Mosaic: An Architecture for Linking Databases and Scalable Interactive Visualizations

Jeffrey Heer
jheer@uw.edu
University of Washington
Seattle, United States

Dominik Moritz
domoritz@cmu.edu
Carnegie Mellon University
Pittsburgh, United States

Ron Pechuk
rpechuk@uw.edu
University of Washington
Seattle, United States

ABSTRACT

Real-time interaction and visualization over large data volumes requires careful coordination of data queries and visual updates. Mosaic is an architecture for optimizing scalable and interoperable visualizations backed by a database, providing a platform for developing and deploying optimizations that span both visualization clients and backing databases. Mosaic applications consist of data-consuming clients that publish data needs as declarative queries, parameterized by shared filtering selections. These queries are managed and automatically optimized by a coordinator that proxies access to a scalable data store. For example, by analyzing selection predicates and client queries, the coordinator automatically constructs materialized views to perform selection updates over pre-aggregated data at interactive rates. Given only a high-level specification, Mosaic automatically enables orders-of-magnitude performance improvements over standard update queries.

KEYWORDS

Interactive data analysis, Scalable visualization, Query optimization, Linked selections, User interfaces, Software architecture

ACM Reference Format:
Jeffrey Heer, Dominik Moritz, and Ron Pechuk. 2018. Mosaic: An Architecture for Linking Databases and Scalable Interactive Visualizations. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

To support effective analysis, interactive data systems should respond to user input at "rates resonant with the pace of human thought" [11], as "milliseconds matter" [2, 6, 17, 27] for user interface latency. Unfortunately, most visualizations fail to provide low-latency interactions with large datasets [3, 26]. Existing methods for low-latency updates to user selections [1, 16, 18–20, 24, 25] typically support only a subset of visualization types (such as scatter plots or time series) and interactions (such as panning and zooming). Moreover, many optimizations are not designed to work together in a single system, and require manual tuning or precomputation.

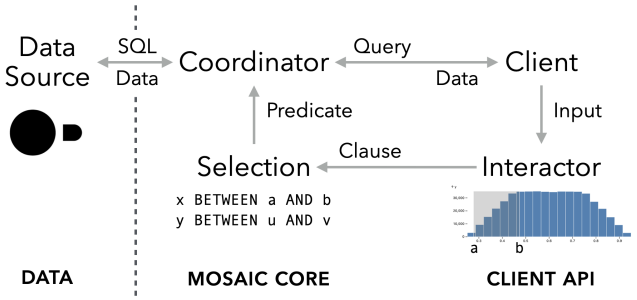

Figure 1: Mosaic architecture overview. A *coordinator* proxies queries to a backing *data source* for one or more data-consuming *client views*. *Selections* broadcast reactive updates for query predicates. Interactions that update selections may be handled directly by a client, or via *interactor* components.

Instead, we contend that data analysts should be able to specify desired views and interactions at a high-level, then benefit from automatic optimizations that provide low-latency interaction over datasets with millions or even billions of records.

Mosaic [10] is an open-source software architecture for linking databases and interactive views. It provides a principled separation of concerns between *client views* that consume data, *interactors* that generate predicate clauses to select data subsets, and *selections* that manage clauses to resolve view-specific filters. Mosaic's selection model translates common operations performed in data analysis interfaces (using inputs such as menus, sliders, tables, and interactive visualizations) to filtering predicates, and enables coordinated updates across interface components. Unlike many prior formulations [5, 9, 22], this model flexibly supports *cross-filtering* in which selections apply to some interface views, but not others.

Since data queries follow predictable patterns [1, 18, 20], the Mosaic *coordinator* can build materialized views automatically to optimize selection update queries. The coordinator optimizes multiple queries using views [7], sidestepping the NP-hard problem of multi-query optimization [13]. In addition to pre-aggregated materialized views, Mosaic supports query caching, query consolidation, and prefetching optimizations. Together, these optimizations can reduce interface latency by orders of magnitude, enabling real-time interaction (under 100ms, and often 1-10ms) over datasets with millions to billions of records. The coordinator manages both the front- and backend and can therefore optimize across data and rendering concerns—each of which are insufficient in isolation [3]. This flexible architecture makes Mosaic an effective platform for developing and deploying optimizations for scalable visualizations.

We demonstrate how Mosaic and its current optimizations support interactive visualizations of diverse datasets, including U.S.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

flight records, New York City taxi trips, and the Gaia star catalog. Using DuckDB [21] as a backing store, Mosaic can be flexibly deployed in web browsers, Jupyter notebooks, or database servers.

2 THE MOSAIC ARCHITECTURE

A Mosaic application consists of data-consuming *client views* registered with a central *coordinator*. Interactions among components are mediated by *params* and *selections*, reactive variables for scalar values and query predicates, respectively. Figure 1 illustrates this architecture. For clarity the figure depicts a single client; Mosaic applications typically include multiple clients with shared selections.

Interactive *selections* indicate records of interest in terms of discrete point values or continuous intervals; these records may then be filtered or highlighted within the interface. Formally, a selection $S = \langle C, R \rangle$ consists of a set of predicate clauses C provided by interactors and a resolution operator $R : \langle C, v \rangle \rightarrow p$. Each clause $c \in C$ corresponds to a boolean-valued selection predicate, as in a SQL `WHERE` clause. The resolution operator R takes a clause set C and a client view v as arguments and produces an output predicate p that can subsequently be applied to filter data for v. The operator R does so by aggregating boolean predicate clauses, for example via intersection or union, including client-sensitive cross-filtering.

Client views publish their data needs as declarative queries. Each client view may be associated with a selection for filtering that client's data. Each client is responsible for incorporating a resolved selection predicate into its query. Selection updates result in new client queries to filter and potentially re-aggregate data.

The coordinator manages client queries, performs optimizations, submits queries to a *data source*, and returns results or errors back to clients. The default configuration uses DuckDB [21] as the backing engine, though in principle other database systems can be used. The separation between clients, selections, and the coordinator enables automatic optimizations that span visualization clients and query generation. By transforming queries before execution, the coordinator enables exploration of optimization techniques without modifications to either client code or the underlying database. The coordinator currently supports query caching, query consolidation (merging compatible queries into a single issued query), and prefetching (clients may speculatively publish queries before results are needed). Critically, the coordinator can also automatically materialize pre-aggregates to optimize selection updates.

2.1 Pre-Aggregated Materialized Views

As datasets grow to millions or more rows, rendering all records becomes infeasible due to both perceptual and computational concerns [18]. Instead, it is common to bin, group, and aggregate data for more scalable displays, from basic histograms to high-resolution raster views. By analyzing queries and a current active selection clause, Mosaic determines appropriate dimensions and pre-aggregated measures to support efficient update queries as selection parameters change. Querying the materialized views can improve performance by multiple orders of magnitude.

To pre-aggregate data automatically, the coordinator (1) checks if a query filtered by an active selection performs aggregation amenable to optimization, (2) determines dimension columns by extracting query `GROUP BY` criteria and possible selection values

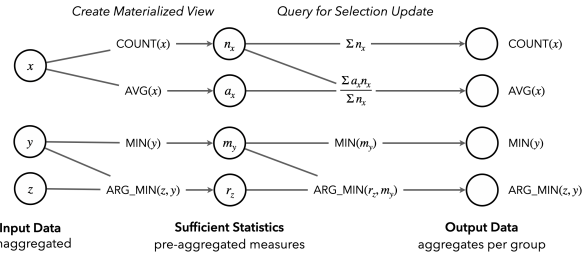

Figure 2: Pre-aggregation and querying for univariate measures. Each sufficient statistic is included as a column in a materialized view, alongside grouping dimensions.

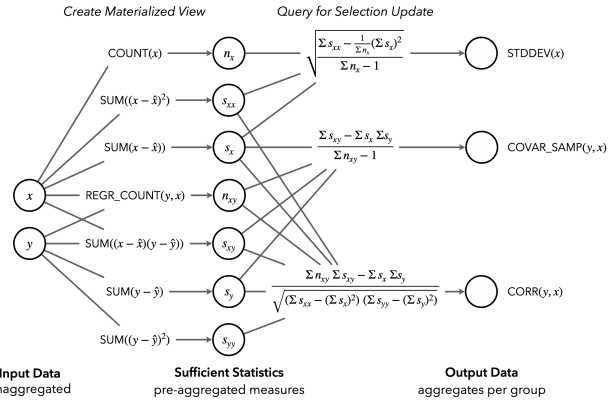

Figure 3: Pre-aggregation and querying for standard deviation and bivariate measures. Each sufficient statistic is included as a column in a materialized view, alongside grouping dimensions. The symbol $\hat{x}$ indicates the average value of x across the full dataset; it is included to mean-center the data to prevent floating point error.

(e.g., pixel-level binning over selection intervals), and (3) creates a materialized view with measure columns containing sufficient statistics for all aggregate query results.

To determine the dimensions of the pre-aggregated materialized view, Mosaic first extracts all columns referenced in the `GROUP BY` clause of a client query. It must then add the dimensions corresponding to an active selection clause. For selections over point values, discrete point values are binned for each column included in the selection. For example, if a clause selects $(A = a_i)$ `AND` $(B = b_j)$, the values of columns A and B are included as materialized view dimensions. For an interval selection clause, the system determines an appropriate pixel-level binning scheme based on screen size and scale transform (linear, log, etc.) metadata provided by an interactor.

Materialized view measure columns consist of *sufficient statistics*: pre-aggregated data, binned by the dimensions, from which the requested aggregates can be constructed. For example, to construct a `COUNT` aggregate, one can apply the `COUNT` function upon materialization, then query the materialized view for updates using the `SUM` function. Functions such as `SUM`, `PRODUCT`, `MIN`, `MAX`, and logical aggregates can be applied for both construction and update queries. In these cases, only a single measure column is needed. Figure 2 shows pre-aggregation schemes for some basic aggregates. Mosaic also optimizes more complex aggregates including standard deviation and regression calculations, as shown in Figure 3.

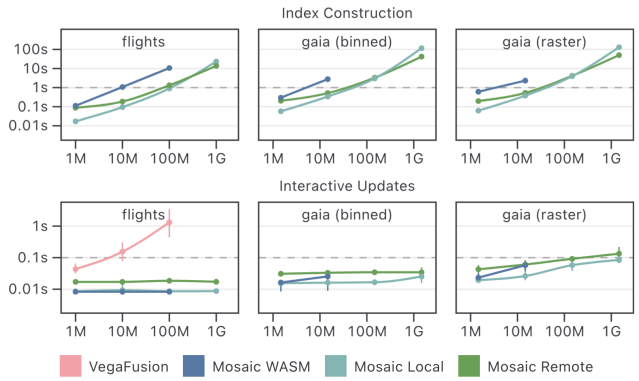

Figure 4: Pre-aggregated materialized view construction (top) and interactive update performance (bottom), from [10]. Median times and interquartile ranges are shown.

2.2 Mosaic Component Libraries

Mosaic includes a set of example client components. Basic inputs include `menu`, `search`, and `slider` widgets. These components publish selection clauses upon interaction and may be configured with values from a backing data table column. The `table` component provides an infinite-scroll grid view over data columns. The grid may be filtered by a *selection*, and the component can publish selection clauses for user-selected rows. To reduce latency, the table component queries for row batches, and sends requests to the coordinator to *prefetch* adjacent off-screen data batches.

For visualization, `vgplot` is a grammar of interactive graphics that combines concepts from prior visualization tools. A *plot* consists of graphical *marks*, each of which is a query-generating Mosaic client. Individual marks span basic shapes (rectangles, areas, lines, symbols) to gridded raster images, and may include associated transformations that are pushed to the database (e.g., binning, aggregation, linear regression, and M4 optimization [12, 14]), or run in the browser (e.g., kernel density estimation [8]). Plots may include *interactors* that listen to input events and generate selection clauses, for example based on clicked points or brushed intervals.

2.3 Performance Benchmarks

Figure 4 shows performance benchmarks from an earlier paper [10]. The tests compare Mosaic using DuckDB-WASM, a local DuckDB server, and a remote server. They were conducted on a 2021 MacBook Pro with an M1 Pro processor. Pre-aggregated materialized views are computed in 1-2 seconds for dataset sizes up to 100M rows; beyond that, precomputation is helpful. Mosaic server configurations maintain interactive update rates (under 100ms) at high data volumes, though begin to degrade when pre-aggregated raster data becomes denser. VegaFusion [15], a state-of-the-art optimizer for Vega [23], is not competitive, as it does not perform pre-aggregation. Mosaic provides low-latency updates with order-of-magnitude improvements over existing web-based visualization tools.

3 DEMONSTRATION

Our demonstration highlights Mosaic's support for interactive data visualizations at scale. Users can interact to visually explore diverse datasets and experiment with system configurations.

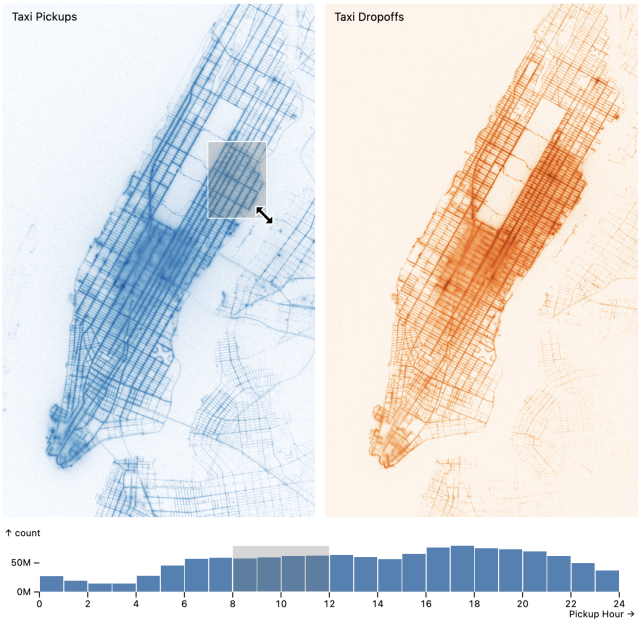

Figure 5: Interactive maps of 1.3B taxi pick-ups and drop-offs in New York City, cross-filtered by pickup time and location.

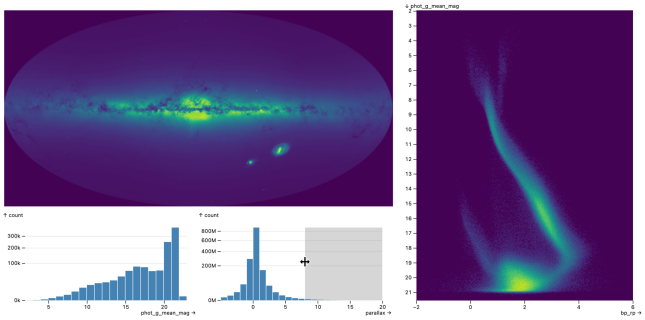

Figure 6: Linked visualizations of over 1.8B stars in the Gaia star catalog. High parallax stars are selected.

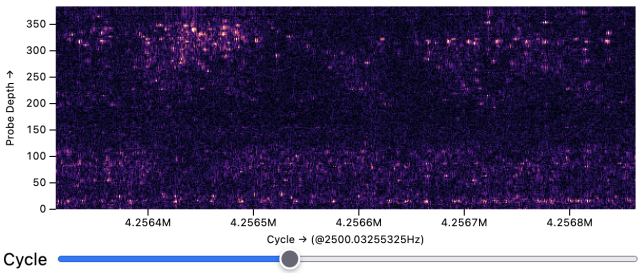

Figure 7: Probe recordings from a mouse brain. Prefetching enables smooth panning over 10.7M time samples (4.1B rows).

3.1 Datasets and Visualizations

To demonstrate Mosaic's capabilities, we include datasets of varying size and complexity, combined with visualizations that range from simple single-view graphs to linked, multi-view dashboards. Below we describe some of these datasets and visualizations.

U.S. Flights: The flight data consists of 44.8M U.S. flight records, with attributes such as arrival delay and departure time. Linked histograms enable dynamic exploration of subsets, such as identifying multivariate patterns through cross-filtering.

New York City Taxi Trips: This dataset describes over 1.3B taxi rides in NYC. These are visualized using two interactive maps representing pickup and drop-off location, alongside a temporal histogram for pickup times (Figure 5). Users can explore both spatial and temporal patterns with linked filtering across all three views.

Gaia Star Catalog: The ESA Gaia catalog contains observations of 1.8B stars, with properties such as location, magnitude, and parallax. Our interactive dashboard (Figure 6) includes a sky map, Hertzsprung–Russell diagram, and associated histograms. All plots are linked via selections, enabling detailed inspection.

Mouse Neuron Spikes: A dataset of mouse brain neural recordings, using a probe with 384 sensor channels across 10.7M time points (4.1B rows). The data is visualized as tiled raster images (Figure 7); prefetching enables smooth panning and zooming for fine-grained exploration of high-density neural signals.

Each visualization is specified declaratively in a YAML format, defining data to load, visualization specifications (marks, interactors, etc.), and linking via selections. Specifications are compiled into JavaScript programs that use the Mosaic and vgplot APIs.

3.2 Demonstration Workflow

Participants first select a dataset and visualization, such as the Gaia star catalog or NYC taxi trips datasets. Then they can use a configuration interface to alter the deployment context (e.g., among DuckDB-WASM in browser or a local DuckDB server), change visualization parameters, and enable or disable optimizations including query caching and pre-aggregated view generation. Each change is reflected in real time, with the GUI displaying generated SQL queries and performance metrics like response times. By adjusting these settings, participants can experience how Mosaic enables interactivity and scalablity in visualizations via automatic optimization, maintaining low-latency updates across diverse datasets and visualization configurations.

4 RELATED WORK & CONCLUSION

Visualization tools such as D3 [4], Vega [23], and Vega-Lite [22] support various visual encodings and interactions, but fail to provide low-latency interaction for larger datasets. VegaFusion [15] and VegaPlus [26] rewrite Vega [23] workflow specifications to query a database, but do not provide pre-aggregation optimizations. Other visualization methods that pre-aggregate data (e.g., [16, 18, 20]) support fewer aggregation functions, are not applied automatically, and some require lengthy precomputation to build indexes.

Mosaic generalizes Vega-Lite's selection model [22] to support a more expressive and extensible set of interactions. It enables low-latency updates (under 100ms, often 1–10ms) and supports "cold start" exploration by pre-aggregating data on the fly for up to hundreds of millions of rows. Mosaic also provides a platform for new optimization techniques; future work could investigate adaptive indexing based on interaction patterns, approximate query processing, or hybrid client/server execution models. The Mosaic architecture, optimizations, and examples are available at idl.uw.edu/mosaic.

REFERENCES

[1] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1363–1375.

[2] Leilani Battle and Jeffrey Heer. 2019. Characterizing Exploratory Visual Analysis: A Literature Review and Evaluation of Analytic Provenance in Tableau. *Computer Graphics Forum* 38, 3 (6 2019), 145–159.

[3] Leilani Battle and Carlos Scheidegger. 2021. A Structured Review of Data Management Technology for Interactive Visualization and Analysis. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2 2021), 1128–1138.

[4] M. Bostock, V. Ogievetsky, and J. Heer. 2011. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (12 2011), 2301–2309.

[5] Hong Chen. 2003. Compound brushing. In *Proc. IEEE Information Visualization*. IEEE Computer Society, Seattle, Washington, 181–188.

[6] Wayne D. Gray and Deborah A. Boehm-Davis. 2000. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied* 6, 4 (2000), 322–335.

[7] Alon Y. Halevy. 2001. Answering queries using views: A survey. *The VLDB Journal* 10, 4 (12 2001), 270–294.

[8] Jeffrey Heer. 2021. Fast & Accurate Gaussian Kernel Density Estimation. In *2021 IEEE Visualization Conference (VIS)*. IEEE, 11–15.

[9] Jeffrey Heer, Maneesh Agrawala, and Wesley Willett. 2008. Generalized selection via interactive query relaxation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.

[10] Jeffrey Heer and Dominik Moritz. 2023. Mosaic: An Architecture for Scalable & Interoperable Data Views. *IEEE Transactions on Visualization and Computer Graphics* (2023), 1–11.

[11] Jeffrey Heer and Ben Shneiderman. 2012. Interactive dynamics for visual analysis. *Commun. ACM* 55, 4 (4 2012), 45–54.

[12] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. M4. *Proceedings of the VLDB Endowment* 7, 10 (6 2014), 797–808.

[13] Tarun Kathuria and S. Sudarshan. 2017. Efficient and Provable Multi-Query Optimization. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, 53–67.

[14] André Kohn, Dominik Moritz, and Thomas Neumann. 2023. DashQL – Complete Analysis Workflows with SQL. (2023).

[15] Nicolas Kruchten, Jon Mease, and Dominik Moritz. 2022. VegaFusion: Automatic Server-Side Scaling for Interactive Vega Visualizations. In *2022 IEEE Visualization and Visual Analytics (VIS)*. IEEE, 11–15.

[16] Lauro Lins, James T. Klosowski, and Carlos Scheidegger. 2013. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (12 2013), 2456–2465.

[17] Zhicheng Liu and Jeffrey Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (dec 31 2014), 2122–2131.

[18] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. 2013. *imMens*: Realtime Visual Querying of Big Data. *Computer Graphics Forum* 32, 3pt4 (6 2013), 421–430.

[19] Haneen Mohammed, Ziyun Wei, Eugene Wu, and Ravi Netravali. 2020. Continuous prefetch for interactive data applications. *Proceedings of the VLDB Endowment* 13, 12 (8 2020), 2297–2311.

[20] Dominik Moritz, Bill Howe, and Jeffrey Heer. 2019. Falcon. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM.

[21] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB. In *Proceedings of the 2019 International Conference on Management of Data*. ACM.

[22] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (1 2017), 341–350.

[23] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (jan 31 2016), 659–668.

[24] Wenbo Tao, Xinli Hou, Adam Sah, Leilani Battle, Remco Chang, and Michael Stonebraker. 2021. Kyrix-S: Authoring Scalable Scatterplot Visualizations of Big Data. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2 2021), 401–411.

[25] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çağatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive Pan/Zoom Visualizations at Scale. *Computer Graphics Forum* 38, 3 (6 2019), 529–540.

[26] Junran Yang, Hyekang Kevin Joo, Sai Yerramreddy, Dominik Moritz, and Leilani Battle. 2024. Optimizing Dataflow Systems for Scalable Interactive Visualization. *Proceedings of the ACM on Management of Data* 2, 1 (mar 12 2024), 1–25.

[27] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. 2017. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Transactions on Visualization and Computer Graphics* 23, 8 (aug 1 2017), 1977–1987.