

FileMaker

Performance Tips

Collected from the 24U newsletter, March 2025 to August 2026

Companion handout · FileMaker Performance Under the Microscope

HOnza Koudelka · EngageU 2026

In this handout

1. Keep the relationship graph lean
2. Watch your server schedules
3. Use conditional formatting sparingly
4. Use FileMaker 2025's JSON speed without breaking compatibility
5. Use indexes wisely
6. Avoid record view privilege calculations
7. Cache your calculation results
8. Use transactions for bulk changes
9. Debug and benchmark custom functions in the Data Viewer
10. Use a server-side script queue

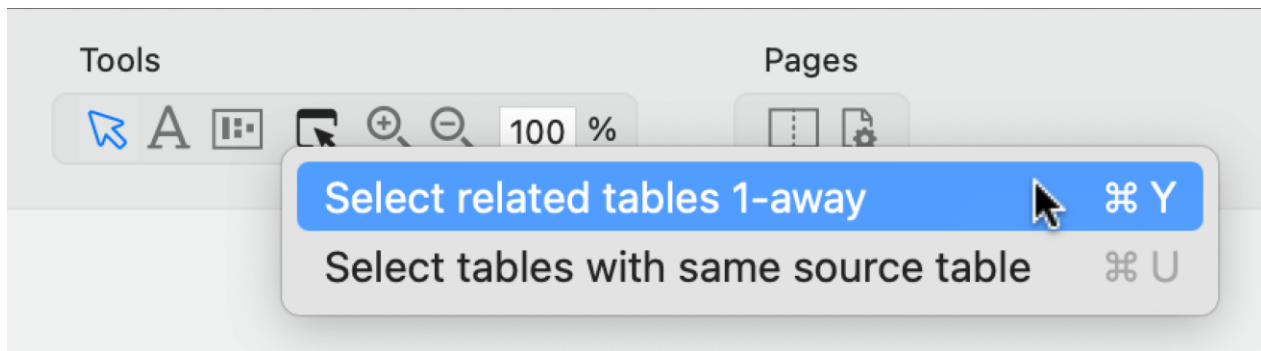
Measure before you change anything.

Every tip here has a case where it makes things worse. Test it against your own solution rather than trusting the general claim, including the ones in this document.

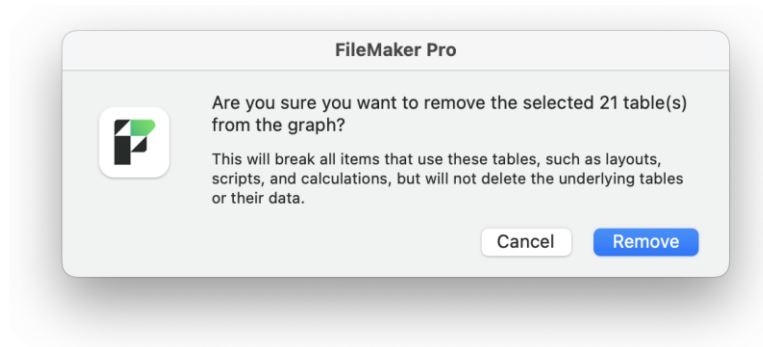
Keep the relationship graph lean

24U newsletter, March 2025

Every new table occurrence carries a performance penalty, and it grows exponentially with the number of occurrences in the same group. This is the thing we can all do something about daily, and the thing we most reliably fail to do.



The size of a table occurrence group is not obvious, which makes it easy to believe one more occurrence cannot matter. There is a simple way to find out exactly how many there are.



Use “Select related tables 1-away” repeatedly until the selection stops growing. Then press backspace, as though you meant to delete the whole group. FileMaker asks whether you are sure, and the warning tells you precisely how many table occurrences you would remove. Note the number, then click Cancel.

If it is high, say over 100, consider a separate group for whatever you are building. Few layouts or scripts need more than a hundred relationships, but one group often gets reused for everything. It should not be. If you are unsure how to move context between groups, look at the Teleport example.

Watch your server schedules

24U newsletter, April 2025

This one comes from a server I look after personally. Our external SSD degraded to a write speed of 5 MB/s, and backup schedules started taking far longer than they used to.

FileMaker Server can warn you when a scheduled task exceeds a set time, and you should always turn that on. A schedule running longer than expected is never good news. The warning goes to Event.log, and you can be notified through Server's own email notifications or a log watcher such as our open-source LogMonitor.

Enter a parameter (optional)	☒ Timeout (minutes)
<input type="text" value="Parameter"/>	<input type="text" value="5"/> ☒ End at timeout

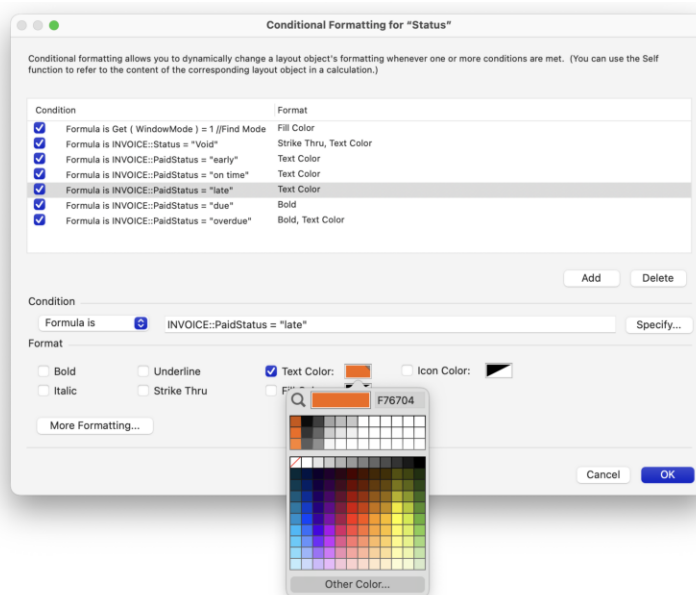
The gap: Server does not let you set a timeout for Verify and Backup schedules, so those need another way of being watched. Worth the effort, because a schedule that overruns hurts server performance directly as well as pointing at some other problem, a failing disk for instance.

The symptom to watch for is schedules that never used to overlap starting to overlap, with the next one beginning before the previous has finished.

Use conditional formatting sparingly

24U newsletter, May 2025

A client asked: I read that calculated fields are faster than conditional formatting. Why?



Because FileMaker evaluates every condition, every time it renders the formatted object. A Case function in a calculated field exits as soon as it reaches the first condition that is true.

So calculate the text style and colour in a single calculation using TextColor and TextStyleAdd. Storing the referenced result in a local variable cuts how often it is re-evaluated, and Case cuts how many conditions are evaluated at all.

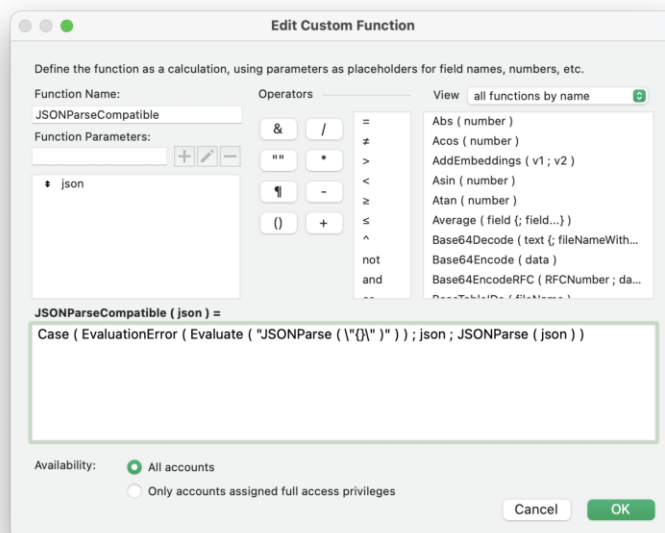
Conditional fill colour is more awkward, because there is no function for it. One old approach, predating conditional formatting entirely, is a calculation with a container result type. If performance is the priority, though, it is hard to beat a coloured string of solid-block Unicode characters.

Use FileMaker 2025's JSON speed without breaking compatibility

24U newsletter, August 2025

FileMaker 2025 brought a large improvement to working with JSON. But what if you cannot make every client upgrade yet? Two ways to benefit without breaking older versions.

Wrap JSONParse in a custom function. Used directly, it makes the calculation invalid in earlier versions. Inside a custom function you can test whether JSONParse exists and only call it when it does.



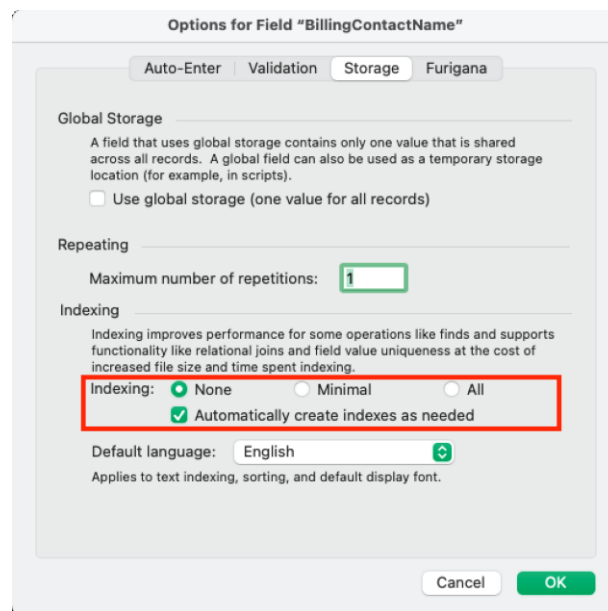
Or do not call it at all. Every function returning JSON keeps the parsed binary form automatically, for as long as it stays in memory. Set it to a variable, local or global; pass it to a script as a parameter; return it as a result. Just avoid writing it to a field or turning it back into a string for as long as you can.

```
JSONSetElement ( $json ; value ; type ) → stays parsed when set to a variable  
"" & JSONSetElement ( $json ; value ; type ) → converted back to an unparsed string
```

Use indexes wisely

24U newsletter, November 2025

FileMaker differs from most database platforms in how it treats indexes. Where MySQL punishes you for forgetting one, FileMaker helpfully enables a field index the moment you do something that could benefit from it, including by mistake.



That is dangerous in a database that grows. Use Quick Find, and FileMaker immediately indexes every field on the layout that Quick Find can reach. A feature meant to help performance quietly adds permanent overhead.

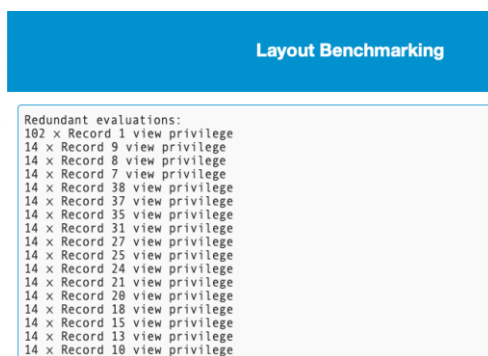
The impact starts negligible. Every index then grows with every record you add, and so does the cost of maintaining it.

Learn how indexing actually works, and decide yourself which fields deserve one. Do not let FileMaker decide on your behalf.

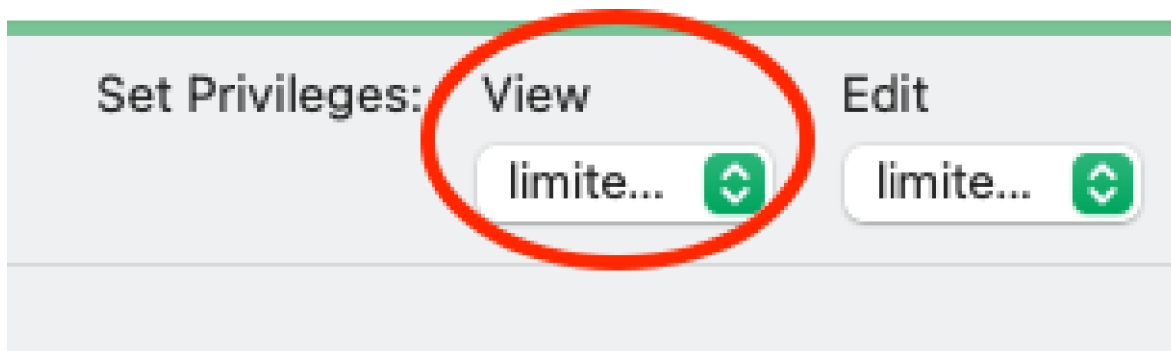
Avoid record view privilege calculations

24U newsletter, December 2025

Record-level privilege calculations are useful for security and expensive for performance, but until the layout benchmarking work, we had no idea how expensive.



The calculation engine is roughly 200 times slower than JavaScript at the same arithmetic. It can also be invoked around 100 times more often than necessary. Applied to a record view privilege, those multiply: about 20,000 times slower than it needs to be.



This is not an argument for giving up security. It is an argument for protecting data more cheaply where you can: denying access to a whole table, or simply simplifying the privilege calculation. The saving is multiplied by a factor in the thousands.

Cache your calculation results

24U newsletter, March 2026

You cannot control how many times FileMaker Pro decides to evaluate a calculation. Record view privileges, hide conditions, conditional formatting and tooltips are all triggered far more often than necessary. So stop fighting the count and make the repeats cheap.

```
Case ( $calc_expiration[1] > Get ( CurrentTimeUTCMicroseconds ) ; $calc_value[1] ;  
  Let ( [  
    $calc_value[1] = /* your original calculation */ ;  
    $calc_expiration[1] = Get ( CurrentTimeUTCMicroseconds ) + 1000000  
  ] ; $calc_value[1] )  
)
```

The original calculation is now evaluated at most once per second. However many times FileMaker asks, the rest come back cached. The constant on line four is the cache lifetime in microseconds.

The [1] index lets you cache several values in just two variables, keeping the Data Viewer readable. Use a unique index per cached calculation.

Caveat: checking and storing the cached value is itself overhead. This only pays when that overhead is small next to the original calculation. Wrap something trivial and you make it slower.

Use transactions for bulk changes

24U newsletter, May 2026

Data integrity should be the reason you use transactions. Better performance is a bonus you get almost for free.

Modify 100 records in a loop without a transaction and FileMaker Pro sends roughly 300 requests to the server: one to lock each record, one to update it, one to commit. At 50 ms of network latency that loop takes at least 15 seconds.

Open Transaction []

```
Go to Record/Request/Page [ First ]
```

Loop [Flush: Always]

```
Set Field [ X::junk1_indexed ; Random ]
```

```
Go to Record/Request/Page [ Next ; Exit after last: On ]
```

End Loop

Commit Transaction

Enclose the same loop in a single transaction and the count drops to a little over 100.

There is still one lock request per record, but only one commit, and the number of update requests depends on the total bytes sent rather than the number of records, roughly one request per megabyte.

The same 100 records finish in around 6 seconds instead of 15.

Debug and benchmark custom functions in the Data Viewer

24U newsletter, June 2026

A custom function is something you write once and use for years, long enough to forget how it works. Which is fine, until it stops working.

Paste this into the Watch tab of the Data Viewer, with your function's code in the middle. You can now test it against any parameter values you like, and measure how long each evaluation takes.

```
Let ( [
  param1 = <your test value for param1> ;
  param2 = <your test value for param2> ;
  start = Get ( CurrentTimeUTCMicroseconds ) ;
  result =
  /* paste the whole code of your custom function here */
  ; time_elapsed = Get ( CurrentTimeUTCMicroseconds ) - start ] ;
  "Time elapsed: " & time_elapsed & " μs¶Calc result:¶" & result )
```

If you need to see inside, set local variables such as \$debug within the function and add them to the output on the last line.

Use a server-side script queue

24U newsletter, August 2026

Perform Script On Server is convenient, and it carries a cost that is easy to miss: every call starts a new session.

To run one script the engine must read the database schema, build a fresh cache, construct the dependency tree, load plug-ins and open a virtual window to hold the context, then tear all of it down again when the script returns. Negligible on a small file. Significant as the solution grows.

The older technique, from before Perform Script On Server existed, avoids this: a single server-side script runs in an endless loop, watching a queue table. Empty queue, pause a second. Otherwise process the task, write the result back, mark it done, move on.



The session overhead is then paid once, when the looping script starts, rather than on every task.

Rule of thumb: if the delay between calling Perform Script On Server and the script actually starting work is more than a second, a script queue is worth building.

One of these every month or so

Every one of these ten started life as a performance tip in the 24U newsletter, alongside whatever we had just discovered, broken, measured or built an example for. We aim for monthly and sometimes miss, usually because we are in the middle of measuring something.

Subscribe

24usw.com/subscribe

Also from this session

24usw.com/laybench

the layout benchmarking example this handout keeps referring to

24usw.com/WebP

the WebP converter example

24usw.com/backforward

a working Back and Forward button

24usw.com/fmsize

the shell script that tells you what ate your disk space

Tried all ten and it is still slow?

Thirty minutes, no charge. Bring your numbers.
No proposal unless you ask for one.

24usw.com/consulttips

Or just send me a surprising measurement. I collect those.

