Databases are an effective and efficient method for storage and access of mass-spectrometry data

William Kumler
Sam LaRue
Anitra E. Ingalls
https://researcher-resources.acs.org/publish/author_guidelines?coden=jprobs

Abstract

(200 words max)

Introduction

Mass spectrometry (MS) still lacks a performant data access format. The mzML file type[1], a result of over a decade of interlaboratory collaboration and workshopping, struggles to provide rapid computational access to the *m/z* and intensity pairs. This is the crucial component in nearly all mass spectrometry analysis, but mzML's text-based XML format requires time-consuming decompression performed one scan at a time. This is largely due to its preservation of the scan as the unit of transaction while the field moves increasingly away from single-scan analysis[2,3].

As a result, alternative file formats are proposed practically every year. These include direct improvements to the mzML format with indexing[4] and better internal encoding of the data[5], HDF5-based alternatives[5–9], relational databases[10–14], or fully custom alternatives[15,16]. Fundamentally, these alternatives implement tradeoffs between user sanity in exchange for access speed and/or size on disk with clever compression algorithms and modern data structures that move away from the human-readable format of the mzML. These optimized formats are inherently more difficult to understand and usually lack comprehensive documentation or examples (particularly across programming languages) making it difficult for new users to enjoy their benefits or extend their functionality. This steep learning curve, coupled with a lack of support in conversion tools such as Proteowizard's msconvert[17], has prevented widespread adoption of these new formats despite their clear computational advantages. Such formats are also fragile in the sense that without community support, their continued development depends entirely on the original developers and easily become deprecated (as is the case with YAFMS, Shaduf, and mz5, all of whom have links in their papers that currently redirect to missing webpages). A simple, speedy, and small MS data format remains very much in demand.

Relational databases are not new for MS workflows (see references above) and compete predominantly with HDF5-based methods. Both of these systems are widely used for big data and can be applied to MS data in a plethora of ways, leading to the proliferation of implementations we see today. Both backends provide excellent universality, larger-than-memory support, and rapid access to data, but HDF5-based systems excel at self-description and hierarchical structures[9] while the relational database model is optimized for multi-table queries using a consistent syntax[18]. Relational databases are increasingly seen in MS workflows for both raw and processed data, with SQLite backends now supported in the popular peakpicking software xcms[19] via the Spectra package[20] (though in-memory and HDF5 options are also supported) and on MetabolomicsWorkbench[21] while the development of MassQL[22] demonstrates the increasing comfort that MS analysts have with the adoption of SQL.

Relational databases also have several distinct advantages over hierarchical or text-based systems, particularly in performing searches for subsets of data via indices. Importantly, this indexing differs from the byte-offset indexes that already exist in the indexed mzML and HDF5 formats because the search for a particular subset cannot be done efficiently with a byte-offset index when the *m/z* data is encoded, though access to a particular scan can be incredibly rapid. Additionally, data from multiple samples can be stored together in a single database table, differing from formats like mzDB, mzTree, and mzMD and allowing queries of all dataset samples to be performed without looping through each file in turn, thereby avoiding the associated computational overhead and query complexity.

encoded how? Is this common knowledge?

This sentence is hard to follow, partly b/c of the phrase circled.

SQL databases also allow mass spectrometrists to access the continual improvements and long-term stability produced by the industries who specialize in these. While HDF5 is a common scientific data format, databases are constantly under development by industry titans deeply invested in their maintenance and optimization. Online analytical processing (OLAP) methods are particularly well suited for MS data given their optimization for read speed under the assumption of infrequent transactions, making modern systems such as DuckDB[23] or Apache's Parquet formats highly appealing while preserving the familiar file-based serverless approach.

Our previous work showed how the ragged arrays of MS data can be converted into a tidy database table in memory[24] and we now logically extend that method into proper database storage. Here, we test the hypothesis that a "vanilla" implementation of a relational database which exposes the raw m/z and intensity pairs is an intuitive and performant way of storing MS data for exploratory analysis, visualization, and quality control. We compare the time and space required to extract a representative data subset under six conditions and perform these tests on multiple databases as well as mzML and other MS data formats.

Experimental section

We performed a literature search for mass-spectrometry data formats that have been published in the last 15 years and attempted to find or construct parsers for each format in Python, a popular high-level interpreted language. Each parser was written to perform three common exploratory data analysis operations on full-scan data and three common operations on MS/MS fragmentation data. Full scan queries consisted of 1) single scan extraction by scan number, 2) retention time range extraction of all scans within a specified retention time range, and 3) chromatogram extraction, which collects the ions within a specified parts-per-million (PPM) error of a known mass. These queries generally correspond to the methods used in[11], which performed similar tests benchmarking the mzDB format against mz5 and an mzML parser. Note that the chromatogram extraction does not extract a precompiled chromatogram of the sort commonly found at the end of mzML files or as a result of SIM/PRM analysis but instead refers to sifting through the raw data for data tuples with an *m/z* value between specified bounds. MS/MS queries involved extracting three relevant subsets, consisting of 1) a single scan extraction by scan number similar to that of the full scan, 2) extraction of all the fragments associated with a precursor *m/z* within a given PPM, and 3) extraction of all fragments with *m/z* values within a given PPM.

We explored the available documentation on PyPI and Github for each mass spectrometry data format and either identified existing functions and packages that would perform the above queries or wrote our own functions if necessary.

Mass-spectrometry files and software used

We browsed Metabolights[25] for suitable LCMS datasets, looking for studies that included 100+ gigabytes of data from both full scan and MS/MS analysis. We were also restricted to the Thermo Scientific .raw file format, as it was the most widely supported by alternative MS storage methods. We also excluded polarity-switching data as it is unclear whether all converters would be able to separate scans based on polarity.

Files were downloaded as .raw. mzML, mz5, and mzMLb were all natively supported by Proteowizard's msconvert software (version 3.0.25009) while MZA (v1.24.11.16) and mzDB (v0.9.10_build20170802) had separate extensions to this executable enabling their own conversion. MzTree and mzMD were converted via their GUI which did not have release or versioning information available but were downloaded from Github (https://github.com/optimusmoose/MZTree and https://github.com/yrm9837/mzMD-java, respectively) and built via Maven (v3.9.9) for Java (v21.0.6). SQL databases were built using Python 3.11.11 with SQLite (v3.48.0) via the Python sqlite3 package (v2.6.0), DuckDB via the duckdb package (v1.1.3), and Parquet files via the pyarrow package (v19.0.0). We were additionally able to download and install [blah]

mzML access was done with Python's pyteomics package (v4.7.5), the pymzml package (v2.5.10), and the pyopenms package (3.0.0.dev20230306). MZA files were accessed via the mzapy library (v1.8.dev4 from the no_full_mz_array branch on Github) and via custom code built around the h5py package (3.12.1). Custom parsers were required for mzDB, mz5, MzTree, and mzMD.

Database schema

The "vanilla" database style proposed here abandons a 1:1 representation of the original vendor-specific file. This decision was made after discussion with a wide variety of experts, all of whom preserved the original MS files even after conversion to another file type, indicating that a highly-performant addition is more important than direct replacement. Here, we map MS concepts (retention time, *m/z*, intensity, etc.) directly to database fields to make downstream processing as intuitive as possible. Metadata is stored separately in file_info and scan_info tables that are linked by filename and scan number (Figure 1). We do not force compression of any of these fields because we believe that decoding compressed data is both a slow and unintuitive step, though automatic compression is supplied by the DuckDB and Parquet file types.

→ find?

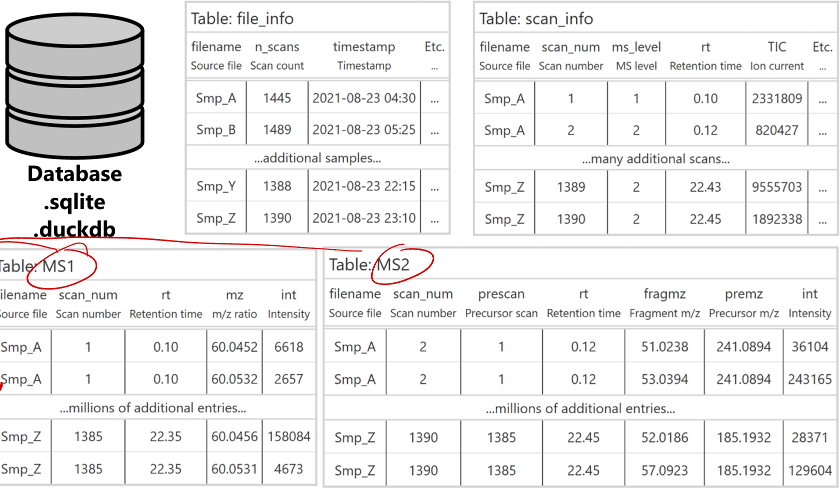


Table: file_info

filename Source file	n_scans Scan count	timestamp Timestamp	Etc. ...
Smp_A	1445	2021-08-23 04:30	...
Smp_B	1489	2021-08-23 05:25	...
...additional samples...			
Smp_Y	1388	2021-08-23 22:15	...
Smp_Z	1390	2021-08-23 23:10	...

Table: scan_info

filename Source file	scan_num Scan number	ms_level MS level	rt Retention time	TIC Ion current	Etc. ...
Smp_A	1	1	0.10	2331809	...
Smp_A	2	2	0.12	820427	...
...many additional scans...					
Smp_Z	1389	2	22.43	9555703	...
Smp_Z	1390	2	22.45	1892338	...

Table: MS1

filename Source file	scan_num Scan number	rt Retention time	mz m/z ratio	int Intensity
Smp_A	1	0.10	60.0452	6618
Smp_A	1	0.10	60.0532	2657
...millions of additional entries...				
Smp_Z	1385	22.35	60.0456	158084
Smp_Z	1385	22.35	60.0531	4673

Table: MS2

filename Source file	scan_num Scan number	prescan Precursor scan	rt Retention time	fragmz Fragment m/z	premz Precursor m/z	int Intensity
Smp_A	2	1	0.12	51.0238	241.0894	36104
Smp_A	2	1	0.12	53.0394	241.0894	243165
...millions of additional entries...						
Smp_Z	1390	1385	22.45	52.0186	185.1932	28371
Smp_Z	1390	1385	22.45	57.0923	185.1932	129604

very non-descriptive names. Are these standard in the literature?

Ion chromatogram extraction: `SELECT * FROM MS1 WHERE mz BETWEEN min AND max`

Retention time range subset: `SELECT * FROM MS1 WHERE rt BETWEEN min AND max`

Fragment search: `SELECT * FROM MS2 WHERE fragmz BETWEEN min AND max`

Precursor search: `SELECT * FROM MS2 WHERE premz BETWEEN min AND max`

Figure 1: Database schema for an example MS/MS dataset showing the organization of mass-spectrometry data into tables. Fields of interest are easily queryable with simple SQL commands as shown in the table at bottom.

Supplemental figure XX: [TBD] Additional possible extensions to the database schema for retention time correction, feature detection and correspondence, ion mobility, and imaging.

Time and space testing

We randomly sampled a single file out of the full dataset for comparison across metrics and file formats (20220923_LEAP-POS_QC04). We sampled 100 random scan numbers using SQLite's ORDER BY RANDOM() function and pulled out the largest ions for chromatogram extraction using a 10 ppm mass range window. Retention time ranges were the highest-intensity retention time of each ion chromatogram plus or minus one minute. Similarly, the largest fragments by intensity were used for the MS/MS metrics with a 10 ppm mass range exclusion window.

Timing was performed via Python's timeit library and the timeit.repeat function, with the various file formats as the innermost loop to ensure bias over time was distributed equally among function calls. File sizes were estimated using Python's os library with os.path.getsize. We did not exhaustively monitor memory usage, though our heuristic exploration of the timing scripts did not ever indicate that memory was a constraint.

Timing data was obtained on an Intel Xeon CPU with two X5650 (@2.67 GHz) processors and 24 total cores running Windows 10 Pro (64 bit version). 96 gigabytes of RAM (DDR3 @ 1333 MHz) were available and a solid-state drive was used for disk storage.

Results

We settled on a large dataset of gut microbiota LC-MS files published in[26] and available on Metabolights under accession number MTBLS10066.

All existing MS data formats demand a high level of domain knowledge

We were able to obtain or write parsers for seven different existing mass spectrometry (MS) data formats: mzML, mzMLb, mz5, mzDB, MZA, MzTree, and mzMD. Multiple Python packages existed for the mzML data format so we used each of the three dominant packages (pyteomics, pyOpenMS, and pymzml) and compared their timing results as well. We failed to produce parsers for the YAFMS and Shaduf file types due to complete deprecation (links to these no longer exist), the toffee file type due to its application solely to time-of-flight (TOF) data-independent acquisition (DIA) data, the Aird file type due to its current deprecation in Python and C#, and the UIMF format due to a complete lack of interface documentation.

File conversion support varied enormously

Conversion from the initial Thermo .raw file type to the open-source .mzML format was seamlessly performed by Proteowizard's msconvert library. Similarly, Proteowizard support for the .mz5 and .mzMLb file types made their conversion trivial.

mzDB and MZA both had reasonably good documentation, both formats notably providing self-contained extensions to msconvert for ease of conversion. However, both converters provide limited coverage, with mzDB notably missing support for Waters and Agilent .d files while MZA currently lacks support for AB Sciex .wiff and Bruker .baf files. Both converters are only available via binary executable (.exe), restricting their use to Windows platforms. Additionally, both parsers appear to be unable to separate scans from a polarity-switching experiment or support any of the other filters available natively in msconvert, as additional arguments passed to the executable throw errors instead of being passed along to the original software.

MZTree and its derivative, mzMD, provided significantly less documentation about the conversion process than the other file types. This documentation consisted solely of the README available in the associated Github repositories and their installation and deployment required rebuilding the Java applet, of which the bare-bones instructions make several assumptions about the user's PATH environmental variable. In the case of mzMD, no documentation for installation and build was provided and this instead needed to be deduced from MZTree. Additionally, we ran into issues with hardware acceleration once the GUI was launched that required extensive debugging. The GUI conversion, however, is straightforward once the app is correctly compiled and launched, albeit requiring a manual entry of a single file at a time with no apparent batch processing available.

The Aird file type was straightforward to convert on Windows via the executable available on Github (v6.0.0) but does not seem to be available for other operating systems, much like MZA and mzDB. Unfortunately the Python package designed to allow an interface to the file type has been deprecated and we were unable to install or use it in a meaningful way and were unable to reverse-engineer the file type sufficiently to compare it here. The UIMF file type from the Pacific Northwest National Lab (PNNL) provided documentation exclusively in the form of C# commands and did not supply instructions for file conversion, making it unclear what input formats were supported. The toffee format also provided zero documentation for conversion from other formats and was restricted to time-of-flight (TOF) data independent acquisition (DIA) MS data. Thus, we were unable to directly compare any of these three file types to the others.

Universal lack of support for the six relevant queries

Despite the relative simplicity and relevance of our queries, none of the available mass spectrometry (MS) formats had existing functions or documented examples of all six queries.

Unsurprisingly, the mzML file type had the most extensive coverage but documentation and prebuilt functionality was still disappointingly sparse. The pyteomics package provides four “combined examples” that focus on the spectrum visualization and annotation common to proteomics research but provide minimal guidance about chromatogram or retention time range extraction. pyteomics also provides native support for the mzMLb file type and was the only one of the three Python packages to do so, deserving praise for the minimal disruption that mzMLb files placed on existing pipelines if they were to switch from mzML to mzMLb. The pyopenms package provides similarly extensive documentation for proteomic and scan-based analysis but again lacks information about subsetting in the retention time direction, though the existence of an undocumented parser (get2DPeakDataLong) provides a simple way to do this for MS1 data. Additionally, pyOpenMS required installing an old version of the package (3.0.0), Python itself (3.11) and the numpy package (<2.0) due to more recent builds requiring AVX support which was unavailable on our hardware. pymzml is intentionally a lightweight parser focused exclusively on reading mzML files but does not supply any functions for the queries other than scan extraction by number and the “Spectrum and Chromatogram” documentation module was empty at the time of writing (February 2025).

mz5’s documentation was remarkably sparse, especially considering it was one of the earliest mzML formats and is supported by Proteowizard. Crucially, the original paper[6] contains links to a website (https://software.steenlab.org/mz5) which currently returns an HTTP error 500. A Python library (pymz5) exists but requires an old version of Python (2.7 or 3.2) and hasn’t been updated in 12 years and is predominately a simple fork of h5py[27] with three mz5-specific commits on top. Most problematically, we were unable to determine how mz5 stores precursor *m/z* ratios, making the fragment and precursor searches impossible. This was largely due to the variable-length nested compound structures mz5 that are not supported in all APIs, e.g. Java.[5]

mzDB access was hamstrung by several issues, primarily the outdated repository that implies Python and R support via a port from Rust but was unavailable at the time of development, though we are grateful for the responsive developer who notified us that this implementation was not feature-complete. This required that we deduce the SQLite BLOB type compression format from scratch when writing a parser and spend extensive time reading through the documentation to determine how best to link the various tables provided in the mzDB file. Scan metadata in this file type is stored as raw XML strings, producing the worst of both worlds in requiring both SQLite knowledge in their extraction and XML processing to obtain the relevant information. Additionally, its failure to implement the clever bounding box and run slice scheme for MS/MS data negated with our ability to avoid parsing every MS/MS spectrum when performing precursor and fragment searches.

MZA provides a complementary Python package, mzapy, for access to MZA files. Here again we ran into several issues with its installation and use stemming largely from the deployed package requiring TOF bins for parsing, though a separate Github branch provides a workaround and the rapid developer response was appreciated. The mzapy package provides a clear example of chromatogram extraction as well as a method for retention time range extraction, though there exists no clear function for the extraction of a single spectrum by scan number despite the internal file structure being highly optimized for this purpose. mzapy also provides good support for ion mobility extraction but fails to index MS/MS information or provide any clear way to extract fragments by *m/z* or precursor.

MZTree and mzMD provide a slightly strange interface to MS data, requiring a separate Java server that is then be queried via an HTTP API. For users without prior knowledge of HTTP request methods or exposure to programming APIs, the README is entirely unhelpful because it simply documents the API's endpoints and provides no complete query strings as examples to guide the user. This combination of GUI server and command-line HTTP request inverts the typical paradigm of GUI for exploration and command line for construction to convoluted effect, though the structure of the data returned by the server is impressively simple. More problematically for this analysis, the API provides no apparent way to access MS/MS data or query the files by scan number, with only RT and *m/z* bounds controlling the subset of data extracted. Finally, the GUI provides no way to open multiple files simultaneously or iterate through files programmatically and instead requiring point-and-click interaction with the GUI each time a file is opened or closed, preventing us from making reasonable comparisons in tests requiring multiple files.

Wow. So much work!

SQL-based parsers were simple to write and use

We then used custom code to convert the mzML files into SQLite and DuckDB databases using a simple schema for full scan (MS1) and MS/MS (MS2) data. The MS1 table consisted simply of fields for filename, scan index, retention time, *m/z* ratio, and intensity. The MS2 table consisted of the same fields except that the *m/z* column was separated into precursor and fragment *m/z*. Although we did not extend these databases to include the metadata associated with each file and scan, the logical framework could be easily extended in future work and the metadata typically represents a small fraction of the total space within the file, allowing us to make reasonable comparisons about file size between the databases and the metadata-rich other file types. We also converted each file's

MS1 and MS2 table into Parquet representations for comparison using the same field/column schema.

We found that the documentation for SQLite, DuckDB, and Parquet file formats in Python far exceeded the documentation available for any mzML parser. This is unsurprising given that these file formats are used widely outside of MS research and are developed and maintained by dedicated teams. Additionally, the use of a consistent SQL syntax for table creation and insertion meant that the same code could be used to write to both SQLite and DuckDB, as well as any other databases supported in Python. The use of packages such as SQLAlchemy could be used to additionally streamline this process to any additional database by simply swapping in a new database engine.

Querying the MS1 and MS2 tables was also very straightforward. After establishing a connection to the database, the six queries could be asked using nearly human-readable SQL syntax. Requesting the thousandth MS1 scan by number consisted simply of `SELECT * FROM MS1 WHERE id = 1000` passed along to the `pandas.read_sql_query` function. More complicated queries such as retention time range (`SELECT * FROM MS1 WHERE rt BETWEEN 6 AND 8`) and a precursor mass search (`SELECT * FROM MS2 WHERE premz BETWEEN 118.086 AND 118.087`) were similarly intuitive.

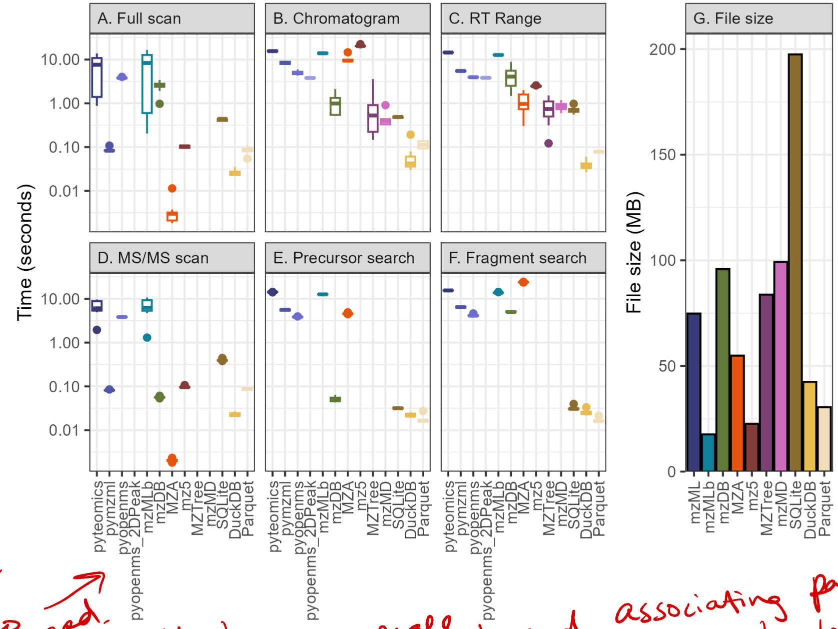
Time and space requirements for a single DDA file across formats

Spectrum extraction

The simplest and most abundantly documented query was the extraction of a single spectrum. In many ways, this is the fundamental unit of mass spectrometry and thus many formats are highly optimized for its extraction into manipulatable data (Figure 2A and 2D). Here, we found that the mzML and mzMLb file types were consistently the slowest to parse the mzML file type and required multiple seconds, likely highlighting inefficiencies in the `pyteomics` package used to parse both file types.

Confusing: file types used twice. I think one is wrong.

pyopenms also struggled to open and extract a specific scan, requiring several seconds due in large part to the expensive initiation function, after which requests were orders of magnitude faster (Supplemental Figure XX). It is also worth noting that while both of these packages provided rapid extraction of a *random* spectrum, a significant overhead was introduced by needing to scan through the file to find a *specific* spectrum by scan number as scans are not always consecutive (e.g. during multiple MS experiments, polarity-switching runs, or when scans have been filtered out during conversion) and no metadata was obviously available that would have allowed using the index directly to a specific scan.

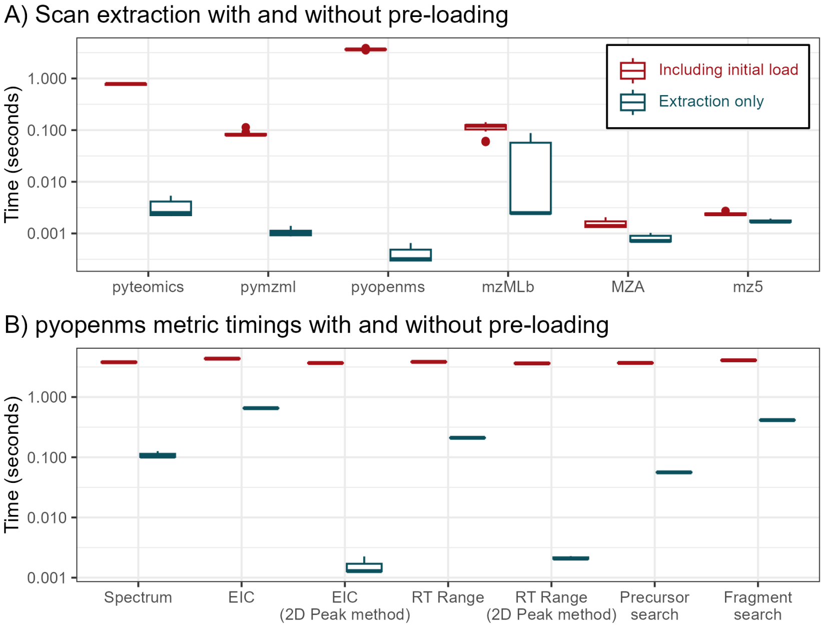


These are hard to read.

Maybe: an overall legend associating package w/ color?

Figure 2: Query time for the six data extraction methods and the associated file sizes for all 13 methods explored in this paper. The left six panels show boxplots representing the time required in seconds to extract an full scan spectrum (A), an ion

chromatogram (B), all data within a retention time range (C), an MS/MS scan (D), the fragments of a specified precursor (E), and all precursors with a specified fragment (F). The error in the boxplot is composed of timing information for 10 repeated queries, each of a different target scan number, retention time (RT), or m/z. The right panel (G) shows a barplot of the size on disk in megabytes (MB) occupied by each file type.



A) Scan extraction with and without pre-loading

B) pyopenms metric timings with and without pre-loading

Supplemental figure XX: Timing information with and without pre-loading the object into Python prior to extraction. Panel A) shows the number of seconds required to extract a scan for the three mzML parsers plus the mzMLb, MZA, and mz5 file types. Panel B) shows the number of seconds required for pyopenms to perform each of the queries if the initial data load was removed from the timing count. Error bars show 3x replicate queries of 5x different targets.

The pymzml package was able to extract both MS1 and MS2 spectra from the mzML file nearly two orders of magnitude faster than the other mzML parsers, largely due to its use of naming the scans by their number and thus avoiding the expensive scan number extraction step. mzDB had the only notable difference between MS1 and MS2 scans, performing slightly better than pyteomics and pyopenms methods for MS1 data and significantly better for MS2 data, placing it approximately on par with pymzml in taking about a tenth of a second. The simple database methods (SQLite, DuckDB, and Parquet) also fell in this ~0.1 second range, with SQLite performing most poorly and DuckDB ~10x faster. Finally, both mz5 and MZA were an additional order of magnitude faster than any other method, returning the data within the spectrum in thousandths of a second. This shows the power of the HDF5 file system for data access when its location within the file is known in advance.

Chromatogram extraction and subsetting by retention time range

Ion chromatogram extraction and retention time range subsets were a key metric for us, corresponding to essential tasks in chromatographic peakpicking and adduct, isotope, and in-source fragment detection (Figure 2B and 2C). EIC query times here were universally slower than those for a single spectrum extraction, reflecting the way in which a scan-based file type is sub-optimal for chromatogram extraction because each scan must be parsed to find data within a given *m/z* range. MZA and mz5 particularly suffered, with this query type entirely negating the advantages of the HDF5 file structure.

MzTree and mzMD are both file types optimized exclusively for chromatogram extraction and performed very well on the EIC metric and were two orders of magnitude faster than those parsing mzMLs, with mzMD surprisingly less performant than the older MzTree file type it was based on. However, we also note that both Java-based applications have a slow initial file load step that must be done through a GUI and therefore could not be counted in the timing comparison, the inclusion of which would likely mitigate any advantage for a single chromatogram extraction. The mzDB file type is also optimized for chromatogram extraction and was an order of magnitude faster than the other existing file types for which all queries could be run (MzTree and mzMD do not provide interfaces for spectrum extraction or MS/MS data).

The SQLite, DuckDB, and Parquet formats were just as speedy as mzMD and MzTree with SQLite taking half a second, Parquet requiring a tenth of a second, and DuckDB reaching query times of hundredths of a second, far outstripping the seconds or even minutes typically expected of this task and resulting in a functionally instantaneous interaction for the user.

Retention time range extraction times were an average of the single-spectrum extraction and the chromatogram extraction times across the board, potentially hinting at a major predictive factor in timing estimation being the total amount of scan parsing required.

MS/MS precursor and fragment search

We also investigated the efficacy of the various MS data formats for MS/MS data and found that support for fragmentation data searches was lacking or absent from the documentation and exposed functionality of each of these file types, requiring custom implementations every time. Despite both precursor searches (where all the precursors of a given fragment are found) and fragment searches (where all the fragments of a given precursor are found) representing intuitive and useful methods of MS/MS data processing, these timings were consistently among the slowest of the six query types for the non-database methods (Figure 2E and 2F).

All existing MS data types required multiple seconds to perform a single fragment search (Figure 2F), representing a significant bottleneck for any downstream analysis requiring the data associated with the fragments of a given precursor. The SQL-based parsers, on the other hand, all took fractions of a second and consistently returned the relevant data hundreds of times more quickly than existing methods. The same was true for a precursor search across all methods aside from mzDB (Figure 2E), which benefited significantly from constructing a single bounding box for all MS/MS information that requires a single decoding into computer memory, though this strategy will fail for any file with sufficiently large MS/MS data.

File sizes

File size is another important constraint on the efficacy of various MS formats. We measured the size on disk of each of the file types and found that they varied by approximately an order of magnitude, with HDF-based file types hovering around one-third the size of the mzML (mzML size = 75 megabytes (MB), mzMLb = 18 MB, mz5 = 23 MB) while mzDB and mzMD were larger (95 MB and 99 MB, respectively). The SQLite object was the largest on disk of all the file types, nearly tripling the mzML's size at 197 MB, while DuckDB improved on MZA at two-thirds of the mzML (42 MB and 55 MB, respectively) and Parquet improved slightly upon that again (30 MB total) with its columnar-based storage format (Figure 2G).

However, these comparisons are not perfect because not all files store exactly the same data. MZTree and mzMD appear to entirely lack the MS/MS information in the sample DDA file, representing a potentially significant size reduction that's difficult to estimate though the extraction of the same file via msconvert containing only MS1 scans was 58 MB, a 23% size reduction. The SQLite, DuckDB, and Parquet formats also lack the extensive scan and file metadata that's present in the other file types, though it is difficult to estimate the fraction of disk space allocated for this (and which will depend upon the precise definition of metadata).

Timings for multiple chromatograms

The single-file, single-metric case discussed above and shown in Figure 2 is largely a worst-case scenario for many MS data systems that have a slow initial setup step to make downstream analysis faster. To compare these systems more fairly to our database schema, we also tested timings across multiple chromatograms. In each case, this was implemented as a for loop iterating over an increasing number of chromatograms corresponding to the largest intensity ions in the file (Figure 3).

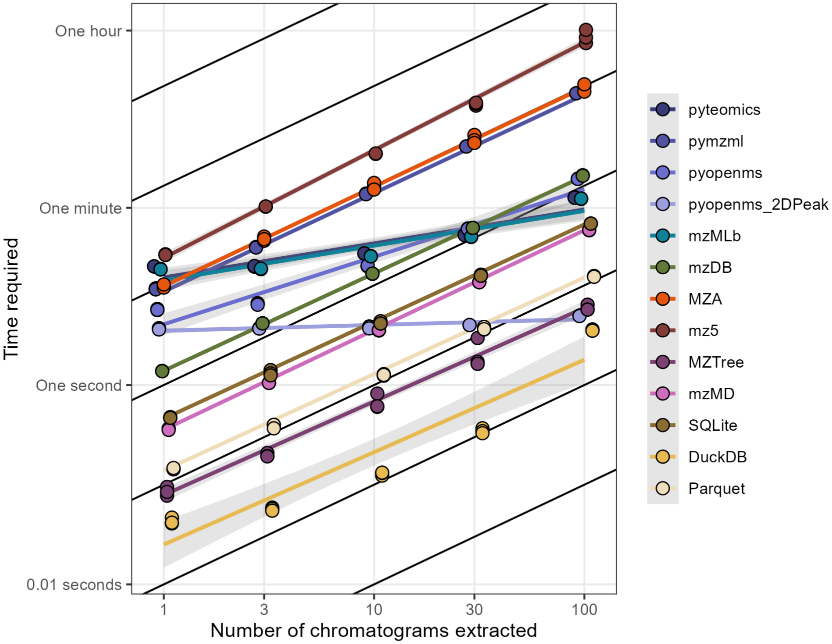
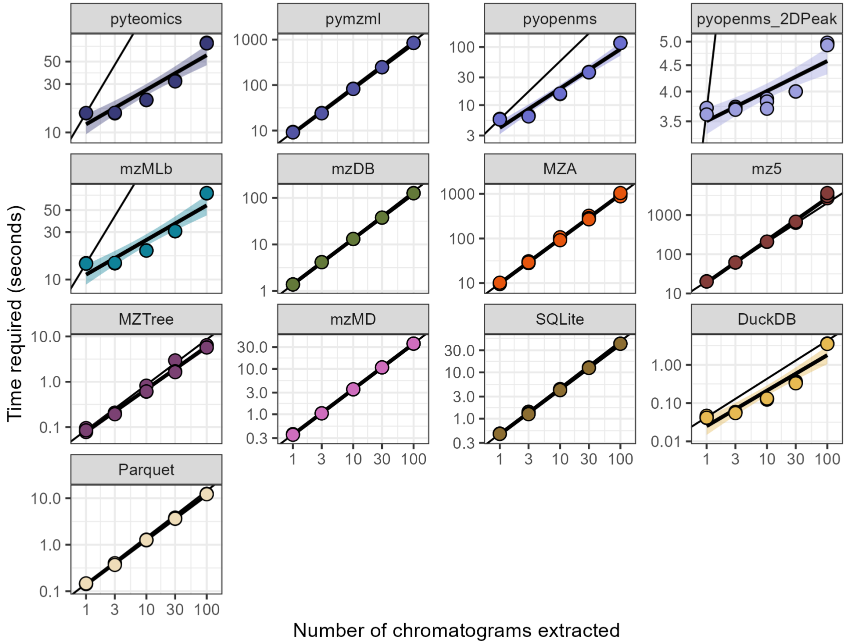


Figure 3: Scatter plot of the time required to extract multiple chromatograms using various methods on logarithmic axes. Best-fit linear models have been added for each method are

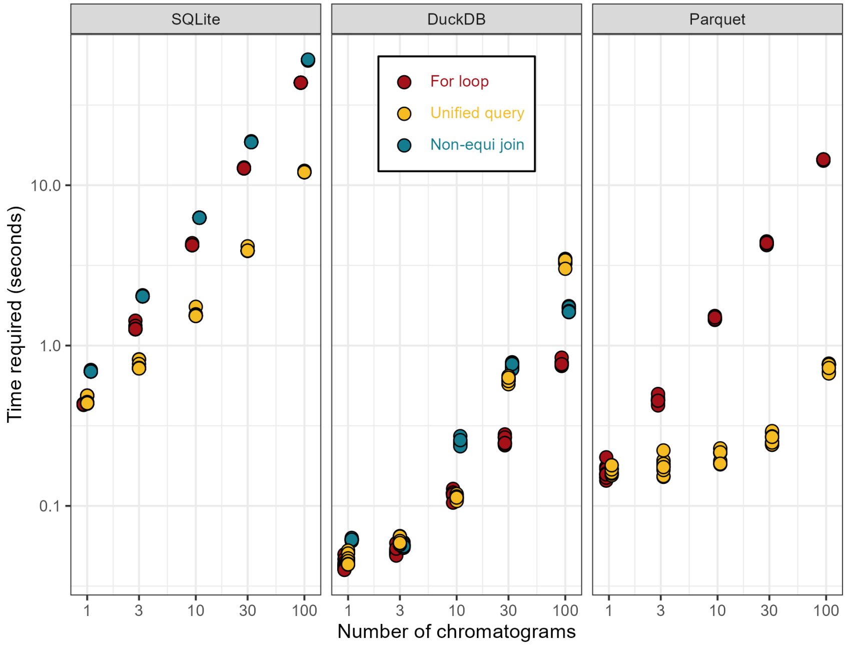
shown behind triplicate timing measurements. Transparent intervals around each best-fit line show a single standard error of the mean. Chromatograms correspond to the largest intensity ions in the file. 1:1 lines have been added in black behind the data for comparison.

Most methods were linear extrapolations of the single chromatogram numbers shown above as expected from a simple for loop, with notable exceptions for pyteomics (and thus the mzMLb format), the 2D peak method of pyopenms, and DuckDB (Figure 3). pyteomics and pyopenms both had significant overhead upon initial load that resulted in faster query times afterward that performed better than a 1:1 extrapolation would expect, with pyopenms matching SQLite's speed after 10 chromatograms and Parquet's speed after 30. Interestingly, the methods that had a best-fit linear slope less than 1:1 also all had exponential fits, with performance at high chromatogram number actually worse than would be expected from a prediction fit to the timings for 1 and 3 chromatogram extractions (Supplemental figure XX).



Supplemental figure XX: Data from main text Figure 3 with independent y-axes showing scatter plots of the time required to extract multiple chromatograms using various methods on logarithmic axes. Best-fit linear models have been added for each method and are shown behind triplicate timing measurements, while the black line shows a 1:1 slope intercepting the first data point to show nonlinear behavior with additional chromatograms. Transparent intervals around each best-fit line show a single standard error of the mean.

We also explored whether database queries could be improved via the use of either a unified query (single SQL statement with multiple OR clauses for each ion's *m/z* range) or a non-equi join between a peak table with *m/z* minimum and maximum columns (Supplemental Figure XX). SQLite and Parquet performed ~3-5 times faster with the unified query than with the loop method despite the necessity of and additional processing step for the looped query to correctly assign each data point to its original peak information. The opposite was true for DuckDB likely due to its optimized reader, with the unified query consistently outperformed by the non-equi join when 100 chromatograms were extracted.



Supplemental figure XX: Scatter showing time required to extract chromatograms from SQLite, DuckDB, and Parquet databases using different methods denoted by color. Ten replicates are shown for each data point.

Database optimization via indices/ordering and multi-file constructions

Databases also provide multiple ways to optimize queries. SQLite allows the construction of indices for a field within a table that then speeds up queries at the cost of additional disk space. Alternatively, DuckDB and Parquet files rely predominantly on the data order when it’s written to disk and use their sophisticated row group methodology when subsetting.

We found that SQLite queries benefitted significantly from the construction of an index on the *m/z* column when extracting chromatograms, improving lookup times by an order of magnitude (dropping from [0.3] seconds to [0.03 seconds], Figure 4). However, because the SQLite index is stored on disk alongside the data, this improvement also increased the file size by 33% from [150MB] to [210MB]. DuckDB also improved significantly with ordered data but to a smaller degree and surprisingly required additional space to do so, likely due to the reordering resulting in a different compression strategy (Figure 4). Parquet files had the smallest improvement upon data ordering but remained the same small size they were when unordered.

how much?
I don't see this
in Fig. 4

Crucially, these improvements also persisted when multiple files were stored in a single database. We built databases consisting of between 1 and 100 individual MS files and tested the time required to extract ion chromatograms from each after an index was constructed (Figure 4). DuckDB was consistently the fastest ion chromatogram extraction method, with query times around 0.03 seconds for a single file and 1 second for one hundred files. SQLite had much higher variance and slower extraction times with datasets consisting of more than one file, typically an order of magnitude slower than DuckDB, while Parquet fell between the two. Importantly, only DuckDB had a slope significantly less than one. This is what would be expected if the database was performing a simple binary search on the index, with an expected time efficiency of O(log(# of files)). However, DuckDB's performance degrades at larger database sizes and approaches a 1:1 slope, possibly due to the overhead of reading large amounts of data into memory after it's found. We additionally compared these values to the timings obtained from converting each file into its own database and looping over each of those to confirm the linearity of that response (Figure 4). Interestingly, SQLite timings were slightly better with multiple database files instead of one combined database (Figure 4).

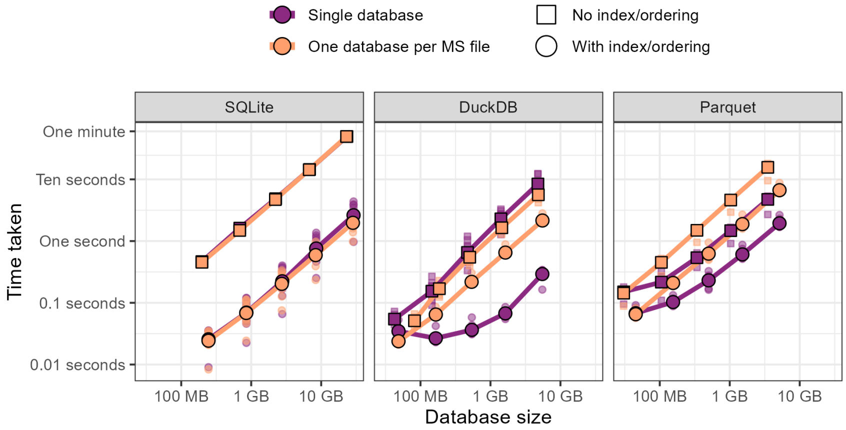


Figure 4: Time required to extract an ion chromatogram from multiple files plotted against the size of the data, broken down by the type of database used (SQLite, DuckDB, or Parquet). Points correspond to a random subset of 1, 3, 10, 30, and 100 files, respectively. Colors specify whether the data was stored as a single consolidated database (purple) or with a single database per file (orange) and the shape of the point denotes whether the database was unstructured (squares) or indexed/ordered by m/z (SQLite has indexes, DuckDB and Parquet benefit from ordering). Ten replicates of each query were performed and are shown transparently behind the mean values connected with lines.

Discussion

As the gap between data scientist and mass spectrometrist continues to narrow, mass spectrometry (MS) data formats should facilitate this convergence. Instead, MS software remains relatively opaque. Documentation is sparse and data structures are complex, resulting in a landscape that is essentially restricted to the original developer's intent. A particular pain point is the way in which MS data is stored because current methods must make trade-offs between simplicity, size, and speed. When we explored the wide range of MS file readers, we found every method had flaws that interfered with widespread adoption. The mzML file type appears to represent low-hanging fruit, with its XML-based structure that sacrifices speed and size in favor of clarity, but no alternative has yet reached a large audience of active users. Fifteen years of active competition continue to favor the highly explicit format, likely because users are leery of incomprehensible alternatives with no guarantee of continued maintenance.

Certain methods are clear winners for individual use cases but all of the existing formats failed to perform well at the full suite of exploratory data analysis tasks we attempted. Scan extraction is perhaps the most widely used query, especially for proteomics, and as a result has been extensively optimized. Here, MZA was blazingly fast thanks to the decision to index in the HDF5 file by scan number. The mzML files used here did have a precompiled index that should have made scan extraction highly efficient but this appeared to be mitigated for pyopenms and pyteomics due to their long initialization times, though pymzml performed very well at this task and the other two mzML methods were much faster after initialization (Supp Fig XX).

However, performant scan-based methods struggled significantly with chromatogram extraction because these axes are inherently orthogonal to each other. Here, mzDB performed quite well and was competitive with the specifically optimized mzTree and mzMD formats and pyopenms performed incredibly well when extracting multiple chromatograms via its 2DPeak method. Chromatograms (and retention time range queries) are of course only relevant for chromatography-based workflows but this type of analysis has become increasingly popular and we expect it to continue to do so, making ion extraction increasingly important. None of the existing MS formats we tested performed very well on MS/MS data, despite the growing availability of fragmentation data, though our use-case is oriented more towards exploration instead of comprehensive analysis.

Finally, a complexity penalty must be noted for formats and packages requiring complex installation procedures. pyopenms appears to be the worst offender here, with its bindings to the OpenMS C++ libraries requiring us to step back to Python 3.11 and numpy 1.26 to successfully access the data on our setup and required direct input from the maintainers. The Java applications for MzTree and, more egregiously mzMD, provided essentially zero documentation on installation consisting of a single README file without an intact HTTP request example and have not seen updates in years. Similarly, mzDB files were difficult to parse due to its opaque SQLite schema and use of the BLOB encoding type that again lacked examples or documentation outside of Java and had to be deduced iteratively. Of course, these methods eventually provided enough information that they could be parsed unlike Aird, toffee, and UIMF.

Timing comparison to existing literature

Novel formats are typically proposed with timing and sizing information, but the inconsistency of what's being queried makes it difficult to directly compare across the literature. However, we mostly obtained results in line with those reported elsewhere where other intercomparisons have been performed and report here the widest set of comparisons between MS formats to our knowledge.

[6] performed comparisons between mz5 and mzML with results

indicating that mz5 was three times faster than mzML parsers. Their values of 0.16 seconds per million *m/z*/intensity arrays correspond to an estimated query time of 0.13 milliseconds which is faster than our measured 3 milliseconds, though our mz5 file was approximately four times smaller (20% mzML size) instead of two times smaller (50%).

[11] also found that mz5 was about 80% smaller than the mzML and on par with their mzDB format. They reported query times of around 30 seconds for a wide (5 Da) ion chromatogram extraction and found that mz5 was about 40 times slower and that mzML was 200 times slower, in contrast to our parser which was 10 times slower for mz5 - possibly due to the large EIC width. They also were able to report mzDB scan queries on par with mz5 which we were able to replicate if the pyopenms or pyteomics libraries were used for full scan queries and pymzml for MS/MS.

MzTree[12] compared their SQLite-based system to mzML, mz5, and mzDB and reported high numbers for both EIC and RT range queries from mzML (4-1000 seconds) that were on par with the values we observed here. Their mzDB and mz5 results were unexpectedly comparable to each other at about 0.5 seconds per random EIC query, with the mzDB values equal to ours but the mz5 values much lower than our parser was able to obtain. Our disk size measurements also corresponded well with their size estimate of MzTree at approximately twice the size of the mzML, a surprising result given that our mzML file contained MS/MS information and theirs appearing to be full-scan only. Their mz5 files were much larger (80% mzML size instead of our 20%) and their mzDB much smaller (20% mzML size instead of our 110%).

The mzMD format[13] appears to be a thin wrapper around the MzTree format that applies a different philosophy for data subsetting and summarization. They report EIC queries in the 50 millisecond range, very similar to the values we obtained for the mzMD file type. They also estimate file size to be approximately 28 bytes per *m/z*/intensity tuple for a total size of 100MB in our test file which agrees reasonably well with our 72MB actual measurement. Their comparison to MzTree also agrees with our results as they report slightly larger disk usage and slightly better performance.

The mzMLb group[5] reports only info for spectrum access at approximately 15ms/scan which agrees with our 5-100ms/scan estimates only if the data is loaded ahead of time. They perform extensive comparisons to mzML at varying compression methods and levels but our default options of 1024 KB chunk sizes for the mzMLb file and zlib but not numpress compression for the mzML resulting in timing values very similar between mzML and mzMLb using the pyteomics library for access. They also compared to the mz5 file type and we are able to validate their results of the mzML+zlib occupying significantly more space, though our mz5 parser outperformed theirs for full scan and MS/MS data by two orders of magnitude in time.

We were also unable to test several other recent and promising formats. Aird does not report full query times for any of the metrics reviewed here, though they claim their StackZDPD algorithm[28] can improve decompression speed by three times and that the file size is 54% of the vendor file.[16] Similarly, the toffee format for time-of-flight DIA data reports sizes about equal to vendor or 60% of centroid mzml + numpress with query speeds 4 times faster for scans (spectrum-centric, 2 seconds for mzML, 0.5 for toffee) and 100 times faster for chroms (peptide-centric, 168 seconds for mzML, 1.8 for toffee).[8] The Unified Ion Mobility Format (UIMF,[14]) from Pacific Northwest National Laboratory format did not report direct comparisons to any of the available formats and thus we must remain unclear on its performance capabilities.

Fundamental inefficiencies in existing mass-spectrometry formats

We identified several fundamental inefficiencies when writing the parsers. First, scan metadata that was encoded within a scan instead of in a separate unit required looping over each scan to see whether it contained the information requested. Scan number, MS level, and retention time were all useful bits of information that could be included in a file header or footer to relate the three to the data location within the file and allow index use instead of looping over every scan. Second, needing to decode or parse a compressed *m/z*/intensity array in each scan introduced an additional overhead that was especially punishing during ion chromatogram extraction and MS/MS search. While the *m/z* and intensity tuples are an obvious candidate for data compression, this penalty should be of significant concern to engineers. Third, looping over files is inherently slow and introduces additional complexity relative to a single unified database that encodes filename or sample ID as an additional column. A particular strength of the database system we propose is its inherent support for multi-file systems, while all other methods require looping over files.

The problems above highlight an important distinction between data *access* and data *search* that has been largely overlooked in our opinion. While HDF5 files or scan indexes excel at improving data access, they assume that the location of the data is known in advance and can be skipped to via bitwise offsets. If a search is required, however, this advantage is fully negated because each bit of information must be queried anyway. Finally, we must note that scan number is not inherently a useful bit of information. While we included it in our extraction metrics, it is entirely unclear when the scan number itself would be known in isolation. Additionally, this method often confuses the scan number with the scan's indexes in the data structure.

Scan number is not always consecutive (i.e. during polarity switching, multi-experiment samples, or if any filtering is performed during processing), so even if the first or second item in the structure can be queried speedily this is no guarantee that the item will contain the information of interest.

Leveraging robust, future-oriented software development with SQL

The proliferation of MS data storage formats and access algorithms illustrates the general dissatisfaction with existing alternatives to the vendor file or mzML. Formats that are faster to query or smaller on disk tend to be significantly more opaque, and those optimized for a particular method often fail to perform well on other metrics. This complexity is generally expected as optimization tends to require more complex data structures and assumptions about its use but it is not required if the complexity is outsourced to a robust and growing framework such as structured query language (SQL).

SQL is widely used for data processing outside of mass-spectrometry, though its adoption is increasing in recent years. Efforts like mzDB, the Pacific Marine Environment Laboratory's UIMF format[14], and the internals of MzTree hint at SQL's suitability for MS data storage. SQL backends for the next-generation R processing package Spectra now exist[20] and the development of MassQL[22] indicates a growing comfort with SQL syntax for downstream processing, though the language itself strives for human readability in simple queries. The searching and subsetting inherent to MS data exploration represent very simple queries in database space, agnostic to high-level programming language and rarely requiring more than a single line of code. Additionally, the extensive documentation that exists across the internet means that large language models such as ChatGPT are easily able to translate queries for those unfamiliar with SQL's syntax.

Just as the original database paper from[18] argued that the same problems were being solved over and over again, mass spectrometry data scientists are re-solving problems that have been more elegantly ironed out by dedicated teams in computer science and industry with much more extensive support. By leveraging existing optimizations in SQLite and DuckDB, we were able to create a highly performant system for storage of MS data that does not come with significant trade-offs between data extraction methods.

While SQLite is broadly used and its long history testifies to its continued utility, we can use even more modern database methods to improve further upon its analytical processing capacity. We tested both DuckDB the Apache Parquet data formats[23,29] and found that they both performed better than SQLite in disk usage and query speed. DuckDB in particular is nearly a drop-in replacement for SQLite in many cases that's been extensively optimized for MS-related queries given its online analytical processing (OLAP) structure. DuckDB provides automatic compression algorithms and uses zonemaps to create bounding boxes for each subset of data, bringing together existing optimizations from mz5 (delta encoding), mzDB (bounding boxes), and MzTree (axis-agnostic queries) at zero additional cost. Importantly, as with all databases, only the subset of interest needs to be written into memory, making the hardware requirements relatively lightweight.

Of course, to claim that existing frameworks should be discarded in favor of a novel method is to ignore decades of discussion and conversation. We acknowledge that our use case, that of largely exploratory and quality-control steps, is not a universal need and our lack of perfect metadata preservation in particular indicates that databases should become an auxiliary data structure alongside the vendor files or mzMLs, not substitute for them directly. Ultimately, the design decision for mass spectrometry data format will likely continue to be a point of contention and will result from a variety of factors, most crucially 1) initial vendor type, 2) programming language of the developer, 3) types of MS data included (e.g. full scan only versus MS/MS or metadata requirements), 4) whether the entire file will be processed or only a subset, and 5) how well a file type interfaces with downstream software. We intend to show with this manuscript that there is significant overlap between the goals of organizations much larger than any individual lab and that MS as a whole can benefit significantly from co-opting their development.

Note: SQL has integrations in nearly all languages so can be a language agnostic choice.

Conclusion

We propose that a simple relational database is an intuitive and performant mass spectrometry (MS) data storage format. Tables containing fields that map directly to known MS concepts means that adoption is straightforward and facilitated by the widely-understood structured query language (SQL), reducing the code required to extract subsets of interest to a single line. We show that this structure can also take advantage of regular advancements in computer science by leveraging modern data formats such as DuckDB and Parquet to reduce the disk space required while improving access times by 1-2 orders of magnitude. We hope that widespread adoption of this format alongside the metadata-heavy vendor and mzML files will reduce the barriers to data access for mass spectrometrists and provide a consistent framework that covers a majority of the exploratory use cases.

[add R/Julia comparison?]

Acknowledgements

Data availability

References

Supplement

(1) Martens, L.; Chambers, M.; Sturm, M.; Kessner, D.; Levander, F.; Shofstahl, J.; Tang, W. H.; Römpp, A.; Neumann, S.; Pizarro, A. D.; Montecchi-Palazzi, L.; Tasman, N.; Coleman, M.; Reisinger, F.; Souda, P.; Hermjakob, H.; Binz, P.-A.; Deutsch, E. W. mzML—a Community Standard for Mass Spectrometry Data. *Molecular & Cellular Proteomics* **2011**, *10* (1), R110.000133. https://doi.org/10.1074/mcp.R110.000133.

(2) Röst, H. L.; Rosenberger, G.; Navarro, P.; Gillet, L.; Miladinović, S. M.; Schubert, O. T.; Wolski, W.; Collins, B. C.; Malmström, J.; Malmström, L.; Aebersold, R. OpenSWATH Enables Automated, Targeted Analysis of Data-Independent Acquisition MS Data. *Nature Biotechnology* **2014**, *32* (3), 219–223. https://doi.org/10.1038/nbt.2841.

(3) Ting, Y. S.; Egertson, J. D.; Payne, S. H.; Kim, S.; MacLean, B.; Käll, L.; Aebersold, R.; Smith, R. D.; Noble, W. S.; MacCoss, M. J. Peptide-Centric Proteome Analysis: An Alternative Strategy for the Analysis of Tandem Mass Spectrometry Data. *Molecular & Cellular Proteomics* **2015**, *14* (9), 2301–2307. https://doi.org/10.1074/mcp.O114.047035.

(4) Röst, H. L.; Schmitt, U.; Aebersold, R.; Malmström, L. Fast and Efficient XML Data Access for Next-Generation Mass Spectrometry. *PLOS ONE* **2015**, *10* (4), e0125108. https://doi.org/10.1371/journal.pone.0125108.

(5) Bhamber, R. S.; Jankevics, A.; Deutsch, E. W.; Jones, A. R.; Dowsey, A. W. mzMLb: A Future-Proof Raw Mass Spectrometry Data Format Based on Standards-Compliant mzML and Optimized for Speed and Storage Requirements. *Journal of Proteome Research* **2021**, *20* (1), 172–183. https://doi.org/10.1021/acs.jproteome.0c00192.

(6) Wilhelm, M.; Kirchner, M.; Steen, J. A. J.; Steen, H. Mz5: Space- and Time-efficient Storage of Mass Spectrometry

Data Sets. *Molecular & Cellular Proteomics* **2012**, *11* (1), O111.011379. https://doi.org/10.1074/mcp.O111.011379.

(7) Bilbao, A.; Ross, D. H.; Lee, J.-Y.; Donor, M. T.; Williams, S. M.; Zhu, Y.; Ibrahim, Y. M.; Smith, R. D.; Zheng, X. MZA: A Data Conversion Tool to Facilitate Software Development and Artificial Intelligence Research in Multidimensional Mass Spectrometry. *Journal of Proteome Research* **2023**, *22* (2), 508–513. https://doi.org/10.1021/acs.jproteome.2c00313.

(8) Tully, B. Toffee – a Highly Efficient, Lossless File Format for DIA-MS. *Scientific Reports* **2020**, *10* (1), 8939. https://doi.org/10.1038/s41598-020-65015-y.

(9) Askenazi, M.; Ben Hamidane, H.; Graumann, J. The Arc of Mass Spectrometry Exchange Formats Is Long, but It Bends Toward HDF5. *Mass Spectrometry Reviews* **2017**, *36* (5), 668–673. https://doi.org/10.1002/mas.21522.

(10) Shah, A. R.; Davidson, J.; Monroe, M. E.; Mayampurath, A. M.; Danielson, W. F.; Shi, Y.; Robinson, A. C.; Clowers, B. H.; Belov, M. E.; Anderson, G. A.; Smith, R. D. An Efficient Data Format for Mass Spectrometry-Based Proteomics. *Journal of the American Society for Mass Spectrometry* **2010**, *21* (10), 1784–1788. https://doi.org/10.1016/j.jasms.2010.06.014.

(11) Bouyssié, D.; Dubois, M.; Nasso, S.; Gonzalez De Peredo, A.; Burlet-Schiltz, O.; Aebersold, R.; Monsarrat, B. mzDB: A File Format Using Multiple Indexing Strategies for the Efficient Analysis of Large LC-MS/MS and SWATH-MS Data Sets *. *Molecular & Cellular Proteomics* **2015**, *14* (3), 771–781. https://doi.org/10.1074/mcp.O114.039115.

(12) Handy, K.; Rosen, J.; Gillan, A.; Smith, R. Fast, Axis-Agnostic, Dynamically Summarized Storage and Retrieval for Mass Spectrometry Data. *PLOS ONE* **2017**, *12* (11), e0188059. https://doi.org/10.1371/journal.pone.0188059.

(13) Yang, R.; Ma, J.; Zhang, S.; Zheng, Y.; Wang, L.; Zhu,

D. mzMD: Visualization-Oriented MS Data Storage and Retrieval. *Bioinformatics* **2022**, *38* (8), 2333–2340. https://doi.org/10.1093/bioinformatics/btac098.

(14) Beagley, N.; Scherrer, C.; Shi, Y.; Clowers, B. H.; Danielson, W. F.; Shah, A. R. Increasing the Efficiency of Data Storage and Analysis Using Indexed Compression. In *2009 Fifth IEEE International Conference on e-Science*; IEEE: Oxford, United Kingdom, 2009; pp 66–71. https://doi.org/10.1109/e-Science.2009.18.

(15) Römpp, A.; Schramm, T.; Hester, A.; Klinkert, I.; Both, J.-P.; Heeren, R. M. A.; Stöckli, M.; Spengler, B. imzML: Imaging Mass Spectrometry Markup Language: A Common Data Format for Mass Spectrometry Imaging. In *Data Mining in Proteomics*; Hamacher, M., Eisenacher, M., Stephan, C., Eds.; Humana Press: Totowa, NJ, 2011; Vol. 696, pp 205–224. https://doi.org/10.1007/978-1-60761-987-1_12.

(16) Lu, M.; An, S.; Wang, R.; Wang, J.; Yu, C. Aird: A Computation-Oriented Mass Spectrometry Data Format Enables a Higher Compression Ratio and Less Decoding Time. *BMC Bioinformatics* **2022**, *23* (1), 35. https://doi.org/10.1186/s12859-021-04490-0.

(17) Chambers, M. C.; Maclean, B.; Burke, R.; Amodei, D.; Ruderman, D. L.; Neumann, S.; Gatto, L.; Fischer, B.; Pratt, B.; Egertson, J.; Hoff, K.; Kessner, D.; Tasman, N.; Shulman, N.; Frewen, B.; Baker, T. A.; Brusniak, M.-Y.; Paulse, C.; Creasy, D.; Flashner, L.; Kani, K.; Moulding, C.; Seymour, S. L.; Nuwaysir, L. M.; Lefebvre, B.; Kuhlmann, F.; Roark, J.; Rainer, P.; Detlev, S.; Hemenway, T.; Huhmer, A.; Langridge, J.; Connolly, B.; Chadick, T.; Holly, K.; Eckels, J.; Deutsch, E. W.; Moritz, R. L.; Katz, J. E.; Agus, D. B.; MacCoss, M.; Tabb, D. L.; Mallick, P. A Cross-Platform Toolkit for Mass Spectrometry and Proteomics. *Nature Biotechnology* **2012**, *30* (10), 918–920. https://doi.org/10.1038/nbt.2377.

(18) Codd, E. F. A Relational Model of Data for Large

Shared Data Banks. *Communications of the ACM* **1970**, *13* (6), 377–387. https://doi.org/10.1145/362384.362685.

(19) Smith, C. A.; Want, E. J.; O'Maille, G.; Abagyan, R.; Siuzdak, G. XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification. *Analytical Chemistry* **2006**, *78* (3), 779–787. https://doi.org/10.1021/ac051437y.

(20) Rainer, J.; Vicini, A.; Salzer, L.; Stanstrup, J.; Badia, J. M.; Neumann, S.; Stravs, M. A.; Verri Hernandes, V.; Gatto, L.; Gibb, S.; Witting, M. A Modular and Expandable Ecosystem for Metabolomics Data Annotation in R. *Metabolites* **2022**, *12* (2), 173. https://doi.org/10.3390/metabo12020173.

(21) Sud, M.; Fahy, E.; Cotter, D.; Azam, K.; Vadivelu, I.; Burant, C.; Edison, A.; Fiehn, O.; Higashi, R.; Nair, K. S.; Sumner, S.; Subramaniam, S. Metabolomics Workbench: An International Repository for Metabolomics Data and Metadata, Metabolite Standards, Protocols, Tutorials and Training, and Analysis Tools. *Nucleic Acids Research* **2016**, *44* (D1), D463–D470. https://doi.org/10.1093/nar/gkv1042.

(22) Jarmusch, A. K.; Aron, A. T.; Petras, D.; Phelan, V. V.; Bittremieux, W.; Acharya, D. D.; Ahmed, M. M. A.; Bauermeister, A.; Bertin, M. J.; Boudreau, P. D.; Borges, R. M.; Bowen, B. P.; Brown, C. J.; Chagas, F. O.; Clevenger, K. D.; Correia, M. S. P.; Crandall, W. J.; Crüsemann, M.; Damiani, T.; Fiehn, O.; Garg, N.; Gerwick, W. H.; Gilbert, J. R.; Globisch, D.; Gomes, P. W. P.; Heuckeroth, S.; James, C. A.; Jarmusch, S. A.; Kakhkhorov, S. A.; Kang, K. B.; Kersten, R. D.; Kim, H.; Kirk, R. D.; Kohlbacher, O.; Kontou, E. E.; Liu, K.; Lizama-Chamu, I.; Luu, G. T.; Knaan, T. L.; Marty, M. T.; McAvoy, A. C.; McCall, L.-I.; Mohamed, O. G.; Nahor, O.; Niedermeyer, T. H. J.; Northen, T. R.; Overdahl, K. E.; Pluskal, T.; Rainer, J.; Reher, R.; Rodriguez, E.; Sachsenberg, T. T.; Sanchez, L. M.; Schmid, R.; Stevens, C.; Tian, Z.; Tripathi, A.; Tsugawa, H.; Nishida, K.; Matsuzawa, Y.; Van Der Hooft, J. J. J.; Vicini, A.;

Walter, A.; Weber, T.; Xiong, Q.; Xu, T.; Zhao, H. N.; Dorrestein, P. C.; Wang, M. A Universal Language for Finding Mass Spectrometry Data Patterns, 2022. https://doi.org/10.1101/2022.08.06.503000.

(23) Raasveldt, M.; Mühleisen, H. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*; ACM: Amsterdam Netherlands, 2019; pp 1981–1984. https://doi.org/10.1145/3299869.3320212.

(24) Kumler, W.; E. Ingalls, A. Tidy Data Neatly Resolves Mass-Spectrometry's Ragged Arrays. *The R Journal* **2022**, *14* (3), 193–202. https://doi.org/10.32614/RJ-2022-050.

(25) Haug, K.; Cochrane, K.; Nainala, V. C.; Williams, M.; Chang, J.; Jayaseelan, K. V.; O'Donovan, C. MetaboLights: A Resource Evolving in Response to the Needs of Its Scientific Community. *Nucleic Acids Research* **2019**, gkz1019. https://doi.org/10.1093/nar/gkz1019.

(26) Portlock, T.; Shama, T.; Kakon, S. H.; Hartjen, B.; Pook, C.; Wilson, B. C.; Bhuttor, A.; Ho, D.; Shennon, I.; Engelstad, A. M.; Di Lorenzo, R.; Greaves, G.; Rahman, N.; Kelsey, C.; Gluckman, P. D.; O'Sullivan, J. M.; Haque, R.; Forrester, T.; Nelson, C. A. Interconnected Pathways Link Faecal Microbiota Plasma Lipids and Brain Activity to Childhood Malnutrition Related Cognition. *Nature Communications* **2025**, *16* (1), 473. https://doi.org/10.1038/s41467-024-55798-3.

(27) Collette, A.; Tocknell, J.; Caswell, T. A.; Dale, D.; Pedersen, U. K.; Jelenak, A.; Bedini, A.; Raspaud, M.; Jialin; Hole, L.; Johnson, S. R.; Brucher, M.; Teichmann, M.; Vaillant, G. A.; Jakirkham; Hinsen, K.; De Buyl, P.; Huebl, A.; Rathgeber, F.; Verstraelen, T.; Spaghetti Sort; Ebner, S. G.; Smutch; Zwier, M.; Choi, K. Y.; Lee, A.; Tyree, J.; Pascual, C.; Pitrou, A.; Salnikov, A. H5py/H5py: REL: 2.7.1, 2017. https://doi.org/10.5281/ZENODO.877338.

(28) Wang, J.; Lu, M.; Wang, R.; An, S.; Xie, C.; Yu, C. StackZDPD: A Novel Encoding Scheme for Mass Spectrometry Data Optimized for Speed and Compression Ratio. *Scientific Reports* **2022**, *12* (1), 5384. https://doi.org/10.1038/s41598-022-09432-1.

(29) Vohra, D. Apache Parquet. In *Practical Hadoop Ecosystem*; Apress: Berkeley, CA, 2016; pp 325–335. https://doi.org/10.1007/978-1-4842-2199-0_8.