Contents

A High Level Overview of the "Why" of our philosophy of data management / data engineering **1**
No hidden agenda here: can we talk about the WHY first, and then we can show the HOW? . . . 1
Something to clear up first: Ingest should be Model actually, but what's in a name? . . . 2
Plain data movement or pure "INGEST" is a ssolved science . . 2
In your experience, what do we need in order to? . . . 2
We want to argue that the better the data modelling, the better the overall solution . . . 3
Why would you say do we fail to model data well? . . . 3
We want to make the case for ELT over ETL, and SQL over general-purpose langs . . . 5
90's - early 00's . . . 5
Late 00's to ~2015 . . . 5
~2015 - present . . . 5
What is your opinion of ETL vs ELT, software engineering rigour applied to analytics (or rather not)? . . . 5
Conclusion . . . 6
Apendix . . . 6
Fun examples of how a name no longer reflects the true meaning of a thing - courtesy of Chat GPT . . . 6
Quotation from the book: "Fundamentals of Data Engineering" . 6
START OF QUOTATION . . . 7

A High Level Overview of the "Why" of our philosophy of data management / data engineering

A recording of this talk is available for you to listen while reading through these notes.

No hidden agenda here: can we talk about the WHY first, and then we can show the HOW?

- Probably no code demos for today, but happy to do those in future
- Would consider today a success if we walk away with a high-level understanding and appreciation.
- Participate, please!

Something to clear up first: Ingest should be Model actually, but what's in a name?

Although we fit these components into the "SIES" framework (Source, Ingest, Enrich, Serve) taxonomically, practically they are more focussed on data modelling and data transformation because the modelling is the more complex and important part.

See at the end of this document for some fun examples of how the true meaning of stuff are sometimes no longer reflected in their names, just as for us, INGEST no longer truely reflects what an "ingest node" actually primarily does.

Plain data movement or pure "INGEST" is a ssolved science

The challenge of pure ingest, idempotent movement of data from one storage system to another, although important, is a solved problem, routine, fairly uninteresting and already industrialised.

Consider:

- ADF (Azure Data Factory) `Copy Action`
- Airflow, Airbyte, Flink, etc.
- Databricks SQL Warehouse `COPY INTO` statement
- Databricks SQL Warehouse `Autoloader`
- All cloud analytics databases making provision for bulk ingest from cloud storage (Snowflake, Databricks, BigQuery, Redshift to name a few)

The problem of effective, fit-for-purpose data modelling for long term storage and usage however is where the true challenge of "ingest" is to be found!

Consider:

- Wide and denormalised tables
- Third Normal Form
- LakeHouse
- Data Vault
- Activity Schema
- Star and Snowflake Schema (Data Warehousing)
- Graph Data modelling

In your experience, what do we need in order to?

- Do data engineering like software engineering: bring the learnings of SE to DE?
- Flatten the learning curve without compromising on requirements / what is ultimately needed?
- Optimise the data engineering process to serve the entire data-intensive solution?
- Reduce cognitive dissonance for developers?

- Develop in a systematic, standardised way?
- Be modular in our thinking?
- Use as much as we can of what has already been developed by others?
- Handle environment promotion (dev, stage, prod)?
- Handle orchestration?
- Handle data modelling?
- Handle testing / data validation/ data observability?
- Encourage engineers to be humble (know thy limits), teachable yet self-driven and motivated?
- Get engineers to reach out and ask, yet not do others work for them?
- Gain parity between running things locally vs running in the cloud?

We want to argue that the better the data modelling, the better the overall solution

- Better data model is more shareable for everyone
- Better data model is more intuitive for everyone
- More intuitive and shareable saves time
- Time saved is extra time for better DS
- Extra time for DS is better data science
- Better data science is better AI
- Better AI is better product
- Better product solves problem
- Better product is happy client
- Happy client whose problem is solved is paying client
- Happy client whose problem is solved is better world
- Paying client is paid Explorers
- Better world - need I say more?

Why would you say do we fail to model data well?

Is it because:

- We have the perception that because storage is cheap, we can just data lake everything?
- We lack the skill / knowledge / know-how?
- We do not make the time?
- We do not HAVE the time to make?
- We lack tools?
- We do not understand the domain?
- We have more important things to worry about?
- We have the perception that if things are slow we can just add compute / storage?
- We do not want to change the raw data and inadvertently throw away something?

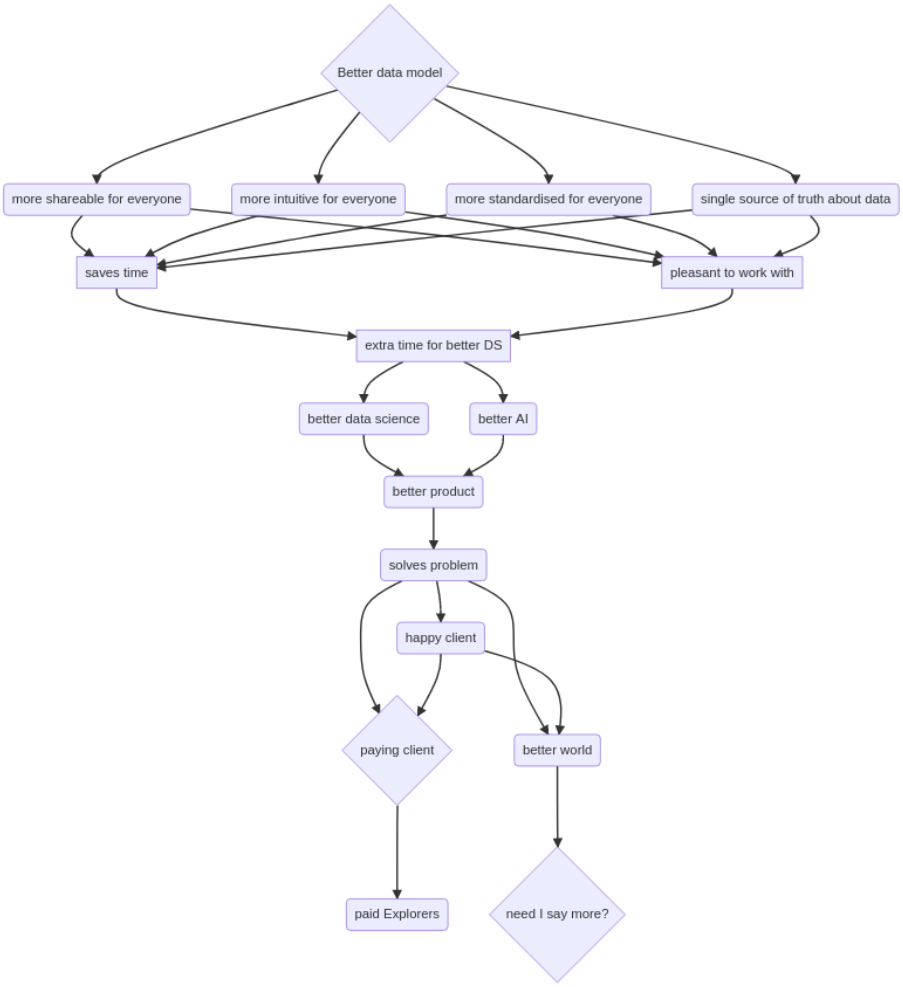

Figure 1: Flow diagram of how better data modelling is better

We want to make the case for ELT over ETL, and SQL over general-purpose langs

A little history anyone?

90's - early 00's

- ETL king
- Storage expensive, slow: optimise for on-prem DW
- The age of data wrangling beyond all recognition...
- ETL purview of DBAs and "visual, flow-based programming" SSIS, Pentaho, Altryx
- Flow-based data movement and transformations leading to spaghetti code mess, big ball of mud
- Low-grade software engineering involved

Late 00's to ~2015

- Birth of "big data"
- Age of Hadoop, Map/Reduce
- Rise of NoSQL: why model data if you can just store JSON?
- ETL now getting more programmatic in map/reduce Java programs etc.
- Complaints that map/reduce programs are hard to write
- Age of the data lake, but fast starting to smell like data swamp
- Level of software engineering is rising in this space!

~2015 - present

- Hadoop honeymoon over: hard to manage, slow, expensive clusters
- Spark as the answer to Hadoop troubles
- NoSQL winter sets in, SQL returns as King!
- Cloud analytics databases optimised for OLAP emerge (Redshift, BigQuery, Snowflake)
- With it, the ELT pattern emerge as more preferable
- With it, SQL as the preferred and universal language
- Even Databricks bringing out Photon and SQL Warehouse
- Data lake exchanged for Lake House, but also just modelling data in cloud analytics database
- Rise of DBT bringing all the SE goodness of years to the land of data analytics
- Level of software engineering in analytics now caught up with general software development

What is your opinion of ETL vs ELT, software engineering rigour applied to analytics (or rather not)?

- we have Databricks, PySpark, Notebooks: what else do we need, really?

- We have PySpark - why should we even consider SQL?
- Data science, data engineering: it's all one thing isn't it?

Conclusion

We hope that these notes and the associated recording of this talk have been helpful in sparking some fresh thoughts around what ever better data engineering might look like. Your feedback is very welcomed, so please get in touch if you have any thoughts, suggestions or ideas on how we can improve or add to this effort.

Apendix

Fun examples of how a name no longer reflects the true meaning of a thing - courtesy of Chat GPT

One example of a phenomenon where the name no longer truly reflects the thing, but the name has stuck, is the case of "jellyfish." Jellyfish are aquatic animals that belong to the phylum Cnidaria, and they are not true fish. However, they were historically named jellyfish because their gelatinous bodies resemble a fish to some extent. Despite the inaccurate name, "jellyfish" has persisted as the commonly used term for these creatures.

Another example is "tin cans." In the early days of canning, food was preserved in cans made of tin-plated steel. However, modern cans are typically made of aluminum or other materials, and tin is rarely used. Nevertheless, the term "tin cans" is still commonly used to refer to metal food containers, even though they may not contain any tin at all.

Similarly, the term "pencil lead" is used to describe the graphite core found in pencils. Historically, graphite was thought to be a form of lead, and the name "pencil lead" has persisted even though the material used in pencils is not lead at all.

Another example is "ringworm." Ringworm is a common fungal infection of the skin that causes a circular rash. Despite its name, it is not caused by a worm but rather by various types of fungi. The name "ringworm" originated from the circular appearance of the rash, which was thought to be caused by a worm under the skin. However, the term "ringworm" has endured even though it is a misnomer.

These examples demonstrate how names can become misleading over time, yet they continue to be used due to historical conventions or widespread familiarity.

Quotation from the book: "Fundamentals of Data Engineering"

Below follows a liberal quotation from Fundamentals of Data Engineering: Plan and Build Robust Data Systems 1st Edition by Joe Reis, Matt Housley in support of the *History* section above.

Note: the first chapter is freely available here as well.

START OF QUOTATION

Now that you have a working definition of data engineering and an introduction to its lifecycle, let's take a step back and look at a bit of history.

Evolution of the Data Engineer

> History doesn't repeat itself, but it rhymes. 'A famous adage often attributed to Mark Twain

Understanding data engineering today and tomorrow requires a context of how the field evolved. This section is not a history lesson, but looking at the past is invaluable in understanding where we are today and where things are going. A common theme constantly reappears: what's old is new again.

The early days: 1980 to 2000, from data warehousing to the web

The birth of the data engineer arguably has its roots in data warehousing, dating as far back as the 1970s, with the business data warehouse taking shape in the 1980s and Bill Inmon officially coining the term data warehouse in 1989. After engineers at IBM developed the relational database and Structured Query Language (SQL), Oracle popularized the technology. As nascent data systems grew, businesses needed dedicated tools and data pipelines for reporting and business intelligence (BI). To help people correctly model their business logic in the data warehouse, Ralph Kimball and Inmon developed their respective eponymous data-modeling techniques and approaches, which are still widely used today. Data warehousing ushered in the first age of scalable analytics, with new massively parallel processing (MPP) databases that use multiple processors to crunch large amounts of data coming on the market and supporting unprecedented volumes of data. Roles such as BI engineer, ETL developer, and data warehouse engineer addressed the various needs of the data warehouse. Data warehouse and BI engineering were a precursor to today's data engineering and still play a central role in the discipline. The internet went mainstream around the mid-1990s, creating a whole new generation of web-first companies such as AOL, Yahoo, and Amazon. The dot-com boom spawned a ton of activity in web applications and the backend systems to support them—servers, databases, and storage. Much of the infrastructure was expensive, monolithic, and heavily licensed. The vendors selling these backend systems likely didn't foresee the sheer scale of the data that web applications would produce.

The early 2000s: The birth of contemporary data engineering

Fast-forward to the early 2000s, when the dot-com boom of the late '90s went bust, leaving behind a tiny cluster of survivors. Some of these companies, such as Yahoo, Google, and Amazon, would grow into powerhouse tech companies. Initially, these

companies continued to rely on the traditional monolithic, relational databases and data warehouses of the 1990s, pushing these systems to the limit. As these systems buckled, updated approaches were needed to handle data growth. The new generation of the systems must be cost-effective, scalable, available, and reliable. Coinciding with the explosion of data, commodity hardware'such as servers, RAM, disks, and flash drives'also became cheap and ubiquitous. Several innovations allowed distributed computation and storage on massive computing clusters at a vast scale. These innovations started decentralizing and breaking apart traditionally monolithic services. The 'big data' era had begun. The Oxford English Dictionary defines big data as 'extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.' Another famous and succinct description of big data is the three V s of data: velocity, variety, and volume.

In 2003, Google published a paper on the Google File System, and shortly after that, in 2004, a paper on MapReduce, an ultra-scalable data-processing paradigm. In truth, big data has earlier antecedents in MPP data warehouses and data management for experimental physics projects, but Google's publications constituted a 'big bang' for data technologies and the cultural roots of data engineering as we know it today. You'll learn more about MPP systems and MapReduce in Chapters 3 and 8 , respectively. The Google papers inspired engineers at Yahoo to develop and later open source Apache Hadoop in 2006.

It's hard to overstate the impact of Hadoop. Software engineers interested in large-scale data problems were drawn to the possibilities of this new open source technology ecosystem. As companies of all sizes and types saw their data grow into many terabytes and even petabytes, the era of the big data engineer was born. Around the same time, Amazon had to keep up with its own exploding data needs and created elastic computing environments (Amazon Elastic Compute Cloud, or EC2), infinitely scalable storage systems (Amazon Simple Storage Service, or S3), highly scalable NoSQL databases (Amazon DynamoDB), and many other core data building blocks.

Amazon elected to offer these services for internal and external consumption through Amazon Web Services (AWS), becoming the first popular public cloud. AWS created an ultra-flexible pay-as-you-go resource marketplace by virtualizing and reselling vast pools of commodity hardware. Instead of purchasing hardware for a data center, developers could simply rent compute and storage from AWS. As AWS became a highly profitable growth engine for Amazon, other public clouds would soon follow, such as Google Cloud, Microsoft Azure, and DigitalOcean. The public cloud is arguably one of the most significant innovations of the 21st century and spawned a revolution in the way software and data applications are developed and deployed. The early big data tools and public cloud laid the foundation for today's data ecosystem. The modern data landscape-and data engineering as we know it now would not exist without these innovations.

The 2000s and 2010s: Big data engineering

Open source big data tools in the Hadoop ecosystem rapidly matured and spread from Silicon Valley to tech-savvy companies worldwide. For the first time, any business had access to the same bleeding-edge data tools used by the top tech companies. Another revolution occurred with the transition from batch computing to event streaming, ushering in a new era of big 'real-time' data. You'll learn about batch and event streaming throughout this book. Engineers could choose the latest and greatest'Hadoop, Apache Pig, Apache Hive, Dremel, Apache HBase, Apache Storm, Apache Cassandra, Apache Spark, Presto, and numerous other new technologies that came on the scene. Traditional enterprise-oriented and GUI-based data tools suddenly felt outmoded, and code-first engineering was in vogue with the ascendance of MapReduce. We (the authors) were around during this time, and it felt like old dogmas died a sudden death upon the altar of big data. The explosion of data tools in the late 2000s and 2010s ushered in the big data engineer . To effectively use these tools and techniques namely, the Hadoop ecosystem including Hadoop, YARN, Hadoop Distributed File System (HDFS), and MapReduce-big data engineers had to be proficient in software development and low -level infrastructure hacking, but with a shifted emphasis.

Big data engineers typically maintained massive clusters of commodity hardware to deliver data at scale. While they might occasionally submit pull requests to Hadoop core code, they shifted their focus from core technology development to data delivery. Big data quickly became a victim of its own success. As a buzzword, big data gained popularity during the early 2000s through the mid-2010s. Big data captured the imagination of companies trying to make sense of the ever-growing volumes of data and the endless barrage of shameless marketing from companies selling big data tools and services. Because of the immense hype, it was common to see companies using big data tools for small data problems, sometimes standing up a Hadoop cluster to process just a few gigabytes. It seemed like everyone wanted in on the big data action. Dan Ariely tweeted , 'Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.' Despite the term's popularity, big data has lost steam. What happened? One word: simplification. Despite the power and sophistication of open source big data tools, managing them was a lot of work and required constant attention. Often, companies employed entire teams of big data engineers, costing millions of dollars a year, to babysit these platforms. Big data engineers often spent excessive time maintaining complicated tooling and arguably not as much time delivering the business's insights and value. Open source developers, clouds, and third parties started looking for ways to abstract, simplify, and make big data available without the high administrative overhead and cost of managing their clusters, and installing, configuring, and upgrading their open source code. The term big data is essentially a relic to describe a particular time and approach to handling large amounts of data. Today, data is moving faster than ever and growing ever larger, but big data processing has become so accessible that it no

longer merits a separate term; every company aims to solve its data problems, regardless of actual data size. Big data engineers are now simply data engineers .

The 2020s: Engineering for the data lifecycle

At the time of this writing, the data engineering role is evolving rapidly. We expect this evolution to continue at a rapid clip for the foreseeable future. Whereas data engineers historically tended to the low-level details of monolithic frameworks such as Hadoop, Spark, or Informatica, the trend is moving toward decentralized, modularized, managed, and highly abstracted tools. Indeed, data tools have proliferated at an astonishing rate (see Figure 1-3). Popular trends in the early 2020s include the modern data stack , representing a collection of off-the-shelf open source and third-party products assembled to make analysts' lives easier. At the same time, data sources and data formats are growing both in variety and size. Data engineering is increasingly a discipline of interoperation, and connecting various technologies like LEGO bricks, to serve ultimate business goals. — #### END OF QUOTATION