Piloting Data Engineering at Berkeley

Joseph M. Hellerstein
University of California, Berkeley
Berkeley, CA, USA
hellerstein@berkeley.edu

Aditya G. Parameswaran
University of California, Berkeley
Berkeley, CA, USA
adityagp@berkeley.edu

ABSTRACT

In the Spring of 2021, we launched a pilot edition of a new Data Engineering course at Berkeley, targeted at our burgeoning Data Science major. We discuss aspects of the design of our first offering of the course, focusing on fluency of data models, languages and transformation tasks.

ACM Reference Format:
Joseph M. Hellerstein and Aditya G. Parameswaran. 2022. Piloting Data Engineering at Berkeley. In *1st International Workshop on Data Systems Education (DataEd'22), June 17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3531072.3535324

1 INTRODUCTION

UC Berkeley has been at the forefront of building a scalable, robust Data Science curriculum for many years [1]. Since the flagship Data8 course [13] was piloted in 2015, the program has quickly developed into a Data Science Major and Minor. In the 2020-2021 academic year, over 3,000 students took Data8; 659 Data Science majors and 345 minors graduated from Berkeley, and this number is expected to grow. Given Berkeley's scale, talented student body, and location in the Bay Area, the impact of this stream of workers is tangible in the industry.

At the time the Data Science program was being designed, Berkeley had built a vibrant AI faculty group, but had only one faculty member whose research and teaching focused on what's often called either Data Management (SIGMOD) or Data Engineering (ICDE) in the research literature. This left limited bandwidth to inject a Data Engineering viewpoint into the curriculum. As a result, the initial vision and curriculum for Data Science at Berkeley started with a bias toward inferential computation [1], reinforcing the popular notion of Data Science as being separate from Data Engineering.

Berkeley has grown its Data Systems and Foundations group substantially since 2015 [10]. At the same time, Data Engineering has gained renewed attention in industry as a necessary foundation for successful Data Science—and a priority for hiring and education according to many anecdotes [2, 5, 8]. With Berkeley's newfound strength in numbers, we set out to develop a Data Engineering course as a core component of the Data Science program at Berkeley.

DataEd'22, June 17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9350-8/22/06.
https://doi.org/10.1145/3531072.3535324

1.1 General Approach

Pedagogically, we took cues from both Data Management foundations and the realities we see in the field based on our local industry experience (e.g., Databricks, Ponder, Trifacta, Turi, etc). The intellectual basis of the course stands on three key themes:

(1) Data Engineering happens in multiple programming paradigms and data models, so Data Engineers need the foundations and the skills to move fluidly across models for data.
(2) Data Transformation (the T of "ETL") covers much of the work in the field, and is different in emphasis and scope to traditional programming or query specification.
(3) Data Engineering traditionally addresses the challenge of robust data management over long timescales. To borrow a modern metaphor [15], Data Engineering is "Data Science integrated over time".

Given constraints on space, this short document focuses largely on the first two of these themes, with a briefer coverage of the last one. The full syllabus of our pilot course is public online [14].

1.2 Class Structure

Due to the Covid-19 pandemic, the initial offering of the course consisted of video-casted faculty lectures (offered both live and asynchronously) and video discussion sections led by TAs. The lecture format was designed to be lively, subject to the constraints offered by Zoom. Lecture content typically involved extensive live examples to illustrate the material, including live-coding, walkthroughs of pre-prepared Jupyter notebooks (shared with students), and live demonstrations of visual tools for data preparation, interactive querying, and charting. This meant that most concepts were distilled quickly into practical form.

While lecture was being delivered by one professor, the other would engage in the live chat stream, which was typically quite active. This included not only questions directed at the lecturer to pause presentation, but also concurrent sidebar discussion concurrent with the lecture, initiated sometimes as an aside from the other professor, and sometimes by students. If the multitasking got too involved, the professor monitoring the chat would pause the lecture for discussion. Anecdotally from course surveys, this multi-channel experience was well-liked by both students and faculty.

The course also involved four major programming projects to drive home the material, which we discuss further in Section 7.

1.3 SQL vs JVM/Python

Our industry experience agreed with conventional wisdom [3] that most Data Engineering shops have a strong bias to either Data Warehouses (SQL-centric) or Big Data platforms (Java/Scala/Python-centric). For coherence, we felt we needed to embed students in only one of these approaches, while firmly acknowledging the other

as a viable option. This left us with the choice of which to make primary. Two main desiderata emerged:

(1) **Pedagogy:** Our prerequisite programming curriculum, designed by our CS colleagues, is heavily biased toward imperative programming in Python and/or Java. We had the choice of doubling down on this worldview, or complementing it with a declarative SQL-centric course; the latter seemed more useful for the students' learning outcomes.
(2) **Practicality:** The market rise of SQL cloud databases was very clear to us: Trifacta business was moving from Spark to BigQuery and Snowflake, Databricks had more uptake with SQL than their RDD or dataframe APIs, and the noise around the SQL-centric "modern data stack" was getting loud. If anything, SQL was emerging as the more prominent choice in the field.

The practicalities assuaged any concern we had that our SQL-centrism was a limited or biased choice. This aligned with our pedagogical goal of exposing students to multiple data models and programming models from the very beginning of the course.

2 INITIAL EXPOSURE TO FORMAL DATA MODELS

Students come into our Data Engineering course with an awareness of three or four data models. They have all seen tensors (or matrices) in prerequisite courses on Linear Algebra and Data Science. They have all used dataframes in Python in prerequisite courses on Data Science. Many have a passing familiarity with relations, and with spreadsheets. But their exposure to these topics varies widely in formality and depth. Fundamentally, students typically have little awareness of the decisions and questions that lie underneath the surface. What does it even mean for two data models to be different? How do the data models that we know differ? How can we transform data from one model to another? What other kinds of data models might exist, and why might they be useful?

Berkeley courses bias toward teaching fundamental concepts that transcend popular technology of the day. So we start with basic formalisms for two core data models and their algebras (tensors and relations), extend our discussion to dataframes, and then cover specific programming interfaces, with an emphasis on SQL.

2.1 The Foundational Models and Algebras

We begin with tensors as a data model, and linear algebra as a core language; this is familiar formal ground for our students. We highlight core data typing issues that often get glossed over: notably that tensor dimensions (the metadata!) come from the domain of natural numbers, while the entries (the data) must come from a single data type—and that data type must be a *field* (obeying the familiar axioms of $+$, $\cdot$ and inverses) in order for linear algebra to "work". We review the notion of dense and sparse matrix representations, with sparse representations as a foreshadowing of relations.

We then transition to a discussion of a branch of math they may not have seen: relations as a data model, and relational algebra as a language. We lean on intuition from tables of data, but highlight the model in classical terms of schema (metadata/types) and instance (data). We introduce the formal relational algebra and its axioms, extended with grouping and aggregation. We contrast the flexibility of relational schemata with the type constraints of tensors.

We then discuss a few example mappings (transformations!) of data from tensors to relations and vice versa. We show how tensors can be represented as relations in various ways, and cases where relations cannot be mapped directly to tensors due to the type constraints of the tensor model. We also show how aggregation is a common pattern for converting from raw facts in relations to summarized statistical features in tensors. This back-and-forth, with its comparisons and contrasts, offers a few important lessons:

- There are multiple foundationally distinct models for representing data. We learned that our students and some of our colleagues were surprised that "relations are math", assuming that databases were an ad hoc "systems" topic.
- With the formalisms side-by-side, students (especially those who are gifted in math) are better able to appreciate the fundamental motivation for multiple data models.
- We observe that the relational model is quite natural for recording observations and computing statistics. We briefly mention the notion of *tidy data*, popularized in the R community as a good design pattern for dataframes, and observe that it is in fact a recasting of the relational model!
- Tensors are the lingua franca of statistical computation, so given the type constraints of tensors, we require data transformations to (often lossily!) coerce "real-world" data into a statistics-capable format. This is just one of many illustrations in the class that data transformation is not just format conversion, it is both a modeling imperative *and* a fateful design space to explore, with potentially profound effects on Data Science outcomes.

Given this introduction to the concepts of distinct data models, we can open the students to consider more varied designs with a strong foundation. If relations and tensors are two distinct models, what other formalisms could exist? In particular, how should we think about the popular notion of dataframes that they use in an ad hoc manner in prerequisite courses? How should we think about spreadsheets? Graphs?

2.2 Dataframes and Beyond

Next we turn our attention to a model that typically lacks a formalism: dataframes. Notably, we tackle the widely-used albeit messy `pandas` package, which is descended from the original dataframes in S. We show students how pandas can be captured in a formal model and algebra for dataframes—taken from our work on Modin [12] — boiling down pandas' 600+ API calls into an algebra of 16 operators without losing expressive power! This large algebra is still something of a kitchen sink spanning relational and linear algebra (and more!), but illustrates clearly why dataframes form a natural "way station" for converting from one of the classical models to the other, and arguably have a basis for existence in their own right.

This triumvirate of Tensor-Relation-Dataframe provides a solid frame for later exposure to new data models. In this first offering of the course we also did a deep dive into JSON and semi-structured data, an informal discussion of spreadsheets, and another informal discussion of knowledge graphs and ontologies. We refer interested readers to the syllabus for more detail [14].

3 TRANSFORMATION IN SQL

Given that Data Transformation is the primary work done in both Data Science and Engineering, we needed to develop a curriculum for teaching Data Transformation in depth, using SQL. This turned out to be quite different from the SQL lessons in our classical Database Systems course, CS186, targeted at CS majors.

Our design for this part of the course was inspired by our experiences in the Data Transformation industry, including experience mapping transformation DSLs down to SQL (in Wrangler [11] and Trifacta), and implementing sampling and sketching libraries in SQL for Apache MADLib [9]. The SQL patterns that arise are fairly different from the Boats and Sailors of our Database Systems course. Our different choices were based on three principles:

(1) **Meet the students' existing skills.** In particular, make students comfortable "porting" their knowledge of Python dataframes to SQL, demonstrating that SQL is a viable and scalable alternative to pandas, and exercising their ability to move fluently across models.
(2) **Preserve exploratory interaction.** Show students how to stay agile and interactive in SQL—even on very large datasets—via techniques like in-query sampling.
(3) **Address the need for speed at scale.** Teach students how to do performance debugging of SQL queries: how to define indexes and materialized views, how to see and influence the behavior of query optimization, etc.

3.1 Example Use Cases

Before we overview our SQL curriculum, it's useful to get a flavor of typical transformation use cases we cover from data exploration, preparation and cleaning. These provide a wealth of requirements that were not covered in our Database Systems course. Here we give three examples; more can be found in the course syllabus.

Sampling. A standard pattern in data science is to extract a sample of a large dataset and load it into a main-memory dataframe on the desktop. SQL offers the opportunity for a more holistic and flexible experience of going from samples to scale. Students are often pleasantly surprised to learn that commodity databases like PostgreSQL offer TABLESAMPLE as a construct. This gives the opportunity to teach Bernoulli sampling. But (among other problems) TABLESAMPLE applies uniformly to all rows in a table, and hence cannot support biasing schemes like stratified sampling. So we also teach reservoir sampling—and show how to implement it via a Table-Valued Function in Python, which they can then combine with joins and `GROUP BY` to implement stratified sampling. This exercises algorithmic thinking in a data engineering context (reservoir sampling is a lovely, handy little algorithm!) It also counters the common bias that SQL is a walled garden of only a few algorithms, but rather a computing framework of significant expressive power for working with data. Having sampling as a SQL "subroutine", they can then do exploratory data analysis over the sampler (which can sample different data each time!) and easily remove the sampler when they're ready to run on all their data—no need to shift environments or languages[1]!

Histogramming. Our students learn the value of visualizing histograms in the initial Data8 course, so writing a query to compute histogram bins and their heights is a natural early task for learning SQL and seeing the benefits of a scalable framework for summarizing large datasets. We start by showing histogram-computing queries that count data in bins of fixed width (a simple `SELECT COUNT(*) ... GROUP BY ... ORDER BY ...` query). Inevitably, though, one wants to know how to have bin boundaries be on quantiles (data-independent, "equi-width" bars), so we show how to write quantile queries as a "subroutine". This motivates the ideas of (a) encapsulating and combining queries via views or common table expressions (CTEs), and (b) window queries, a topic that comes up surprisingly often in Data Science and Engineering, but rarely in undergraduate database courses!

Imputation. As part of data cleaning, it is natural to teach various models for imputing missing values. One of the simplest is to "copy down" data as in a spreadsheet. This again requires window queries. More general schemes for interpolation (even linear!) are more difficult—they require multiple passes of window queries, each as a CTE! Even as experienced SQL users we found this to be a bracing exercise, and we encourage the reader to try it. It also higlights the fact that the relational model and SQL are not ideal for operating on ordered data and in this instance are in stark contrast to the convenience of spreadsheets!

3.2 SQL Curriculum

With those example use cases as background, it is hopefully clear that the course emphasis on different SQL constructs needs to shift from traditional SQL education, which focuses heavily on joins as the workhorse. By contrast, we start with an assumption that single-table queries are the common case, and are the arena for many if not most of the challenging tasks in data transformation. We begin by reviewing the basics of selection and projection (`SELECT ... FROM <table> ... WHERE <simple predicate>`), and move directly to `GROUP BY` and aggregation as core statistical tasks on a single table. This is already enough to begin to show how to convert relational data into tensors, e.g. in a query with integer-valued `GROUP BY` columns and a single float-valued aggregation function.

Simple SQL is also enough of a framework to introduce and exercise a commonly-used skill in data preparation: regular expressions for string analysis and manipulation, including the notion of "capture groups" to extract substrings. This is a standalone, language-agnostic topic that should be taught in any Data Science curriculum, but is not traditionally taught in classes on Databases, Machine Learning or Statistics.

We then introduce extensibility in the form of User-Defined Functions, Aggregates and Table-Valued Functions, making liberal use of PostgreSQL's support for Python extensions. Early exposure to extensibility gets students to view SQL as an open transformation

[1] A more challenging exercise is to implement bootstrap resampling in SQL. The bootstrap is core to our Data8 course, so our students know and love it. Doing it in SQL requires cleverness, as well as extended features like `generate_series` [6].

language that can be easily extended with additional code and access to data outside the database.

The next topic is Window Queries, which occur quite commonly. Time is a near-ubiquitous dimension in data analytics, providing a natural ordering that parameterizes many transformations.

Only after exhausting these "single-table query" issues do we tackle joins, including treatment of `OUTER JOIN` since it is very often needed with dirty and missing data. We also introduce the notion of similarity/proximity scores in the context of text matching, and ways to compare or join data based on similarity metrics.

3.3 Performance Tuning

We devote an entire additional unit to teaching students the basics of database performance tuning. This begins by introducing them to the notion of files and indexes, with just enough detail (i.e. log-based performance in trees, pagination) to help them understand costs and benefits of indexes without, for example, needing to understand the insertion or deletion algorithms of B+-trees. We then show them how to use the `EXPLAIN` facility to see the mapping of SQL to a variant of relational algebra with indexes and scans, and the `EXPLAIN ANALYZE` facility to identify performance bottlenecks. Finally we teach them the basic ideas of query optimization so they understand the decisions made by an optimizer, and techniques for influencing the optimizer's decisions by gathering statistics and "hinting" or constraining the optimizer in its plan search. This material overlaps with the standard database systems canon, but the goal throughout is not to teach students to be able to build a database system, but rather to be able to control one effectively.

4 A BRIEF NOTE ON PIPELINES

As noted above, our third pillar of the course is robust data management over time. This is a hard-earned lesson of the Data Management community going back to the 1970's, with new wrinkles given the more statistical focus of modern applications. The topic potentially covers a wide range of territory. On the statistical side there are issues of temporal dynamics, seasonality and concept drift. On the ML-Operations side there are questions of how to set up pipelines of model training, testing, deployment and live testing, and the feedback loop to model retraining or fine-tuning. In the broader Data Engineering landscape there are topics like scheduling, reliable queueing, stream processing, testing, monitoring, alerting, and data lineage. And there is currently a cacophony of tools, vendors and domain-specific languages in this space.

In our first offering we overviewed the tooling landscape, some best practices, and the foundations of what might be an ideal world—and why that rarely occurs. This included discussion of publish-subscribe and message queue systems (e.g. Kafka, RabbitMQ), workflow systems (e.g. Airflow) and dataflow transformations (e.g. as seen in dbt, Trifacta). Our initial treatment was, frankly cursory—a package of two powerpoint slides on each of these topics.

Then, given the SQL-centric nature of the class, we grounded discussion in the SQL environment. The open-source dbt tool is useful for teaching software engineering practices in a SQL environment. dbt encourages the encapsulation of parameterized SQL CTEs into composable software components. It also encourages authoring of tests and assertions (also in SQL) for data quality, including tests for both schema and data drift. Each of these components is a file in the filesystem that can be versioned in a third-party environment like GitHub, which our students see in earlier courses. We then close out that discussion with an introduction to the practices of Continuous Integration, and the notion of having multiple live environments for development, testing and production, and how data pipelines can graduate from one to the next.

One of the best aspects of a SQL-only environment is that data lineage is very clean. dbt includes a basic webpage that shows the data lineage from base tables through CTEs and queries, leading to discussion of debugging and data quality tests.

We then remind students that this is indeed a "walled garden" view assuming a single SQL platform like a cloud data warehouse. Many (most?) real-world Data Engineering environments involve essentially all the other kinds of systems surveyed above, as well as multiple disparate SQL databases and Big Data runtimes. This leads to a high-level discussions of metadata management and monitoring/testing across systems.

5 DATA TRANSFORMATION TOPICS

Many of the issues described above are embodied in a series of lectures on data transformation, which can be broken down into two subtopics.

5.1 Data Preparation

Under this banner we include data structure and quality assessment as well as transformations between data models. We make use of Trifacta's visual interface in class to make this easy to understand, and more generally encourage students to embrace visualization in this context, as an extension of the Exploratory Data Analysis (i.e. charting) they learn in earlier courses. We focus on scalar numerical transforms as well as hierarchical summarization, and the connection between SQL/OLAP terminology (rollups/drill-downs, groups) and Statistical terminology (quantization, marginalization). We bring in the notion of explicit hierarchies/ontologies and how to combine them with the ideas above, and the use of recursive queries to traverse the hierarchies.

5.2 Data Cleaning

This is often included under the banner of Data Preparation, but the distinction here is a focus on data values, not just structure and model. We start with notions of center (mean vs. median) and dispersal (variance vs. quantiles/MADs) and illustrate the notion of masking and robustness to outliers. We introduce the ideas of trimming and winsorization using quantile-based robust thresholds. We spend time on missing data, data imputation and how it can be implemented in SQL. Finally we overview string distance functions as a tool for addressing misspelling and pragmatic entity resolution. We then show how to implement basic blocking and matching for entity resolution in SQL, and introduce the notion of using more sophisticated models as well.

6 ADDITIONAL TOPICS

Our first offering also included individual overview lectures on a number of topics. Obviously all of these could easily consume

multiple lectures, and we are considering how to weigh these topics over time. This includes:

- ER, FDs and Normalization
- Graph Data, and RDF
- OLAP, Summarization and Visualization
- Transactions
- Storage Formats and Query Processing
- Parallel Data Processing
- Spreadsheets
- Security and Privacy
- Sampling and Sketching.

For some of these topics there are entire courses in the CS department that provide deeper content—notably parallel computing, security/privacy, and approximation algorithms. We hope to see new courses emerge in the Data Science major to focus on the data-centric aspects of some of these issues—this seems especially important for Data Security, Privacy and related societal concerns.

7 PROJECTS AND DATA SETS

Most of the material in the course includes live code examples in Jupyter notebooks (using `ipython-sql` and PostgreSQL). We also show students visual tools including Tableau, Trifacta, DBeaver, dbt, etc. to expand their horizons on the many ways that different users might interact with data. One of our goals in the context of our rather coding-centric Data Science major is to debunk the idea that "real data scientists" stay in code-centric interfaces like terminals, editors and notebooks. We attempt to illustrate tangible the way that visual interfaces can provide significant help to the work practices of technical users.

Modern Berkeley courses in Data Science and Computer Science have to scale up to many hundreds of students, which means that homework projects have to be uniform, auto-gradable, and have clear grading rubrics that keep student complaint rates below 1%. Given these constraints, the students' hands-on experience in the course focuses on four programming-centric assignments:

SQL for Data Engineering. Here we get students connecting Jupyter notebooks to a PostgreSQL backend holding the imdb movie database. The homework exercises data assessment via aggregation and sampling using `TABLESAMPLE BERNOULLI` and `ORDER BY ... LIMIT`, data transformation using regular expressions, views and CTEs, histogram construction at scale using window functions, and joins for combining tables.

Managing Performance. Here we get students evaluating and tuning query performance, via concepts including selection pushdown, index construction, and optimizer hints. The data set used is the Lahman dataset on baseball statistics. Throughout, we have students make use of the PostgreSQL `EXPLAIN ANALYZE` query modifier to assess optimizer decisions and performance bottlenecks.

Data Transformation. Again using SQL, we give students hands-on experience doing data transformation on real data. This project focuses on a quite messy dataset of sensor readings from buildings at UC Berkeley. The core billion-row data table is rife with missing data and outliers. There is a wide range of metadata tables—including imperfectly-scanned text documents—that do not join and group together property. There is also an industry ontology of metadata called Brick [4] modeling the sensors and what their readings mean. The homework requires students to identify outliers, impute missing values, resolve entities based on text similarity, connect disparate tables via outer joins, and navigate ontology hierarchies for rollups/drilldowns.

Working with Semi-Structured Data. The last homework uses the Yelp Academic Dataset (in JSON) and MongoDB to give students experience working with semi-structured data and queries (via MongoDB's MQL language). This exercises students' ability to translate concepts from SQL to a noSQL language, traverse and transform document structures, and use text indexes to enable search of natural language values. Students also gain an understanding for which nesting order is best for each type of query, as well as the challenges in dealing with varying structure across JSON documents, including missing attributes and heterogeneous data types. It also explores, hands-on, pros and cons of three JSON-aware systems they've seen—MongoDB vs PostgreSQL vs pandas—each of which shines in different scenarios.

8 REFLECTION

We have begun planning the next offering of this course, and are beginning our post-mortem debate on the pilot. One idea we're considering is to shift more emphasis to our third theme, long-running data pipelines. The students learn a lot from three models (tensors, relations and dataframes) and their relationships; perhaps they can learn others (semi-structured, graph data, etc.) in the field. Notably we could replace the fourth project on semi-structured data with something involving operationalization, or pipeline debugging. We'd like to improve our projects to connect more visibly to application outcomes, perhaps using social justice data scenarios we're beginning to explore in Berkeley's EPIC lab [7]. Finally, there is an ongoing challenge of balancing theoretical and practical concerns in the class. This is a primary course preparing our Data Science students for work in the field, but it is also the main chance to expose our students and colleagues to the fact that data management topics can be as elegant and intellectually satisfying as any of the topics in the Data Science major.

ACKNOWLEDGMENTS

Thanks to our TAs—Samy Cherfaoui, Aditi Mahajan, Mantej Panesar and Allen Shen—who offered us wise counsel from the student perspective, and tireless energy making the course successful.

REFERENCES

[1] Ani Adhikari, John DeNero, and Michael I Jordan. 2021. Interleaving Computational and Inferential Thinking: Data Science for Undergraduates at Berkeley. (2021).
[2] Jesse Anderson. 2018. Data engineers vs. data scientists. (April 2018).
[3] Jesse Anderson. 2018. The Two Types of Data Engineering. https://www.jesse-anderson.com/2018/06/the-two-types-of-data-engineering/, accessed 03/13/2022.
[4] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh Gupta, Mikkel Baun Kjærgaard, Mani Srivastava, and Kamin Whitehouse. 2016. Brick: Towards a Unified Metadata Schema For Buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments* (Palo Alto, CA, USA) *(BuildSys '16)*. Association for Computing Machinery, New York, NY, USA, 41–50.

[5] Vicki Boykis. 2019. Data science is different now. https://vickiboykis.com/2019/02/13/data-science-is-different-now/, accessed 03/13/2022.
[6] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. 2009. MAD skills: new analysis practices for big data. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1481–1492.
[7] Magdalene L. Crowley. 2021. EPIC Lab receives $2M NSF grant to build tools for criminal justice big datasets. https://eecs.berkeley.edu/news/2021/09/epic-lab-receives-2m-nsf-grant-build-tools-criminal-justice-big-datasets, accessed 05/09/2022.
[8] Mihail Eric. 2021. We Don't Need Data Scientists, We Need Data Engineers. https://www.mihaileric.com/posts/we-need-data-engineers-not-data-scientists/, accessed 03/13/2022.
[9] Joseph M. Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library: Or MAD Skills, the SQL. *Proc. VLDB Endow.* 5, 12 (Aug. 2012), 1700–1711.
[10] Jill Hodges. 2019. UC Berkeley Powers Up to Lead in Data Systems. https://data.berkeley.edu/news/uc-berkeley-powers-lead-data-systems, accessed 03/13/2022.
[11] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 3363–3372.
[12] Devin Petersohn, Stephen Macke, Doris Xin, William Ma, Doris Lee, Xiangxi Mo, Joseph E. Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya Parameswaran. 2020. Towards Scalable Dataframe Systems. *Proc. VLDB Endow.* 13, 12 (jul 2020), 2033–2046.
[13] Berkeley Data 8 Course Staff. 2022. Data 8: The Foundations of Data Science. data8.org, accessed 03/13/2022.
[14] Berkeley Data Engineering Course Staff. 2021. Data Engineering. https://cal-data-eng.github.io/, accessed 03/13/2022.
[15] Titus Winters, Tom Manshreck, and Hyrum Wright. 2020. *Software engineering at Google: Lessons learned from programming over time.* O'Reilly Media, Sebastopol, CA.