

Modern data transformation re-thought from scratch

Version 1.2, November 23rd 2021

Abstract

This whitepaper describes the reasons behind creating [EasyMorph](#) -- a data transformation and process automation application designed for business needs and IT landscape typical for a modern enterprise.

Despite significant advances in computer technology and drastic changes in the IT landscape¹ over the last 20-30 years, the vast majority of the commonly used data tools are built on concepts from the 80s and 90s of the previous century when business requirements and computer technology were quite different. Unsurprisingly, the efficiency of tools designed for a different era is far from optimal in the modern environment. Most notably, they fail to embrace two major trends: the emergence of the citizen developer and the "no-code" movement, and the proliferation of SaaS and web APIs.

EasyMorph is a result of a thorough re-thinking of what a modern data tool should be. It dramatically simplifies designing ETL² and automation workflows due to a new approach to data transformation enabled by the recent advances in computer technology. This empowers self-service data preparation and automation among non-technical business users who can now automate tedious data-related work routines without involving IT personnel. Because EasyMorph is also successfully used in IT departments for enterprise IT automation, it becomes de facto the common "data language" spoken by non-technical people and IT developers, two groups that frequently have challenges communicating with each other efficiently.

Evolution of the IT landscape

The ongoing digitalization of everything, which started with the invention of the computer, has entered a third wave in which the IT landscape and users' needs drastically differ from the previous stages. A brief overview of this evolution over the last 50 years is necessary for a better understanding of this whitepaper's context.


¹ IT landscape here is the state of computer technology in general and computer systems in use (both hardware and software) at a particular time period.

² Extract, Transform, and Load.

First wave

The first wave is characterized by digitalizing essential paper forms and documents. The primary use case of the first business computers and applications was to move business transactions such as sales orders or inventory movements from paper documents to computer databases³. Hardware was costly, with software typically added for free. The SQL query language was invented in 1974 for querying collected data. It was intended and perceived as the most user-friendly way to query a relational database (and it was, at the time).

Business calculations and modelling became another popular use case for computers. It led to the invention and popularization of spreadsheets. Interestingly, spreadsheets were not intended for loading and analyzing transactional data. Instead, they were viewed as advanced calculators. For instance, the initial version of VisiCalc released in 1979 allowed only 256 rows and 63 columns, not enough for analyzing even the small volumes of the transactional data captured and stored in databases at that time.



	A	B	C	D
	ITEM	NO.	UNIT	COST
1	MUCK RAKE	43	12.95	556.85
2	BUZZ CUT	15	6.75	101.25
3	TOE TONER	250	49.95	12487.50
4	EYE SNUFF	2	4.95	9.90
			SUBTOTAL	13155.50
			9.75% TAX	1282.66
			TOTAL	14438.16

Screenshot 1: VisiCalc.

Non-technical users had simple expectations of business software - store entered data reliably and produce a paper printout with needed numbers⁴. Advanced users wrote SQL queries.

Second wave

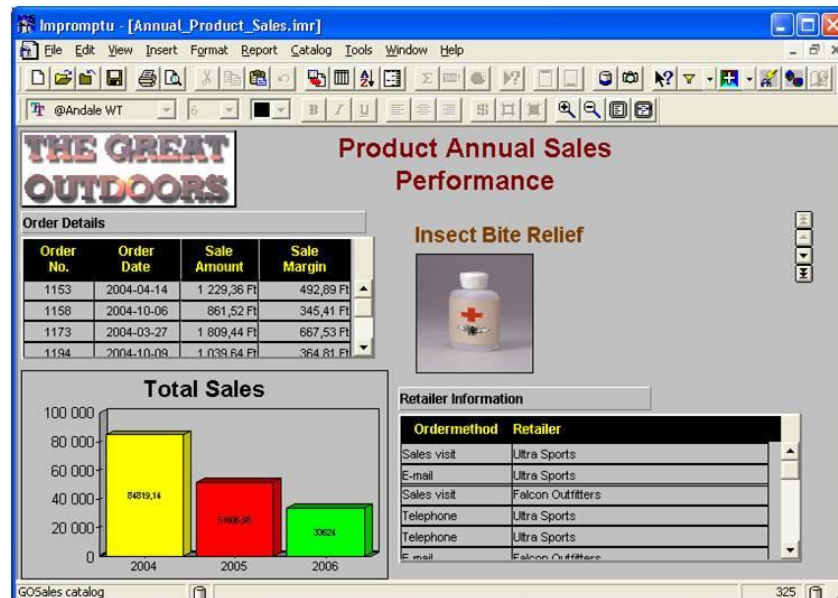
In the 90s, a significant share of essential business transactions had been automated in business applications and captured in databases. The proliferation of operating systems with graphical user interfaces, such as Windows, enabled the breakthrough of new data visualization and analysis applications known as business intelligence (BI) applications and extract, transform, load (ETL) tools.

Unlike the business applications of the first wave, the goal of BI applications (such as Cognos, BusinessObjects, or Tableau) wasn't to capture data and accumulate transactions. Instead, the goal was

³ Transactional row-based databases that were designed and optimized for adding uniform data records as quickly as possible.

⁴ This need hasn't changed much, except printouts have eventually been replaced with PDFs.

to provide a means for analyzing and visualizing the data *already* accumulated in databases and business applications.

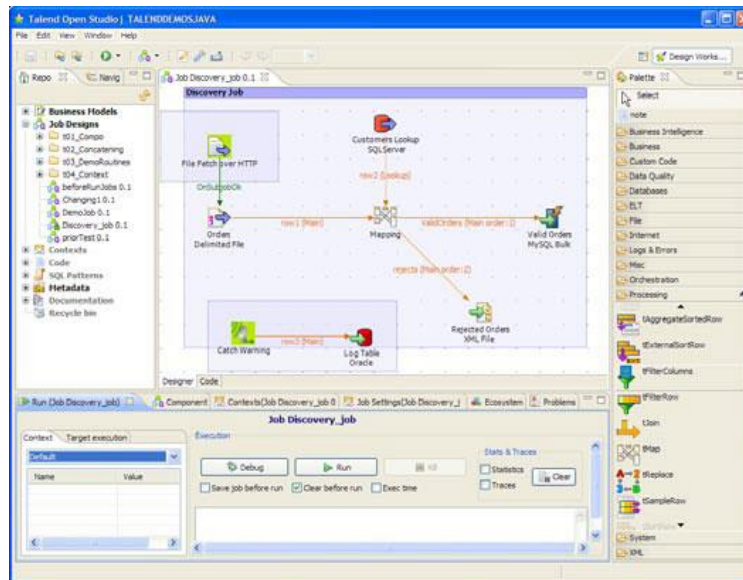


Screenshot 2: Cognos Impromptu.

By this time, a large organization could easily have a few business applications, frequently from different vendors. Each business application stored data in its own database and usually didn't integrate with other applications, which led to the proliferation of data silos. To consolidate data from transactional business applications and make it suitable for data analysis, graphical ETL tools were used to move data into *data warehouses*. A data warehouse is a large-scale centralized database that aggregates and consolidates data from many transactional business applications. It makes data analysis faster, provides a consolidated view of business activities, and removes heavy analytical workloads from transactional business applications. A data warehouse usually has multiple tiers (e.g. staging area, normalized data, and datamarts) with pre-aggregated data and pre-calculated metrics to simplify report generation and data analysis. It was commonly believed that an enterprise data warehouse must become the single source of data for the entire organization. Hence, "Single source of truth" became a popular motto.

While designing data flows visually with ETL⁵ tools was a huge leap ahead in terms of work productivity, it should be noted that they have never been intended for use by non-technical users.

⁵ There is also the subcategory of ELT (Extract, Load, and Transform) tools which have the same purpose as ETL applications. Unlike ETL tools, ELT tools don't have their own data transformation engine. They load data into a database first then leverage the data transformation capabilities of that database (usually via generating a script in SQL or an SQL-based language such as T-SQL or PL/SQL). Otherwise, they are very similar to ETL applications.



Screenshot 3: ETL application Talend.

Business intelligence applications sat on top of data warehouses and business applications and provided users with visual reports and dashboards with various metrics and KPIs calculated over consolidated data.

Excel had become ubiquitous. In contrast to the first wave, business users became more pro-active in data-related tasks. They started enjoying the small degree of freedom and self-service provided by Excel and BI applications. Their expectations from the BI systems and business applications could be described as follows:

- Query databases (data warehouses and business applications) without writing SQL
- Provide necessary data and metrics (already cleansed, consolidated, and aggregated, if needed) in reports and dashboards
- Provide means for basic but interactive data analysis
- Be able to export reports to spreadsheet applications (Excel)

The average data volumes that non-technical users dealt with were well below 1 million records per table. Microsoft Access, a popular desktop database system at that time, had a limit of 2GB per database, and that was considered quite generous and rarely reachable.

The need to analyze large volumes of accumulated data led to the development of *columnar databases*. Unlike the traditional row-based transactional databases, columnar databases were not intended for fast data insertion. Instead, they store data in a compressed form and are optimized⁶ to read and aggregate large datasets, a typical workload in analytical queries.

⁶ Transactional databases store data in rows and therefore are not optimized for analytical workloads that require aggregations over large datasets at once. Columnar databases store data in compressed columns which makes them far more optimized for analytical queries and calculations.

Third (current) wave

Three big trends marked the beginning of the third wave:

- Data lakes
- Software-as-a-Service (SaaS)
- Citizen developers and the no-code movement

Also, three technical innovations:

1. Dramatic price reduction for fibre-optics communications enabled fast internet
2. Massive transition from 32-bit computer architectures to 64-bit
 - Besides performance, this move vastly improved the ability to handle large amounts of data, both in memory⁷ and on disk
 - The 64-bit architecture made it possible to process large amounts of data entirely *in memory*, thus eliminating traditionally slow disk I/O operations; This inceptioned a new category of fast in-memory databases and calculation engines such as in Qlik and EasyMorph
 - The transition enabled the use of the inexpensive x86 CPUs for massive server-based computations instead of RISC CPUs, and pushed cloud computing prices down
3. Massive popularization of multi-core commodity CPUs unlocked the next wave of computation speedups⁸

Faster internet speeds, the maturing of web technology, and cost-effective servers enabled a new kind of software – software-as-a-service (SaaS), also known as *cloud applications*. Cloud applications allowed automating processes of an organization while storing the transactional data from it in a centralized store outside the organization. Data is transferred between the organization and the cloud application over the internet using standard internet technology such as HTTP. The SaaS boom led to the proliferation of cloud storage services and web APIs⁹.

Cloud applications have significantly reduced¹⁰ software TCO (total cost of ownership) for organizations and made business departments less dependent on internal IT departments. This led to an explosion of business applications used in organizations. More internal processes have also become automated. Nowadays, many organizations employ a mix of on-premises applications and cloud applications. Large organizations can use hundreds of applications across different departments – an order of magnitude more compared to the 2nd wave.

⁷ A Windows computer with a 32-bit architecture can have only up to 4 GBs of memory. In comparison, 64-bit Windows 10 can address up to 256 TBs of memory.

⁸ Software applications don't automatically work faster on multi-core CPUs; They must be explicitly designed to employ parallel, multi-threaded computations.

⁹ Application Program Interface – a way to programmatically access and change data in an application.

¹⁰ SaaS has entirely removed hardware costs from consideration. Of course, they are implicitly accounted in SaaS fees, but they are no longer on the radar for customers. A stark difference with the 1st wave when TCO was driven mainly by hardware costs.

Meanwhile, volumes of accumulated data have kept growing exponentially (hence the marketing term "Big Data"). The continuing growth made it obvious that trying to consolidate *all* corporate data in a single enterprise data warehouse becomes counter-productive and sometimes simply not technically feasible. Therefore, instead of copying data from business applications to a data warehouse and permanently consolidating it there, data is left where it was created (or only copied in its raw form, without transformation) and is only consolidated and aggregated on the fly when needed. This data architecture has been called the *data lake*.

Data warehouses still exist in the 3rd wave. They are no longer considered "the single source of truth". Instead, they are viewed as yet another source of data (usually of better quality than raw data).

On-premises business applications, cloud applications, data lakes, and still existing data warehouses have created a landscape where the burden of data consolidation and processing has been greatly shifted onto data consumers, both technical and non-technical.

But are they sufficiently well equipped for it? When it comes to IT departments, the answer is "more or less, yes". However, for non-technical employees, the answer is not that positive. Today's business users have to deal with:

- An ever growing number of cloud and internal business applications from which information must be extracted, merged, and analyzed, frequently in an ad-hoc manner
- Delegating data-related tasks to the IT department is frequently not an option due to the need to have the tasks done quickly
- An increasing number of public datasets on the internet (enabled by open data policies implemented by many governments) that contain valuable, sometimes business-critical, data
- Large volumes of data - datasets with millions of records are no longer considered extremely large
- The existing purpose-built data tools are heavily IT-centric and have a too steep learning curve, so they can't be used in a self-service manner

As the idea of having a corporate "single source of truth" entirely maintained by IT staff turned out utopic¹¹, non-IT employees increasingly find themselves needing to satisfy their own data demands proactively, in a self-service manner. The problem is that business users don't have good software tools because all the tools (or, more precisely, the concepts of these tools) they currently can use were originated in the times when the IT landscape was quite different, and user requirements were very different too. Figuratively speaking, it's like trying to fit a steam engine into a stealth fighter to make it fly.

So, what's wrong with the current data tools available to non-technical people, exactly? Let's look closer at the currently available options.

¹¹ In a large organization there are so many data sources that at any given time some of them will be shutting down while new ones will be being created. The data warehouse can never really keep up. The business cannot wait for this.

What's wrong with the current data tools?

Excel

It's no wonder we'd start the overview of existing popular data tools with Excel. Excel is an ingenious, ubiquitous, and probably the most successful data application ever. People love it for its simple concept, high interactivity, and extreme flexibility. It's not a secret that whole industries run on Excel. The problem with Excel is that it's too good! Due to its versatility and the lack of tools better suited for particular tasks, Excel has become the go-to application when anything data-related has to be done by someone without technical education.

Spreadsheets were invented in the 1970s, almost 50 years ago. At those times, the IT landscape, typical data volumes, and user needs were far different from what we have nowadays:

- Spreadsheets were intended as a general-purpose calculation tool
- Business users worked with datasets that were 2-3 orders of magnitude smaller on average than nowadays
- Even a large organization had only a very few business applications at that time; the need to merge datasets from various systems was nearly non-existent
- The internet, cloud applications and web APIs didn't exist

Of course, over the years, all kinds of features have been added to Excel: charts, pivot tables, VBA scripting, native tables, external data sources, integration with Power BI, etc. The spreadsheet has been put on steroids and then put on steroids again. However, the core concept of the spreadsheet hasn't changed, and it started showing its age a long time ago. No matter what other new shiny feature Microsoft adds to Excel, it will be increasingly incapable of meeting modern demands for data-related automation.

This doesn't mean that spreadsheets are no longer relevant, absolutely not. But it means that spreadsheets should no longer be viewed as the "one size fits all" data tool, especially by non-technical people.

Here is a summary for Excel (without VBA):

Criterion	Verdict
Ease of use	Easy to learn and use, visual and interactive.
Calculation capabilities	Excellent.
Data visualization and analysis	Excellent.
Handling common data volumes	Poor.
Data merging	Poor.
Work automation	Non-existent without VBA.
Extracting data from cloud/internet	Poor.
Writing data back into applications/databases	No.
Cost of mass use in an organization	Affordable.

Scripting

Scripting languages (e.g. VBA in Excel, Python, SAS) and query languages (e.g. SQL) are other options for non-technical people to handle their data-related tasks.

Since any kind of scripting is basically text-based programming (coding), it inherits all the good and bad parts of it. The main advantage of scripting is the extreme flexibility of a programming language. With an appropriate scripting language, basically any data related task can be automated regardless of its complexity. While flexibility is definitely a plus, there are also a few downsides of scripting. The idea of text-based programming originated in the 1960s as a simpler way to generate low-level sequences of CPU instructions. Back then, computers had only text monitors. No graphic visualization was possible. Today, the first challenge is that scripting remains entirely text-based.

Second, because scripting is programming, it requires becoming a programmer, which itself is a full-scale job. The flexibility of scripting can easily become overwhelming for a non-technical user: there are many ways to write the script, meaning there are many ways to make a mistake. When script complexity grows beyond the trivial, scripts created by non-professionals tend to become unreadable, difficult to maintain, and error-ridden.

Third, scripting was never intended to be interactive. If your script fails, you have to re-run the entire script from the beginning, even if the script takes a long time to calculate. This contrasts with Excel, in which a change to a formula results in only the formula and dependent cells to be re-calculated.

Finally, modern multi-core CPU architectures may not come out of the box with scripting, which is historically designed for single-core CPUs and non-parallel execution. Designing parallel computations and parallel workflows in scripts may require additional effort, which can be non-trivial (especially for non-technical people), if possible at all. Again, compare it with a spreadsheet where calculations are executed in parallel, when possible, invisibly to the user.

Here is a summary for scripting:

Criterion	Verdict
Ease of use	Steep learning curve, not visual or interactive.
Calculation capabilities	Excellent. However, parallel computation and concurrent workflows may require additional non-trivial effort or be unavailable at all.
Data visualization and analysis	Complicated, especially anything that requires interactivity.
Handling common data volumes	Excellent for general-purpose scripting languages. For application-specific ones depends on the application.
Data merging	Average. Although can be good in some high-level scripting languages.
Work automation	Excellent for general purpose scripting languages (e.g. Python or VBA), average or poor for application-specific ones (e.g. Qlik or SAS). General-purpose scripts have no built-in

	scheduling.
Extracting data from cloud/internet	Yes.
Writing data back into applications/databases	Yes.
Cost of mass use in an organization	Usually, no extra cost.

ETL tools

ETL applications are built specifically for complex data manipulations and moving data between enterprise applications and databases, so they naturally excel at it. They support a wide range of data sources, file formats and are capable of dealing with large data volumes. The visual design of workflows greatly improves work productivity. However, ETL applications have never been intended for use by a non-technical audience. They are heavily IT-centric and have a steep learning curve.

Originated in the 1990s, almost 30 years ago, the core concept of ETL tools is showing its age. Back then, computers had much less memory (RAM). For instance, 30 years ago when ETL applications were designed, a typical data-processing server had 16MB (megabytes) of RAM. Nowadays, an average laptop has 16GB (gigabytes¹²) of RAM. Just think about it: a typical laptop today has 3 orders of magnitude more RAM than a typical server in times when ETL tools were conceived.

Due to the memory constraints typical for computers in the 90s, ETL applications were designed to process data in *batches* of rows. Instead of loading and processing an entire table in memory (which was technically unfeasible due to small RAM) the workaround was to load and process only a subset of rows (e.g. 1000 rows) at once. It is important to emphasize that the architectural decision to process data in batches was *not* deliberate. It was forced upon ETL software designers by the limited capabilities of the computers of that era. If they weren't limited by the RAM capacities of that time, they would probably have come up with a concept more similar to EasyMorph. Yet, 30 years later, ETL applications (even newly developed ones) still blindly copy the concept that was invented for a different era. Even recently developed ETL software usually has the same "traditional" representation of ETL workflows: boxes connected with arrows.

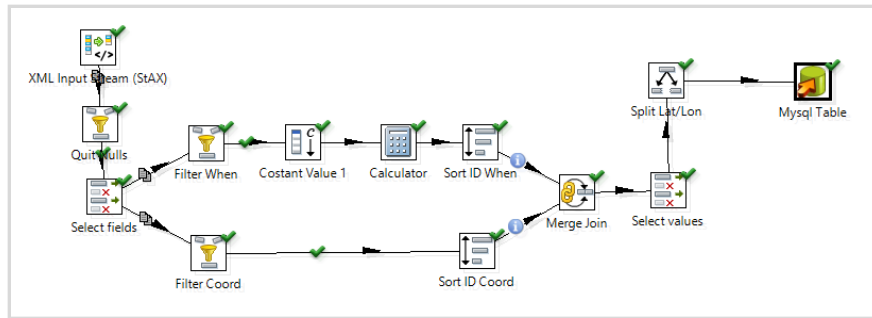
The batch processing of data has a few interesting consequences. First, such ETL tools are not interactive by design. Compare them with Excel, for instance. In Excel, all data is stored in memory, which makes it very interactive. Once a cell or formula is edited, Excel immediately and automatically re-calculates dependent cells without loading anything from the disk. Therefore in Excel, what you see is what you get¹³ -- plain and simple. Not so with ETL tools.

Second, because data is not stored in memory, ETL tools have the same downside as scripting – if something fails, then the entire calculation (workflow) has to be run from the beginning. Such wasted time increases cost of ETL development and negatively affects work productivity.

Third, ETL tools tend to be heavily focused on *dataflow* rather than on data itself. Here is what a typical workflow designed in a traditional ETL tool looks like:

¹² 1 GB = 1024 MB.

¹³ Frequently abbreviated as WYSIWIG.



Screenshot 4: An ETL workflow.

In a typical ETL workflow, individual transformations are depicted as pictograms that are connected with arrows. That type of representation is also known as DAG (Directed Acyclic Graph). Such design has a few significant drawbacks:

First, there is no *data* in this design. It doesn't show any data *entities*. Is the flow above about customers or orders? It's not clear. When the merge happens, what exactly does it merge from a semantic perspective? Order details with order headers? Or maybe customer data from CRM with scoring indexes from a scoring system? It's hard to answer these questions by looking at the dataflow diagram because the main entity here is transformation, not data. And no, some scattered annotations won't solve the problem because annotations are merely a decoration, not an integral part of a semantic model.

Second, profiling data is hard. Answering even basic questions about an ETL workflow can be difficult. What are the names of columns in step #5? What are the unique values in these columns? Data transformation is highly prone to human errors, but because ETL tools don't keep data loaded in memory, profiling and analyzing data at different steps can be tedious. ETL tools frequently can only show a preview of the first 1000 or so rows in a transformation's output. Obtaining the full result of a transformation requires running the whole workflow from the beginning, which can be time-consuming.

Third, a dataflow diagram has poor readability on complex workflows. The workflow depicted above has only 13 steps connected with 13 arrows. What about a workflow with 300 steps? 300 arrows going in different directions, connecting different steps, will look like a chaotic entangled mess¹⁴. Traditional ETL tools provide no means to clearly articulate a workflow structure.

Finally, many ETL tools lack advanced workflow capabilities such as conditional IF...THEN...ELSE branching, or various types of loops. This makes dealing with some real-life scenarios difficult or impossible.

Here is a summary for ETL tools:

Criterion	Verdict
Ease of use	Steep learning curve, visual but not interactive.
Calculation capabilities	Excellent.
Data visualization and analysis	Almost non-existent.

¹⁴ Some ETL tools address the problem using containers that pack multiple transformations into a "black box", but such containers have no semantic meaning and increase obscurity.

Handling common data volumes	Excellent.
Data merging	Excellent.
Work automation	Typically good, although advanced workflows with loops and conditional branching can be impossible in some more simple ETL tools.
Extracting data from cloud/internet	Yes.
Writing data back into applications/databases	Yes.
Cost of mass use in an organization	Expensive, not intended for mass use.

Advanced analytical applications

To address the need of power users for advanced analytical calculations, a new class of applications appeared about 25-30 years ago – advanced analytical applications (e.g. SAS, Alteryx, Lavastorm, KNIME). Some of them adopted visual workflow design from ETL tools, with all the efficiencies and inefficiencies that come with it. Some of them use proprietary high-level scripting languages and inherit all the pros and cons of scripting.

Today, they represent basically more user-friendly ETL applications with additional functionality built on top of it – data visualizations, predictive or geospatial analytics, or advanced statistical calculations. Therefore for the purpose of this article, we will bundle them with ETL tools, keeping in mind that they are not exactly the same.

Re-thinking data transformation from scratch

If the old concepts don't work well for the current 3rd wave of digitalization, what will? EasyMorph focuses on accomplishing three goals:

Address the challenges of the current, 3rd wave of digitalization

As we see it, non-technical business users and departments are facing the following challenges when it comes to data-related tasks:

- Multiplying data sources means data comes in in different formats and needs to be wrangled, merged, and consolidated in order to add value – spreadsheets in shared folders, database extracts, emails and email attachments, enterprise portals (e.g. SharePoint or Confluence), SQL and No-SQL databases, enterprise and cloud applications, web APIs, and government portals.
- Business logic is increasingly less static requiring calculations to be constantly updated and improved
- End users must use a variety of ways to obtain data from various data sources and process it - SQL for databases, VBA in spreadsheets, Python for text files, etc.
- Growing volumes of data call for a casual need to deal with datasets of millions and tens of millions of rows

- The existing data tools have only one or two of these three characteristics: user-friendly, capable, affordable

One more challenge is of an organizational nature: business users and technical departments don't speak the same "data language". They typically have different, not overlapping data toolkits. For instance, a business user may prototype a report in Excel, but it will need to be converted into a T-SQL script by a database developer in order to productionalize the report. The business user can't write T-SQL, and the database developer can't productionalize an Excel spreadsheet in a database. EasyMorph solves this problem because it can be used by both non-IT and IT employees in the same environment. It becomes the common "data language".

Leverage modern computer technology

From a technical perspective, the requirement for EasyMorph was to leverage the advances in modern computer technology available in off-the-shelf computers:

- Multi-core, 64-bit CPUs
- Lots of memory (RAM)
- Larger screens

[Moore's law](#) tells us that in the future consumer computers will have even more CPU cores, more RAM, and larger screens. EasyMorph piggybacks these trends, especially those for RAM and multi-core CPUs. Every year it will work better and better without any additional optimization required.

Keep what works well

While the existing data tools are suboptimal for the current digitalization wave, not everything about them should be dismissed. Certain features have been proven to work well. They are definitely worth learning from. For instance, visualness and interactivity have proven to be mandatory features of a data tool for the non-technical user. In the table below, we've consolidated the summaries of data tools analyzed earlier with two more rows added: "Data always in memory" and "Automatic parallelization". What works well is highlighted. As you can see from the table, EasyMorph combines the best of many worlds:

(Please continue to the next page.)

Criterion	Excel	Scripting	ETL/AA tools	EasyMorph
Ease of use	Easy to learn and use, visual and interactive.	Steep learning curve, not visual or interactive.	Steep learning curve, visual but not interactive.	Low barrier to entry, gradual learning curve. Visual and interactive similarly to a spreadsheet.
Calculation capabilities	Excellent.	Excellent.	Excellent.	Native support for tabular datasets. Business calculations comparable to Excel and SQL. Support for complex business calculations and rules with many conditions.
Data visualization and analysis	Excellent.	Complicated, especially anything that requires interactivity.	Almost non-existent in ETL apps. Good in AA tools.	Extensive means for interactive filtering, visualization, and analysis of data at any point of a calculation chain.
Handling common data volumes	Poor.	Excellent for general purpose scripting languages. For application-specific ones (e.g. VBA) depends on the application.	Excellent.	Effortless handling of millions and tens of millions of records per table on a consumer-grade PC/laptop.
Data merging	Poor.	Average. Although, can be good in some high-level scripting languages.	Excellent.	Merge data from heterogeneous sources – files, spreadsheets, databases, web APIs. Deal with loosely typed data (such as in spreadsheets, XML or JSON) where a column or field may contain values of different types, e.g. text and numbers.
Work automation	Non-existent without VBA.	Excellent for general purpose scripting languages (e.g. Python or VBA), average or poor for application specific ones (e.g. Qlik or SAS). General purpose scripts have no built-in scheduling.	Typically good, although advanced workflows with loops and conditional branching can be impossible in more simple ETL tools.	Automation of daily data-related routines such as file transfer or sending/receiving emails. Triggering actions in external systems applications. Advanced workflow patterns such as conditional branching, loops, and subroutines. Failover recovery. Scheduling.
Extracting data from cloud/internet	Poor.	Yes.	Yes.	Downloading files from the internet. Pulling data from web (REST) APIs and cloud apps.
Writing data back into applications/databases	No.	Yes.	Yes.	Extensive export capabilities to files, databases, and web APIs.
Cost of mass use in an organization	Affordable.	Typically free.	Expensive, not intended for mass use.	Affordable.
Data always in-memory	Yes.	Typically no.	No.	Yes.
Automatic parallelization	Yes.	Non-trivial.	Yes.	Automatic calculation parallelization out of the box.

Table 1: Comparative analysis of data tools.

Legend:

Great quality	Could've been better	Downside
---------------	----------------------	----------

Design concepts of EasyMorph

How exactly does EasyMorph achieve the goals? Let's dive deeper into the design concepts of EasyMorph:

*Tables, not actions (transformation), are the main design unit*¹⁵

What Excel gets right is that business users don't think in terms of transformations when it comes to data preparation. They think in terms of data entities. That's why Excel cells, by default, show data, not formulas.

If you happen to overhear a dialog of two marketing analysts (let's call them Alice and Zak), it may go as follows:

Alice: We need to send a special offer to our customers with low activity.

Zak: How do we know who they are?

Alice: Let's get a list of our customers from that database extract on the shared folder.

Zak: OK. But that includes everyone.

Alice: Do you see the column with the last order date?

Zak: Yes.

Alice: Filter customers that didn't buy anything during the last 3 months.

Zak: Done.

Alice: Now calculate the average order for every customer, and remove those with an average order of less than \$50. They typically don't respond to special offers anyway.

Zak: Oh, I see. Here you go.

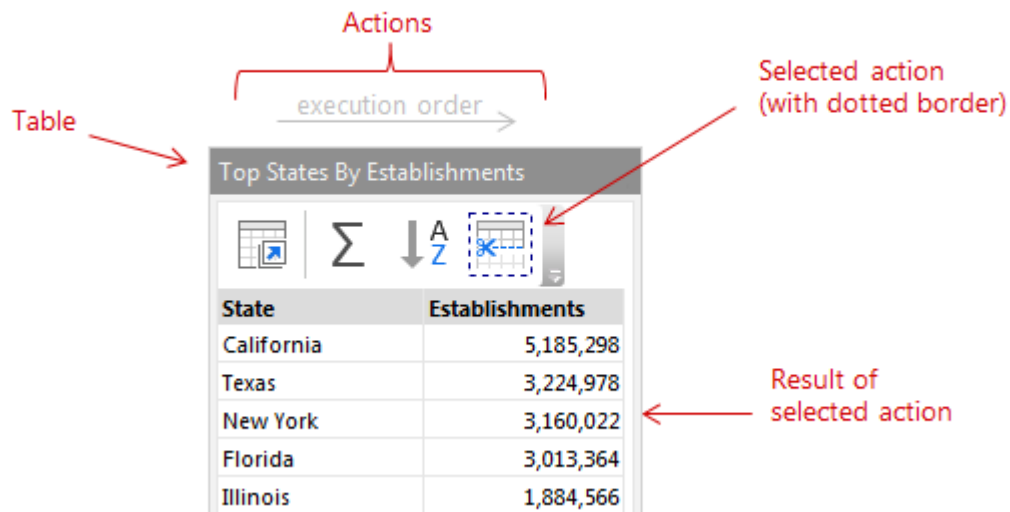
In this fictional conversation, the two persons talk about the same data entity (i.e. customers), despite having 3 transformations applied (filter, calculation, then another filter). This is how non-technical people typically describe data preparation in real life – by explaining how a data entity (e.g. a list of customers) must be modified (or, if you will, *morphed*¹⁶) step by step until it gets to the desired shape.

Also, people tend to intuitively understand when a data entity remains the same entity, albeit modified. They also understand when an entity becomes a semantically different entity (e.g. after aggregation).

To reflect this natural way of thinking, in EasyMorph, the main design element is the table not the transformation. Table with a chain of applied transformations, to be more precise.

¹⁵ The choice for the first concept may be unexpected. However, it's instrumental to understanding other concepts.

¹⁶ Hence the name "EasyMorph".



Screenshot 5: A table in EasyMorph.

Focusing on data entities instead of transformations provides a number of benefits:

- Natural, intuitive look at a data transformation workflow
- Meaningful, semantic context for calculations
- Dense, arrow-less representation of calculations that allows comfortable navigation across tens of tables (data entities) and hundreds of transformations per workflow
- Instant insight into every transformation step of any table (more on that later)
- Meaningful merging because it's merging of semantic entities, rather than outputs of independent transformations

Of course, a calculation may involve multiple data entities. Customer data may eventually be merged with order data, which in turn can be merged with product data, and so on. Calculations may require intermediate (helper) entities that help produce the desired result. For instance, in order to keep only customers with the average order above \$100/month, a helper table with calculated average order amounts may be required and later joined to the table with customers for further filtering.

Key takeaway: The presentation of a calculation algorithm as semantic entities that are morphed step by step reflects the natural thinking process and, therefore, is intuitive for users.

Action chaining

What makes scripts extremely flexible is their *composability* – the ability to put any instruction after any instruction. It's true for spreadsheets, too – any cell can reference any cell in a formula.

EasyMorph borrows the same principle of composability too. In EasyMorph:

- Every action has one and only one output (the action's result)
- Any action can follow any action, even if it doesn't require any input.

- In this case, the output of the previous action automatically becomes the/an input of the next action
- This makes editing workflow easier because actions can be trivially re-ordered by dragging, inserted at any point, or deleted at any point without explicit re-linking to other actions
- External actions (e.g. "Send email") don't transform anything and simply pass the input data to the next action making it possible to insert external actions (needed for automation) anywhere in a calculation.

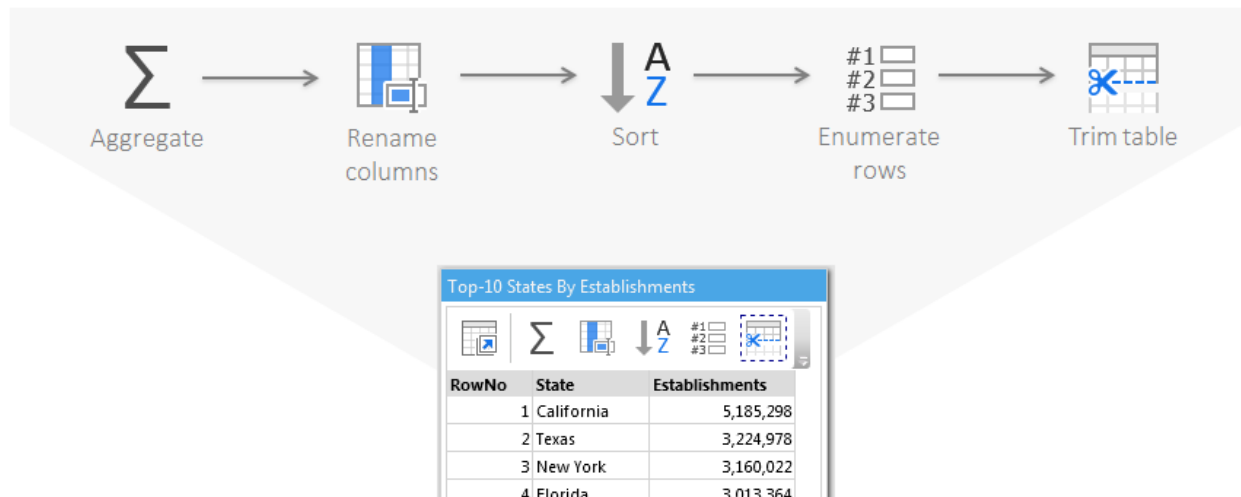


Figure 1: Action chaining in EasyMorph.

Key takeaway: Chaining actions is simple, intuitive, and easy to learn.

Instant access to the full result of every step

When ETL tools were developed in the 90s, the idea of keeping in memory the full output of every transformation step would be considered absurd because the computers of that era had very little memory. That's why the inventors of ETL tools resorted to processing data in batches of rows. However, given a choice everybody would prefer to be able to instantly see and profile the full result of any transformation at any point of a workflow. This would provide great insight into the data and make debugging workflows easier.

Luckily in the 21st century we can afford that luxury. Thanks to Moore's law (that's given us cheap RAM) and a bit of data compression (more on that later) EasyMorph can keep the full result of every action of a workflow in memory (even if it's tens of millions of rows) even on an average laptop. This provides a number of great benefits:

- Instant profiling of any action's result – unique values, counts, data types, etc.
- Easier debugging – walk a workflow back and forth, seeing data being transformed action by action like it was a time machine
- Re-calculation from the point of failure or change, not from the beginning – a huge time-saver

- Intuitive and friendly for the non-technical user

Key takeaway: Keeping all data in memory enables interactivity, provides excellent data analysis capabilities, and significantly reduces time spent on designing workflows.

Reactive calculation (Auto-run)

One of the things that make Excel so intuitive is the reactive calculation (i.e. automatic re-calculation) when a formula or a cell value is changed. Imagine what a hassle it would've been if, after any edit in a spreadsheet, you would have to explicitly trigger re-calculation by clicking a button. Reactive evaluation doesn't have to be explained – people just get it.

Excel keeps all data in memory, which makes fast reactive evaluation possible. EasyMorph keeps all data in memory as well and, similarly to a spreadsheet, performs calculations reactively, on the fly. This capability is called "Auto-run" in EasyMorph and can be disabled when needed.

Reactive calculations are performed in the background (so the user can keep working with the application) and can be triggered automatically in various cases, for instance:

- Action property was edited (e.g. filtering condition changed) - The edited action and all actions that follow it are automatically re-calculated
- Parameter value was changed - All actions in which the parameter is used are re-calculated, and all actions that follow
- A chain of actions changed - Actions were re-ordered, or an action was deleted, or a new action was inserted; As in previous cases, all the actions starting from the point of editing are re-calculated

Not only does reactive evaluation play nicely with data transformations but also with some external actions. For instance, pulling data from a database or a web API can be done reactively because it doesn't cause side effects¹⁷. Typically, any read-only external action can be calculated reactively.

To avoid possible data corruption, reactive calculation stops on actions with side-effects (e.g. exporting to a database, file modification, or sending an email, etc.). Calculation of such actions (and consequent actions) requires manual triggering from the user.

Key takeaway: Keeping all data in memory enables reactive re-calculation, which is intuitive to the end user and provides an excellent user experience.

¹⁷ Of course, in some cases even read-only external operations may produce a significant workload in the external system. In such cases "Auto-run" can be switched off.

A non-restrictive data type system

Tables in EasyMorph consist of named columns, unsurprisingly. However, unlike ETL tools, and similar to spreadsheets, a column in EasyMorph doesn't have to be of a certain data type. Instead, values of different types, e.g. text and numbers, can be mixed in one column. Such

Such a permissive data type system has two significant benefits:

- 1) It is friendlier for people who work with Excel. Excel spreadsheets can have a very complicated structure with nested or overlapping tables, side calculations, multi-line table headers, etc. Enforcing one and only one data type per column when importing a spreadsheet is unnecessary restriction.
- 2) It also simplifies dealing with semi-structured data such as JSON or XML. Processing and merging JSON or XML data from web APIs is a task that didn't exist 30 years ago. ETL tools weren't designed for it. They were designed for moving data between relational databases with a strict data schema. Expecting strict data types in a schema-less data format such as JSON frequently makes no sense and only introduces unnecessary complications and loss of metadata.

A non-restrictive data type system is not new to the industry. For instance, Qlik has a similar data type system, and it's been proven to work well.

Key takeaway: The non-restrictive data type system in EasyMorph is friendlier to the non-technical user than the strict one and is ideal for working with web APIs and semi-structured data such as JSON.

Fixed point decimal numbers

A billion dollar question:

0.1 + 0.2 = 0.3
TRUE or FALSE?

You might be surprised, but the vast majority of reporting, dashboarding, and ETL tools incorrectly returns FALSE¹⁸. Why? Because under the hood, they perform calculations using 64-bit floating-point numbers, which causes certain rounding problems¹⁹. We can safely assume that 99.9% of people who use the tools that fail the test are not aware²⁰ of the errors introduced by the floating-point math. Unfortunately, such errors make their way into reports, dashboards, and databases, usually never discovered.

¹⁸ Excel and Google Sheets use a non-standard 64-bit floating point math and return TRUE. However, the floating point errors became apparent with another expression: $2.03 - 0.03 - 2 = 0$, which returns FALSE in the both applications (EasyMorph returns TRUE).

¹⁹ Humans use decimal numbers but they don't fit nicely into the binary numeral system used in computers. A finite decimal fraction in a base-10 number may become an infinite decimal fraction when converted into the respective base-2 number which is impossible to store in a finite number of bits. It's a well know problem that can be addressed by using a special fixed point math, but for some reason many software vendors ignore it.

²⁰ And that's OK, they shouldn't be aware. They have the right to expect it work as promised.

EasyMorph performs calculations using *128-bit fixed point decimals* and therefore correctly returns TRUE. The fixed point math avoids the errors caused by the 64-bit floating-point math and has the added benefit of more precise number rounding and roll-ups due to the double precision.

Of course, the effective range of floating-point numbers is much wider than that of fixed-point numbers, and that's why floats are great for scientific calculations. However, for business calculations, a huge effective range is not important. The effective range of 128-bit fixed point decimals is $(-7.9 \times 10^{28} \text{ to } 7.9 \times 10^{28}) / (10^0 \text{ to } 28)$, which is 28-29 significant digits with or without decimal point between them. It would suffice for business calculations even in Zimbabwean dollars.

Key takeaway: To be user-friendly and return precise calculations, using fixed-point decimals by a data tool is crucial.

Native tabular datasets

Another cornerstone concept of EasyMorph is that tabular datasets are native to it, just like they are native to relational databases. In other words, EasyMorph has been designed explicitly to deal with tabular data. While it may sound trivial, it's actually not the case with Excel because in the original spreadsheet concept, there are no tables, only a sheet of generic cells. A table in a spreadsheet is *imaginary* because it's just a bunch of non-empty cells put near each other, which makes them look like a table in the imagination of the user. Some cells pretend to be column values. Some cells pretend to be column headers. Cells in Excel are real. Tables are fiction.

Now, an experienced Excel user can object: "Don't the later versions of Excel introduce native tables?" That's true. The concept of table has been introduced in Excel 2010. Although, these tables only represent some mechanisms to make the imaginary abstraction a little bit more real, quasi-real if you will. This new table mechanism is an afterthought to the spreadsheet. No wonder Microsoft doesn't develop this mechanism but instead resorts to table operations (e.g. joins²¹) in PowerQuery, a dedicated data preparation tool in which tables are a native concept.

But why are we even talking about tables and joins? Because in the modern IT landscape, data is increasingly *federated*. It resides in various systems and applications (including cloud ones) inside and outside of the organization. Therefore there is a growing need to merge datasets coming from heterogeneous sources. Typically, these datasets are either tabular (e.g. CSV files) or can be easily converted to tabular (e.g. from JSON or XML).

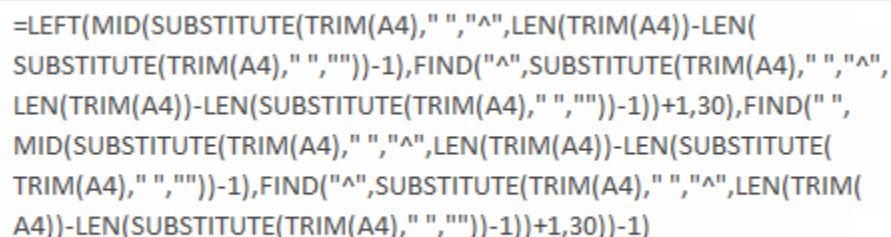
EasyMorph natively operates with tabular datasets. The main calculation unit in EasyMorph is the column, not the cell. Therefore all table operations typical for a relational database (join, union, intersection, subquery, etc.) are natural to EasyMorph. Together with the permissive data type system and reactive calculation, EasyMorph can be thought of as an "Excel for tabular data". The exploratory data analysis capabilities explained later in this article only emphasize that.

²¹ Actually, it is possible to join tables in Excel, but this mechanism is so deeply buried in Excel's UI that the vast majority of Excel users are not aware of it.

Key takeaway: When dealing with multiple source systems, the ability to easily merge data is crucial. A tabular dataset is a cornerstone abstraction in EasyMorph which makes robust data merging (joining) easy.

Simple actions, short formulas

Data workflows and their logic can be quite complicated, which makes them prone to human errors. SQL-queries with hundreds (if not thousands) of lines are not uncommon. Dealing with them is a pain²². Multi-line, entangled, obscure formulas in Excel (such as in the screenshot below) are another bad example of representing complex calculated logic.



```
=LEFT(MID(SUBSTITUTE(TRIM(A4)," ","^",LEN(TRIM(A4))-LEN(SUBSTITUTE(TRIM(A4)," ",""))-1),FIND("^",SUBSTITUTE(TRIM(A4)," ","^",LEN(TRIM(A4))-LEN(SUBSTITUTE(TRIM(A4)," ",""))-1))+1,30),FIND(" ",MID(SUBSTITUTE(TRIM(A4)," ","^",LEN(TRIM(A4))-LEN(SUBSTITUTE(TRIM(A4)," ",""))-1),FIND("^",SUBSTITUTE(TRIM(A4)," ","^",LEN(TRIM(A4))-LEN(SUBSTITUTE(TRIM(A4)," ",""))-1))+1,30))-1)
```

Screenshot 6: An Excel formula.

Poor readability is a major cause of human errors in calculations that end up costing billions in aggregate²³. EasyMorph addresses this problem by making *decomposition* simple by encouraging a decomposition of a complex workflow into smaller calculation steps. Because the result of every step is instantly accessible in EasyMorph, decomposition allows splitting complex logic into several small simple steps, greatly reducing the risk of human error. The ability to see the full result of every action plays very well with the idea of decomposition and provides great insight into calculation logic at every step.

To encourage decomposition, EasyMorph sticks to two principles:

- 1) Simple orthogonal actions.

This principle means that actions must be as simple as possible, and their functionality must not overlap. For instance, the "Aggregate" action doesn't do sorting because there is the "Sort" action, which can always be applied after "Aggregate" if needed. Composability of actions is crucial here.

Simple actions may result in workflows having more actions on average than in a traditional ETL tool. However, it's well compensated by the dense arrow-less action chaining.

- 2) Short formulas.

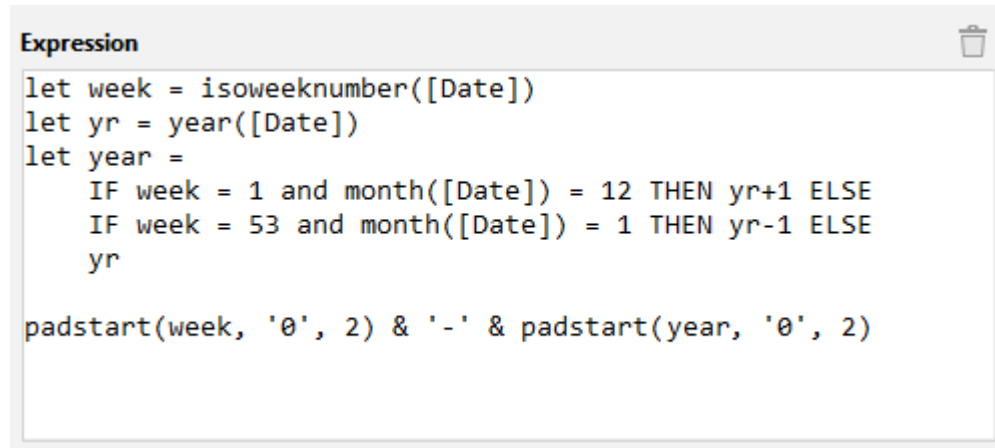
Simple actions enable an efficient decomposition of complex data transformations. The assumption is that complex logic must be performed by actions, not expressions (formulas).

²² One of the main reasons they're a pain is that they're often creating temp tables either implicitly or explicitly and you have to keep these in your head as you're reading the code.

²³ Just a few examples: [link](#)

Shifting calculation complexity to actions makes expressions (formulas) short, simple, and very readable.

Complexity decomposition is possible in expressions too. The formula syntax in EasyMorph has the "let" operator²⁴. The operator can be a part of an expression and allows creating local constants that can be reused in the expression, thus improving readability.



```
Expression
let week = isoweeknumber([Date])
let yr = year([Date])
let year =
  IF week = 1 and month([Date]) = 12 THEN yr+1 ELSE
  IF week = 53 and month([Date]) = 1 THEN yr-1 ELSE
  yr
padstart(week, '0', 2) & '-' & padstart(year, '0', 2)
```

Screenshot 7: The "let" operator in EasyMorph.

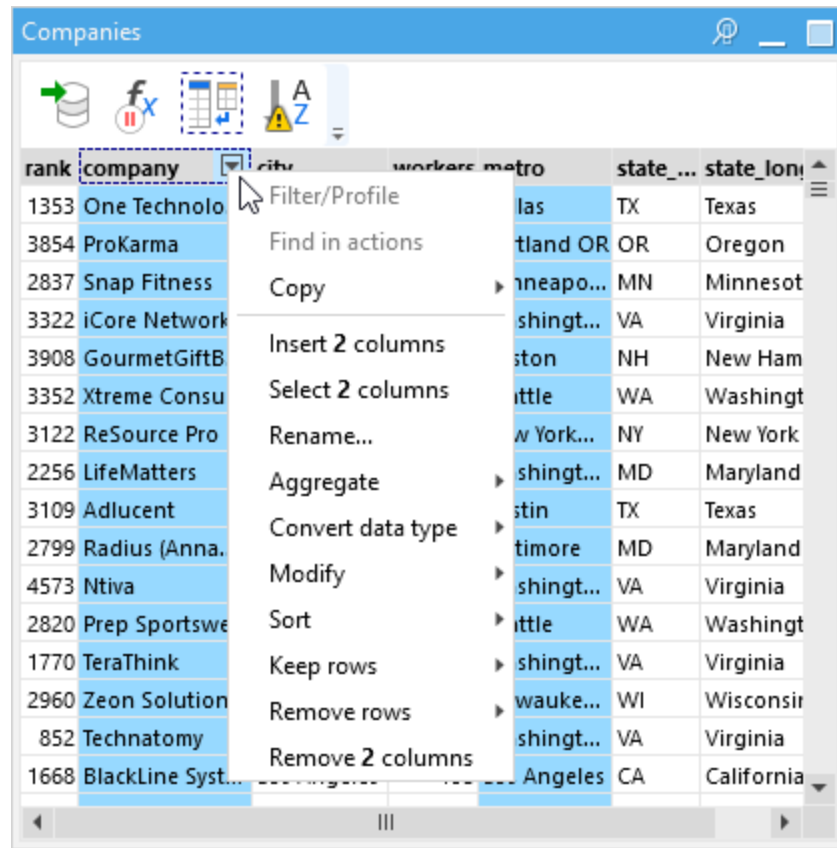
Key takeaway: Complex formulas are hard to understand, which increases the risk of an error, and therefore should be discouraged. By shifting calculation complexity to actions, EasyMorph encourages short, decomposed expressions.

Direct manipulations with data

Once data becomes a part of the design and can be easily viewed and profiled at any point, it makes it possible to directly manipulate data. Again, let's look at Excel. All the data of a sheet is in front of your eyes. Moving a column left or right is trivial – select, copy, and paste. Filtering is trivial – add a filter to a column, click, and select. Users don't program such manipulations. They just do it.

EasyMorph borrows this concept. Because all the data is always available in EasyMorph, it allows certain direct manipulations with data too. For instance, selecting/removing columns, removing/keeping only a particular column value, sorting, aggregation, and a few more operations can be done right via the application's UI (such as context menu or key shortcut). Another example: double-clicking a cell creates a filter that only keeps the clicked values in the column. In all such cases, EasyMorph inserts a necessary action automatically and calculates it on the fly. Designing workflows by directly manipulating data is fast and convenient.

²⁴ The let() function with a similar purpose has been recently added to Excel too.

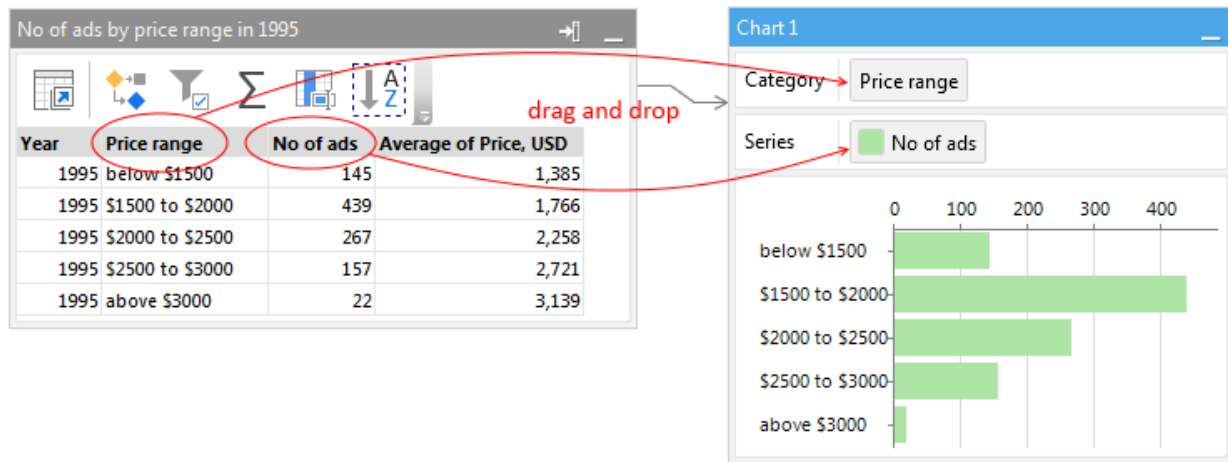


Screenshot 8: Direct manipulations with columns in EasyMorph.

Key takeaway: Keeping all data in memory and interactive re-calculation enables fast, direct manipulation with columns, column values and automatic creation of actions. This simplifies and speeds up workflow design in EasyMorph.

Exploratory data analysis

Reactive calculations and direct manipulations with data enable interactive, exploratory data analysis in EasyMorph. You can develop an ETL flow and explore data at the same time. To support that, EasyMorph incorporates data visualization in the form of interactive charts that can be inserted at any point in a workflow. A chart is linked to a table, which serves as the data source for the chart. Therefore, like any other part of a workflow, charts are also re-calculated reactively, instantly reflecting the result of manipulations with data.



Screenshot 9: Interactive charts in EasyMorph.

The Analysis View is another tool for data exploration and analysis. It is activated when you maximize a table. The Analysis View allows performing instant associative filtering, well known to Qlik users, without creating a filtering action. The filtered state is applied to all actions in the action chain of a table, which greatly simplifies finding root causes of discrepancies and errors.

Screenshot 10: Filtering pane in the Analysis View.

Finally, *sandboxing* is one more tool for ad hoc data analysis in EasyMorph. A sandbox is an isolated table in which you can capture a temporary static copy of any dataset in EasyMorph and then dynamically apply actions and transform data in order to analyze it or answer a question. Because a sandbox is isolated, any data transformations in it don't affect the main workflow. The charts and the Analysis View can be used with sandboxes as with any other table in EasyMorph.

Exploratory data analysis is an example of how EasyMorph combines the best of two worlds. Excel has excellent means of data analysis. However, it struggles to import data from an external data source. ETL tools and scripts can extract data from external systems, but analyzing data in them is a hassle. EasyMorph provides easy to use, powerful means to do both data extraction and exploratory data analysis.

Key takeaway: Keeping all data in memory, interactive re-calculation, and built-in data visualization make EasyMorph a great tool for ad hoc exploratory analysis of data.

Functional programming paradigm

Representing calculation logic as a data entity that is modified step by step by applying various transformations closely resembles *function piping* in functional programming languages. Add to the picture immutable parameters, action composability, iterations²⁵, and the fact that modules in EasyMorph are effectively functions²⁶, and we can see that EasyMorph is basically a visual programming language with a clear functional paradigm.

Reactive calculation (auto-run) also fits nicely into the functional paradigm. Functions (modules) that are "pure", which in terms of functional programming means that they don't have side effects (actions that affect external systems or data, e.g. export to a database) can always be reactively calculated. And the opposite is true, too: if a module has external actions (actions with side effects), then it can't be calculated reactively because it may affect external systems and lead to data corruption or loss. Therefore such modules have to be run explicitly by the user.

Recursion, another basic principle in functional programming, is not implemented in EasyMorph at the moment of writing this article. However, it is technically possible.

Key takeaway: EasyMorph is effectively a visual programming language with a functional paradigm. The flexibility similar to that of a programming language allows designing very complex calculations with EasyMorph.

EasyMorph's in-memory engine

²⁵ Iteration (loop) actions in EasyMorph allow arranging calculations that can be described as iterating, mapping, scanning, and folding in terms of functional programming.

²⁶ A module in EasyMorph can accept parameters and a tabular dataset as inputs (basically, function arguments) and return a tabular dataset (the function result).

At the heart of EasyMorph, there is a highly sophisticated in-memory engine that fully utilizes the recent advances in computer technology, such as the abundance of memory, 64-bit architecture, and multi-core CPUs. In order to enable the new user experience provided by EasyMorph, the engine keeps in RAM all data²⁷ that is being transformed by a workflow. Keeping all data in memory provides a number of significant benefits:

- Slow disk I/O operations are eliminated, thus making computations very fast²⁸
- Edited workflows can be re-calculated from the point of change instead of running the entire workflow from the beginning, which reduces the time necessary to design/edit a workflow
- When data never touches the disk, it is better from a security/privacy standpoint
- In-memory processing piggybacks Moore's law because every year, computers have more and more RAM, thus allowing processing bigger and bigger datasets entirely in memory using the same software

Even when a dataset can't be loaded in memory entirely, there still remains the option of splitting the dataset into chunks (partitions) and processing them chunk by chunk using iterations (loops). While EasyMorph provides means for processing by chunks, they are only required for very large tables and files.

Here is a brief overview of how the in-memory engine works:

Data compression

The ability to instantly see the full result of any action is surely very convenient and provides excellent insight into data transformation logic. For a workflow with, for instance, 100 actions operating on a dataset with 1 million rows and 100 columns, this can mean instant accessing up to $1000000 \times 100 \times 100 = 10$ billion data points. How can EasyMorph store so much data in the RAM of an ordinary laptop? The answer is "data compression". EasyMorph employs various compression techniques that eliminate data redundancy when possible.

The main technique is called "columnar vocabulary compression". Unlike traditional relational databases, EasyMorph stores data not in rows but in columns. This is not a new idea. Columnar databases such as Sybase IQ and Vertica have existed for many years and have shown very good performance in analytical workloads. Cloud data warehouses such as Amazon Redshift and Snowflake also use columnar data storage under the hood.

In a tabular dataset, a column can frequently have only a few distinct values. For instance, a table with international sales orders may have millions of rows, but the column with order currency may have only a few currencies such as USD, EUR, or CAD. In this case, we can create a vocabulary of unique values and

²⁷ It should be noted that EasyMorph Desktop keeps the full result of every action in memory because it provides great workflow designing experience. When workflows are executed in EasyMorph Server the engine consumes less RAM because it doesn't have to keep the result of every action in memory.

²⁸ Accessing data in RAM is 10-100 times faster than accessing data on an SSD disk. For small datasets that fit into the L2 CPU cache the difference can be 10,000 times and more.

a vector of pointers corresponding to each row in the table. A pointer points to the respective vocabulary entry instead of storing it directly in the row.

For instance, the column "Currency" in the table below can be compressed using vocabulary compression. After compression, the column is represented by a vocabulary and a vector of pointers (see below).

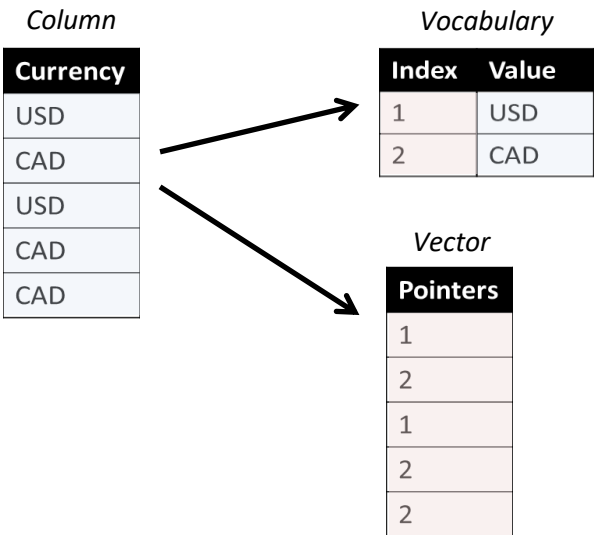


Figure 2: Vocabulary compression.

Vocabulary compression can reduce the memory footprint of a dataset dramatically. On average, the compression ratio is about 3:1. In tables with low cardinality, the compression ratio can easily reach 10:1 or even higher.

Another compression algorithm used in EasyMorph is reducing columns that have only one unique value down to a single constant (which is the unique value). For example, a column may have 1 million values. But if all of them are the same (e.g. "USD"), there is no point in storing 1 million identical values or applying the vocabulary compression. The column can be reduced down to just one constant, which is "USD" in this example.

One more technique used to remove data redundancy is reusing columns. For instance, the "Calculate new column" action adds a new column to a dataset. But the rest of the columns in the dataset remain unchanged. Therefore there is no need to copy them from the action's input to the action's output. We can simply reference them from the output dataset as if they are really there.

Calculations directly on compressed data

It may appear that compressing data is not a good idea for a data transformation application. Does it mean that in order to transform data, we need to decompress data, transform it, and then compress it back? That wouldn't appear very efficient.

It turns out that many common data transformations such as filters, aggregations, lookups, or joins can be easily performed directly on data compressed using vocabulary compression or constant compression, without decompressing and re-compressing it back. More than that, operating directly on compressed data frequently makes transformations *faster*²⁹, not slower. The idea of performing transformations directly on compressed data was proposed by Michael Stonebraker³⁰ and showed great efficiency when it was implemented in Vertica (fast distributed columnar database).

Let's take, for instance, the "Filter" action, which removes rows with selected values of a column. The action works faster on compressed data because instead of comparing every value in every row of a column, only values in the column vocabulary are compared and marked as filtered out. Then the pointers to the marked values are removed from the vector of pointers. Since comparing pointers (which are 64-bit integers) is much faster than comparing text strings or decimal numbers, filtering a compressed column becomes faster than filtering an uncompressed column.

As a bonus, in the newly filtered column, we can reuse (i.e. reference) the vocabulary of the original column instead of creating a new one. So not only do we save time, but we also save memory. Win-win!

Automatic parallelization

The functional programming paradigm, together with the non-linear representation of workflows, enables *automatic parallelization* of calculations. Most programming (and scripting) languages are designed for linear single-threaded calculations because they were invented in times when general-purpose computers usually had only 1 CPU, and that CPU was single-threaded, i.e. all CPU instructions were executed in one and only one predetermined sequence.

Nowadays, even consumer-grade laptops have CPUs with 2, 4, or even more cores, with each core able to perform calculations in two threads at least. Such multi-threaded computer architecture allows splitting a computation into multiple threads and significantly cuts down its overall execution time. While means for parallel computations have eventually been added to scripting (programming) languages, they don't work automatically. Using them requires special programming skills and can be non-trivial. Also, parallel programming is highly prone to human errors that lead to deadlocks³¹ or data loss.

EasyMorph fully benefits from the modern multi-core CPUs. It automatically splits a workflow into independent pipes of actions and executes them in parallel whenever possible (see screenshot below). This helps avoid bottlenecks caused by I/O operations. For instance, in contrast to scripting, reading from a database and importing a file in the same workflow would be typically done in parallel in

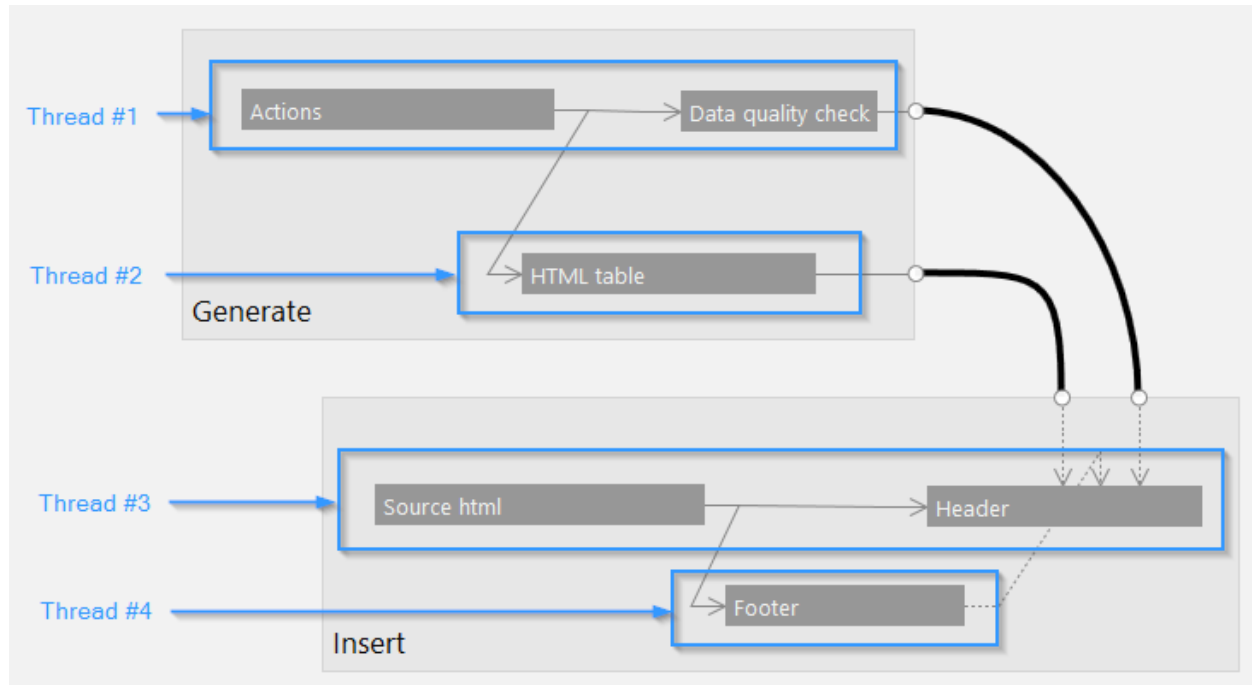
²⁹ Calculations on compressed data are also faster because accessing a smaller volume of data leads to lower cache miss rates in L1-L3 CPU cache thus reducing the number of roundtrips to the main memory.

³⁰ Michael Stonebraker is the "father of relational databases". See https://en.wikipedia.org/wiki/Michael_Stonebraker.

³¹ A deadlock is a situation in parallel programming when a computation freezes because two of its threads have stopped and wait for each other to finish computation. For instance, thread A waits for thread B to return a result in order to continue, but at the same time thread B waits for thread A to do the same. Deadlocks can't be detected automatically and require advanced programming skills to avoid.

EasyMorph in a non-blocking fashion. Also, data transformations (actions) themselves are parallelized when possible.

Parallelization in EasyMorph requires no programming skills, works out of the box, and requires no configuration. It also benefits from Moore's law as it works better and better, with the average number of CPU cores per computer increasing every year.



Screenshot 11: Parallel execution of a workflow in EasyMorph. Each dark grey box represents a table with transformations.

Conclusion

Our journey to re-think, from scratch, ways of working with data has resulted in a rather unexpected finding. As it turns out, the recent advances in computer technology open new opportunities. They allow us to be bolder in our demands and not be restricted by the limitations of the past. The past has given us great technology, ideas, and concepts. However, the current 3rd wave of digitalization brings new challenges and demands that can't be effectively addressed by the tools of the past. More data, more systems, more work to be done create new challenges. New challenges demand new solutions.

An attempt to re-think data transformation from scratch and address the present-day challenges of non-technical (and technical) people has led us to EasyMorph, a contemporary data tool that organically and seamlessly combines not only comprehensive data transformation capabilities but also robust workflow automation and interactive exploratory data analysis. A tool that works for both business users and IT developers and can become the common "data language" that technical and non-technical people communicate in.

But we don't dismiss entirely what was done previously. Quite the opposite! While working on EasyMorph, we tried to consolidate the best from many worlds. We were inspired by the ease of use and intuitiveness of Excel, impressed by the robustness and power of relational databases, enchanted by the beauty of SQL and functional programming, and fascinated by the flexibility of scripting languages. EasyMorph combines the best of many data technologies and offers it in a single product.

If you haven't seen EasyMorph in action yet, go ahead and try the time-unlimited forever free edition. Download it at <https://easymorph.com>.

If you would like to book a demo or just have a chat, contact us at sales@easymorph.com.

Special thanks to Mezan Khaja and Christophe Vogne for reviewing this whitepaper.