL** has consistently been the most popular version of SQL in Stack Overflow questions. Second in line is **Microsoft SQL Server** (including T-SQL, the name of Microsoft’s dialect of SQL), which remains a consistently more popular tag than **PostgreSQL** and **SQLite**. This means that if you have a question specific to one of these systems, you’re more likely to find that someone already asked your question.

**What’s the difference?**

PostgreSQL, MySQL, and SQLite use very similar syntax, with some notable differences highlighted below. Microsoft SQL Server has the greatest contrast in SQL syntax, as well as a wide variety of functions not available in other platforms. The table below highlights some examples of basic differences between SQL platforms.

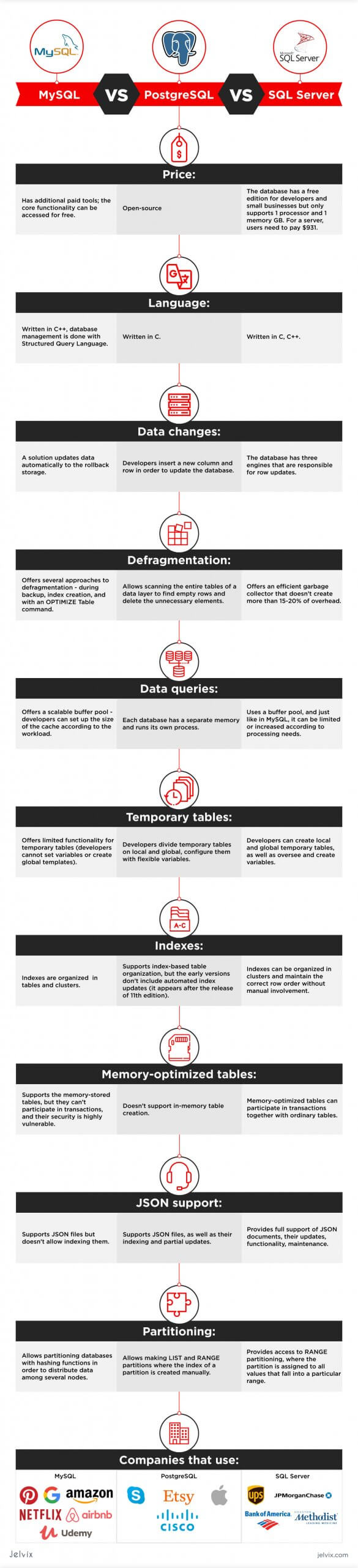


**MySQL** offers a scalable buffer pool – developers can set up the size of the cache according to the workload. If the goal is to save CPU and storage space, developers can put strict benchmarks on their buffer pool. Moreover, MySQL allows dividing cache by segments to store different data types and maximize isolation.

**PostgreSQL** isolates processes even further than MySQL by treating them as a separate OS process. Each database has a [separate memory](https://www.postgresql.org/docs/12/runtime-config-resource.html" \t "_blank) and runs its own process. On the one hand, management and monitoring become a lot easier, but on the other, scaling multiple databases takes a lot of time and computing resources.

**SQL Server** also uses a [buffer pool](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/buffer-pool-extension?view=sql-server-ver15), and just like in MySQL, it can be limited or increased according to processing needs. All the work is done in a single pool, with no multiple pages, like in Postgresql.

If your priority is to save computing resources and storage, choose flexible solutions: the choice will be between MySQL vs SQL Server. However, if you prefer clear organization and long-term order, Postgre, with its isolated approach, might be a better fit.



## JSON Support

The use of JSON files allows developers to store non-numeric data and achieve faster performance. [JSON documents](https://en.wikipedia.org/wiki/JSON#:~:text=JavaScript%20Object%20Notation%20(JSON%2C%20pronounced,or%20any%20other%20serializable%20value).) don’t have to be parsed, which contributes to much higher processing speed. They are easily readable and accessible, which is why JSON support simplifies maintenance. JSON files are mostly used in [non-relational databases](https://jelvix.com/blog/relational-vs-non-relational-database), but lately, SQL solutions have supported this format as well.

**MySQL**supports JSON files but doesn’t allow indexing them. Overall, the functionality for JSON files in MySQL is very limited, and developers mostly prefer using classical strings. Similarly to non-relational databases, MySQL also allows working with geospatial data, although handling it isn’t quite as intuitive.

**Postgresql**supports JSON files, as well as their indexing and partial updates. The database supports even more additional data than MySQL. Users can upload user-defined types, geospatial data, create multi-dimensional arrays, and a lot more.

**SQL Server**also provides full support of JSON documents, their updates, functionality, and maintenance. It has a lot of additional features for GPS data, user-defined types, hierarchical information, etc.

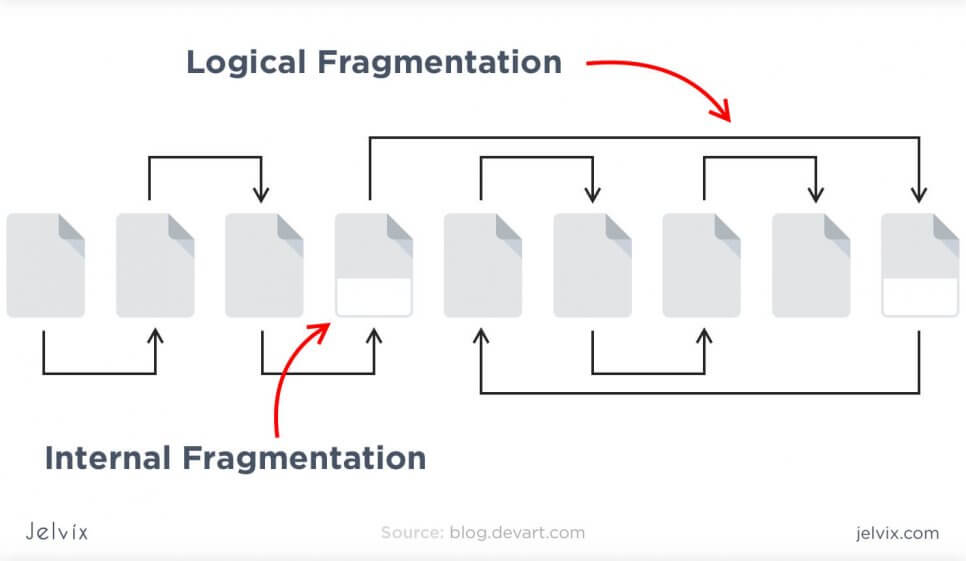
Overall, all three solutions are pretty universal and offer a lot of functionality for non-standard data types. MySQL, however, puts multiple limitations for JSON files, but other than that, it’s highly compatible with advanced data.

### Defragmentation

When developers update different parts of an SQL database, the changes occur at different points of the systems and can be hard to read, track, and manage. Therefore, maintenance should include defragmentation – the process of unifying the updated database by assigning indexes, revisiting the structure, and creating new pages. The database frees up the disk space that is not used properly so that a database can run faster.

**MySQL** offers several approaches to defragmentation – during backup, index creation, and with an [OPTIMIZE Table](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html) command. Without going into much detail, we’ll just say that having that many options for table maintenance is convenient for developers, and it surely saves a lot of time.

**PostgreSQL** allows scanning the entire tables of a [data layer](https://jelvix.com/blog/guide-to-web-application-architecture) to find empty rows and delete the unnecessary elements. By doing so, the system frees up the disk space. However, the method requires a lot of CPU and can affect the application’s performance.

[](https://jelvix.com/wp-content/uploads/2020/07/04.jpg)

**SQL Server** offers an [efficient garbage collector](https://sqlserver-help.com/tag/garbage-collection/) that doesn’t create more than 15-20% of overhead. Technically, developers can even run garbage collector on a continuous basis, because it’s that efficient.

Overall, MySQL and SQL Server offer more of defragmentation methods that Postgresql does. They consume less CPU and provide more flexible settings.

## Data Queries

Here, we take a look at how the systems cache and process user requests, what approaches they take in storing data, and how developers can manage it.

### Buffer Pool

Some systems call a buffer to pull cache, but regardless of terminology, our goal is to summarize the algorithms that systems use to process user queries and maintain connections.

**MySQL** offers a scalable buffer pool – developers can set up the size of the cache according to the workload. If the goal is to save CPU and storage space, developers can put strict benchmarks on their buffer pool. Moreover, MySQL allows dividing cache by segments to store different data types and maximize isolation.

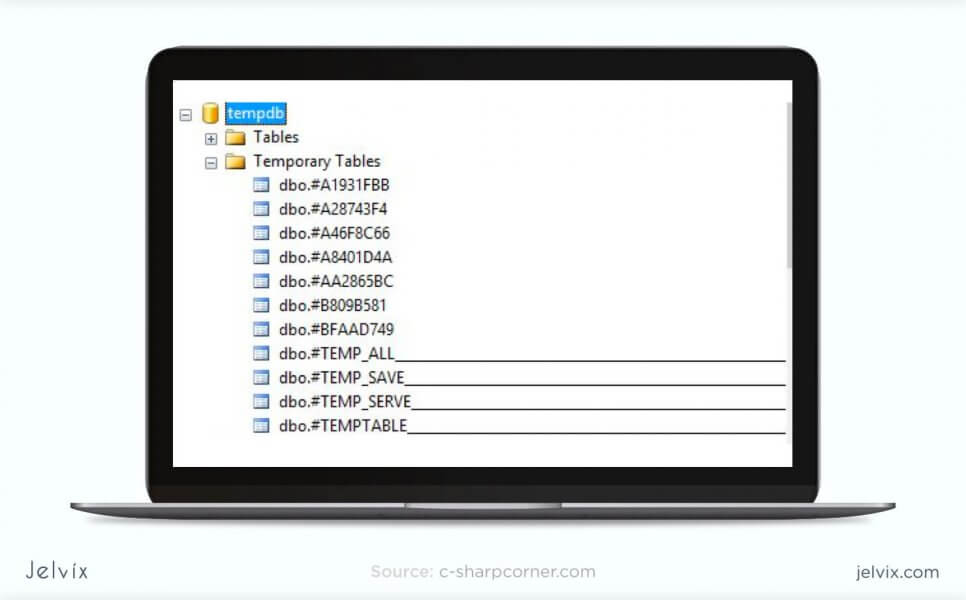
**PostgreSQL** isolates processes even further than MySQL by treating them as a separate OS process. Each database has a [separate memory](https://www.postgresql.org/docs/12/runtime-config-resource.html" \t "_blank) and runs its own process. On the one hand, management and monitoring become a lot easier, but on the other, scaling multiple databases takes a lot of time and computing resources.

**SQL Server** also uses a [buffer pool](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/buffer-pool-extension?view=sql-server-ver15), and just like in MySQL, it can be limited or increased according to processing needs. All the work is done in a single pool, with no multiple pages, like in Postgresql.

If your priority is to save computing resources and storage, choose flexible solutions: the choice will be between MySQL vs SQL Server. However, if you prefer clear organization and long-term order, Postgre, with its isolated approach, might be a better fit.

### Temporary Tables

Temporary tables allow storing intermediate results from complex procedures and branched business logic. If you need some information only to power the next process, it doesn’t make sense to store it in a regular table. Temporary tables improve database performance and organization by separating intermediary data from the essential information.

[](https://jelvix.com/wp-content/uploads/2020/07/05.jpg)

**MySQL** offers limited [functionality for temporary tables](https://dev.mysql.com/doc/refman/8.0/en/create-temporary-table.html" \t "_blank). Developers cannot set variables or create global templates. The software even limits the number of times that a temporary table can be referred to – not more than once.

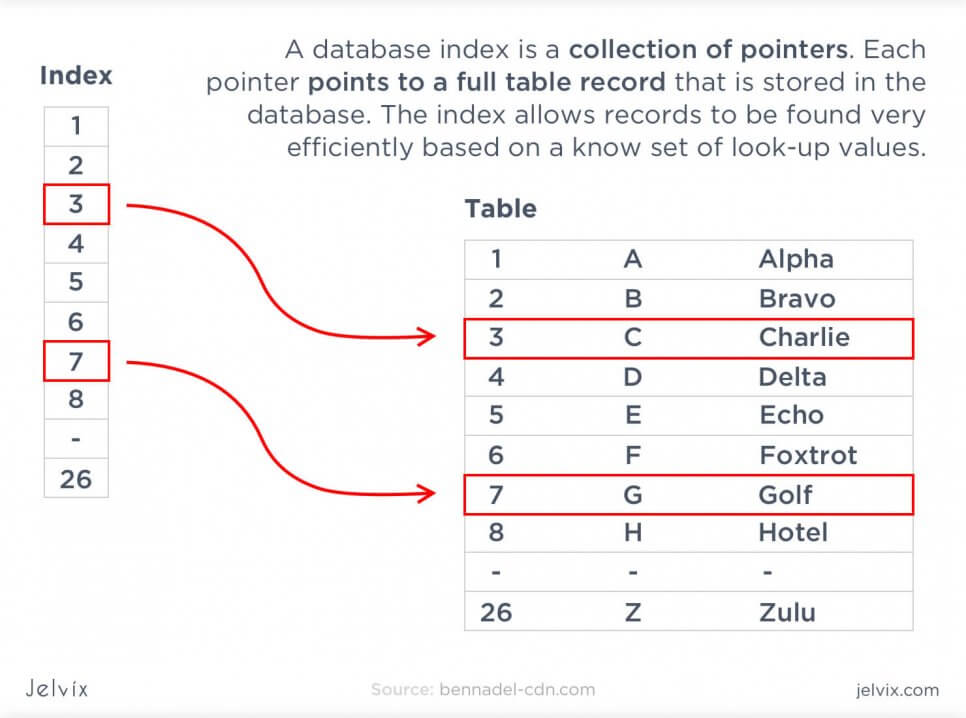
**Postgresql** offers a lot more functionality when it comes to temporary content. You divide temporary tables into local and global and configure them with flexible variables.

**SQL Server** also offers rich functionality for temporary table management. You can create local and global temporary tables, as well as oversee and create variables.

Temporary tables are essential for applications with complicated business logic. If your software runs a lot of complex processes, you will need to store multiple intermediary results. Having rich customization functionality will often be necessary throughout the development process.

## Indexes

The way a database handles indexes is essential because they are used to locate data without searching for a particular row. Indexes can refer to multiple rows and columns. You can assign the same index to files, located in the different places in the database, and collect all these pieces with a single search.

[](https://jelvix.com/wp-content/uploads/2020/07/06.jpg)

In this comparison, we evaluated the way indexes are created in every solution, the support of multiple-index searches, and multi-column indexes, as well as partial ones.

**MySQL** organized indexes in tables and clusters. Developers can automatically locate and update indexes in their databases. The search isn’t highly flexible – you can’t search for multiple indexes in a single query. MySQL supports multi-column indexes, allowing adding up to 16 columns.

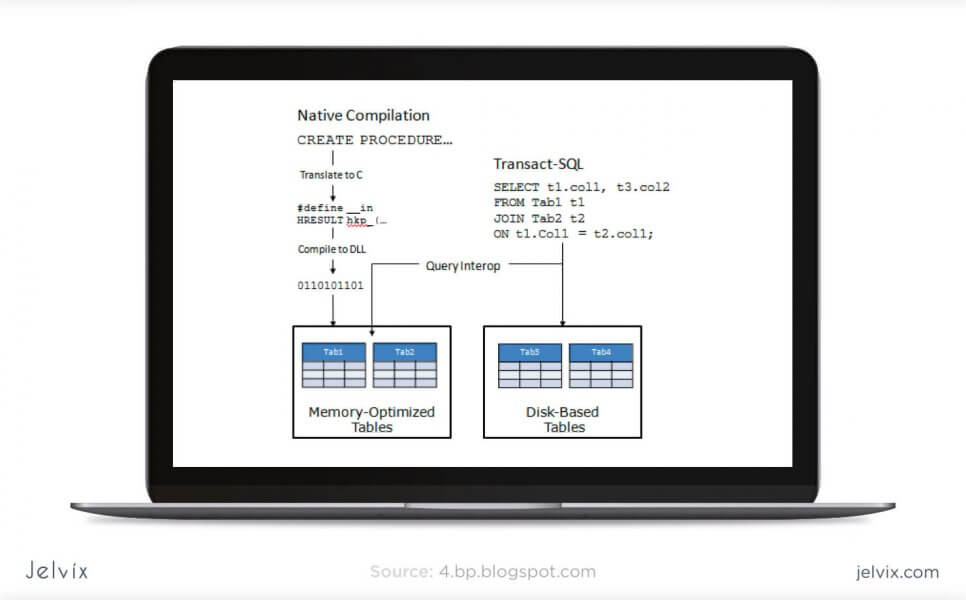
**Postgresql** also supports index-based table organization, but the early versions don’t include automated index updates (which appear only after the 11th edition release). The solution also allows looking up many indexes in a single search, which means that you can find a lot of information. The multi-column settings are also more flexible than in MySQL – developers can include up to 32 columns.

**SQL Server** offers rich automated functionality for index management. They can organize in clusters and maintain the correct row order without manual involvement. The solution also supports multiple-index searches and partial indexes.

Having flexible index settings allows looking up information faster and organizing multiple data simultaneously.

### Memory-Optimized Tables

Memory-optimized tables are mainly known as a [SQL Server concept](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/introduction-to-memory-optimized-tables), but they also exist in other database management solutions. Such a table is stored in active memory and on the disk space in a simplified way. To increase the transaction speed, the application can simply access data directly on the disk, without blocking concurrent transactions. For processes that happen on a regular basis and usually require a lot of time, a memory-optimized table can be a solution to improve database performance.

[](https://jelvix.com/wp-content/uploads/2020/07/07.jpg)

**MySQL**supports the memory-stored table, but it can’t participate in transactions, and its security is highly vulnerable. Such tables are used only for reading purposes and can simplify exclusively primitive operations. For now, MySQL doesn’t come close to making the most out of memory-optimized tables.

**PostgreSQL**doesn’t support in-memory database creation.

**SQL Server**uses an optimistic strategy to handle memory-optimized tables, which means they can participate in transactions along with ordinary tables. Memory-based transactions are faster than regular ones, and this allows a drastic increase in application speed.

As expected, memory-optimized tables are best set up in MySQL – it’s basically their native approach. It’s not an essential database feature, but still, a good way to improve performance.