Big Data Storytelling through Interactive Maps

Jayant Madhavan, Sreeram Balakrishnan, Kathryn Brisbin, Hector Gonzalez, Nitin Gupta,
Alon Halevy, Karen Jacqmin-Adams, Heidi Lam, Anno Langen, Hongrae Lee,
Rod McChesney, Rebecca Shapley, Warren Shen
Google Inc.

Abstract

Google Fusion Tables (GFT) brings big data collaboration and visualization to data-experts who possess neither large data-processing resources nor expertise. In this paper we highlight our support for map visualizations over large complex geospatial datasets. Interactive maps created using GFT have already been used by journalists in numerous high-profile stories.

1 Introduction

Much of the current excitement about Big Data refers to performing analytics over terabytes or petabytes of data. However, this typical focus on compute-intensive tasks ignores the data management needs of a large class of non-technical data experts, who are also relevant to the Big Data generation. Prime examples of such users include journalists, NGOs, social scientists, high school teachers, and other domain data activists. These users have access to large interesting data sets, but do not have the resources nor technical expertise to build their own systems, nor the need to run large-scale analytics. Instead their motivation is centered on advocacy through data and their expertise is comparable to that of advanced spreadsheet users. Among other things, to them Big Data implies the ability to tell their stories through compelling visualizations backed by large amounts of data, often stitched together from varied sources.

Google Fusion Tables (GFT) offers collaborative data management in the cloud for such data experts. We emphasize ease of use for tasks such as sharing, collaboration, exploration, visualization and web publishing. We support interactive visualizations, such as maps, timelines, and network graphs, that can be embedded on any web property. Users can update data and those updates will propagate to all uses of the data. We also enable users to easily find and reuse related data sets thereby enabling the integration of data from a myriad of producers.

Thus far, GFT has received considerable use among journalists who embed customized map visualizations as part of their articles. Maps created using GFT are regularly featured in articles published by prominent news sources such as the UK Guardian, Los Angeles Times, Chicago Tribune, and Texas Tribune. In this paper we focus on the scalable serving infrastructure that we built to support interactive maps over large complex geospatial data sets. Not only can viewers zoom, pan, and click on map features, the map presentation can be customized on the fly, reflect near-real time changes to the underlying data, and show the results for a potentially arbitrary viewer query.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

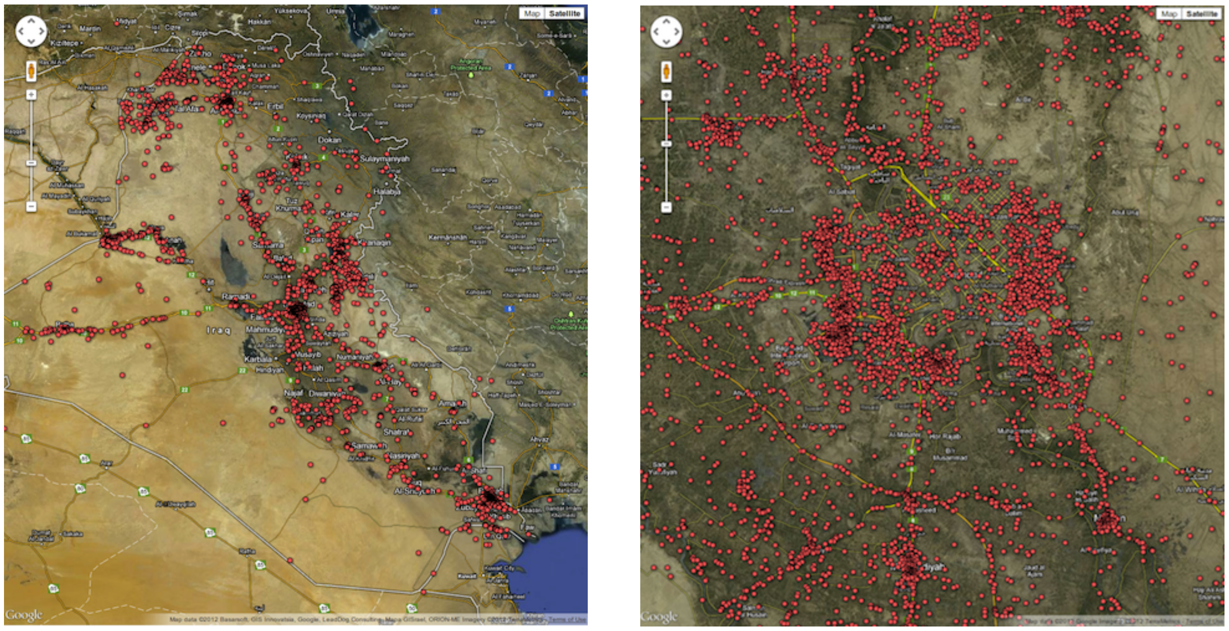


Figure 1: Map from Guardian Datablog article titled *Wikileaks Iraq war logs: every death mapped* [10]. The left shows the entire country, while the right is zoomed into the area around Baghdad. The map plots the more than 50,000 mortality incidents in Iraq recorded in the documents released by Wikileaks.

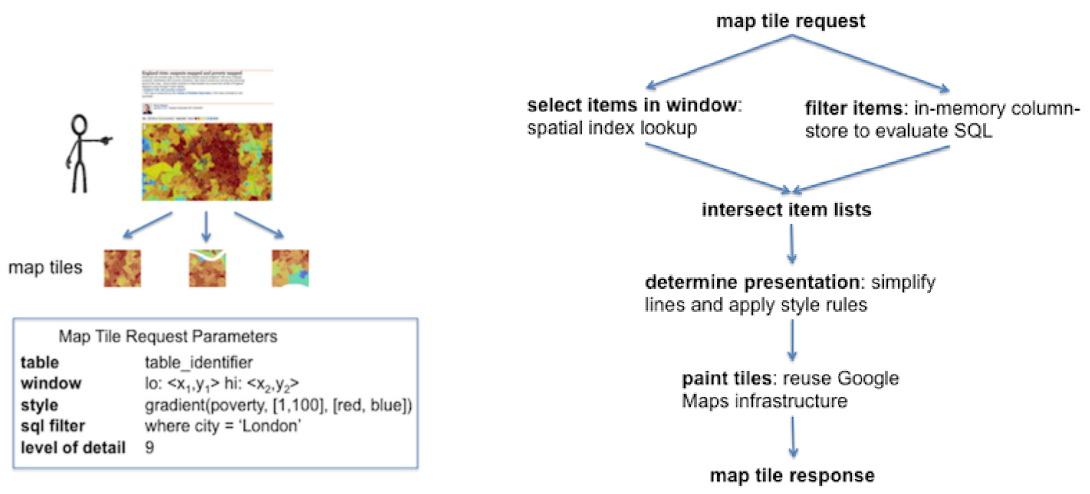
Through three examples (Figures 1, 3, and 5), we highlight the challenges addressed in order to scale our support to increasingly complex geospatial data. The net result is that we can now support interactive maps of data sets that have millions of features and can be embedded on high-traffic websites. Most importantly, we have relieved our users, the data experts, from the burden of thinking about systems and scalability, thereby letting them focus their efforts on their data quality and storytelling.

Consider our first example. In October 2010, Wikileaks released a collection of documents that have come to be known as the Iraq War Logs. Among other things, the documents list each incident between 2004 and 2009 that resulted in a civilian or military death. The documents were shared with multiple news organizations. As part of their analysis, the UK Guardian Datablog published an interactive map plotting each incident in the Iraq War Logs (Figure 1). The map visualization was backed by a Fusion Table created by the Guardian staff. Readers could zoom in and pan around the map or click on individual points to get specific details about the corresponding incident. The interactive nature of the map enabled their readers to better grasp the severity of the situation on the ground, be it the large numbers of murders in city of Baghdad or the impact of IEDs on the highways.

Embedding a Google Map with custom data was by no means new in 2010. However, the size of the underlying data and the magnitude of the user requests supported made the maps novel. The map in Figure 1 represents more than 50,000 incidents. When the story broke, more than 10 million user requests for this map were served during a 24 hour period, with a peak traffic of more than 1000 QPS.

Maps are only one of the novel features GFT offers. We support other interactive visualizations such as timelines and network graphs. We offer extensive support for the creation of virtual tables that can be created as views over other tables or by merging multiple tables. We also enable users to search for other public tables. A detailed discussion of these features is beyond the scope of this paper.

The rest of the paper is organized as follows. In Section 2, we begin with an overview of our basic map serving architecture. In Sections 3 and 4 we describe how we responded to increasing demands on our system, as users started storing larger data sets and items with more complex geometries. In Section 5, we highlight the


map tiles
map tile request
select items in window: spatial index lookup
filter items: in-memory column-store to evaluate SQL
intersect item lists
determine presentation: simplify lines and apply style rules
paint tiles: reuse Google Maps infrastructure
map tile response

Map Tile Request Parameters

table	table_identifier
window	lo: <x_1,y_1> hi: <x_2,y_2>
style	gradient(poverty, [1,100], [red, blue])
sql filter	where city = 'London'
level of detail	9

Figure 2: Overview of map processing in GFT. Viewing a map in a browser initiates multiple tile requests (a). Each tile request contains parameters used to compose the response, through the steps outlined in (b).

challenges in supporting merged tables, part of our longer-term goal to bring the worlds structured data together in a useful way. Section 6 concludes.

2 Architecture Overview

GFT lets users store tables of up to 100MB. The main goal of the GFT backend is to support SQL queries (selection, projection, grouping, aggregation, and equijoin) with very low latency and often very high QPS. In addition to the usual data types, we have geospatial data support that lets users store points, lines and polygons to render on a map. The polygons often have very high fidelity (e.g., imagine the polygon describing the boundaries of Canada). Users can set up customized presentations where the shape and color of the features on the map are determined dynamically based on the data.

When a user opens a map, the browser initiates multiple *tile requests* to Google backend servers. The user's viewport is composed of multiple *tiles*, each of which is a 256x256 PNG image. Users interact with the map by zooming in/out or panning around. Each of these actions can lead to more tile requests.

The parameters of a tile request are outlined in Figure 2a. In addition to *table* that identifies the dataset being mapped, the *window* identifies the boundary of the tile. The presentation of each item can be customized by a *style* specification. The style can either be a constant value, a value from a column, or a function (like bucket or gradient) applied to a value from a column. The request can also include an *sql filter*, a conjunction of SQL conditions over various columns of the table. The last parameter, *level of detail* (lod), indicates the zoom level of the tile. The lod specifies the granularity of the tile and ranges from 1 (entire surface of the Earth) to 20 (most zoomed in).

In Figure 2b, we outline the steps involved in computing a tile response. We first determine the items visible in the requested tile with a lookup in a spatial index. In parallel, we evaluate the filter conditions to identify matching items. For items in the intersection of the two lists, we determine the presentation of their geometries. In addition to applying the requested style function, for complex geometries, we try to *simplify* their representation by reducing the number of vertices without losing any fidelity. Finally, we delegate tile rendering to the common Google Maps rendering infrastructure. This rendering component expects a payload that lists the geometry and styling necessary to paint each tile. We discuss the optimizations implemented at each step in the rest of the paper.

3 Scaling to large datasets

Given our close integration with Google Maps, users soon realized the potential in creating interactive maps with large numbers of data-driven features. This led us to focus on some of the scale-up issues involved in serving these visualizations. To ensure the maps are interactive, our servers typically have to answer tile requests within a few hundred milliseconds. In practice, our goal is to answer requests under 10 milliseconds, except in the case of large datasets with very complex geometries. In this section we describe two aspects of our system implemented to achieve fast responses: (1) a highly optimized in-memory column store and (2) a representative and consistent sample of the underlying data per tile.

In-memory Column Store: To meet our millisecond response time goal, we implemented an in-memory query processing system. However, given our desire to support visualizations over a large number of tables (with limited resources), our processing is implemented over an LRU cache that loads tables only on demand. Tile requests typically only access a small number of columns in the underlying table; the only required column is the one defining the geometry to be mapped. Filters and styles are optional. If present in a request, filters and styles typically only require one additional column in the table. We implemented a cache that is managed in terms of column indices. Only columns that have been accessed by active map requests are maintained in memory. Using column-based indices also enables custom compaction algorithms to ensure that even million-row tables have a very compact memory footprint.

The most crucial column index is the spatial index that enables us to perform fast lookups of geometries that intersect any requested tile. This is implemented as a *one-dimensional* index based on the idea of Hilbert curve encoding [7]. An overview of our spatial query processing is presented in [5].

Consistent Spatial Sampling: To ensure quick rendering of individual tiles by the Google Maps infrastructure, the payload for each tile has to be limited to a few hundred items with at most a few thousand vertices. Polygons and lines can have many vertices. Large datasets, especially when zoomed out, can have numerous items per tile. In such cases, we respond with a representative sample of the underlying dataset.

Responding with too many items can sometimes be counter-productive and present a undesirable cognitive load on viewers. When looking at the entire map in Figure 1, all that a viewer can possibly perceive is a dense distribution of points. Hence, it is unnecessary to plot every single point. Instead, it is necessary that the chosen sample is sufficiently representative of the underlying distribution that it looks right to the viewer familiar with the data.

To ensure that tiles are globally consistent, sampling cannot be performed in isolation for each tile request. For example, as a viewer zooms in previously visible items must be retained as new items are displayed. Likewise, items can span multiple adjacent tiles, and hence when panning any such item must either be included in all the adjacent tiles, or in none. To achieve such consistent sampling, we associate a minimum level of detail l_i^m with each data item d_i. For a tile request with level of detail l, we only include items that have at least a minimum level of detail of l, i.e., only if $l \geq l_i^m$. To facilitate efficient query processing, the value l_i^m is stored in the spatial index and thus only items known to be visible at l are retrieved during the spatial lookup.

When sampling point-only datasets, in general, the points can be considered equally important. However, the same is not the case for line and polygons datasets. For example, when zoomed out on a map of islands of the world, excluding Greenland or Madagascar leads to a very misleading and unrepresentative map. We use a weighted sampling scheme where polygons are biased by their area and lines by their length.

In [13], we describe our solution for efficient spatial sampling. We show that spatial sampling in our context can be modeled as an integer logic program (ILP). We present optimizations that make the problem more tractable and also describe practical randomized solutions.

The in-memory column indices and spatial sampling ensure that for datasets with as many as a million features, we can support interactive maps with SQL filters and custom styling. For example, the tile requests for the map in Figure 1 can typically be answered in less than one millisecond. In Section 4, we present the results

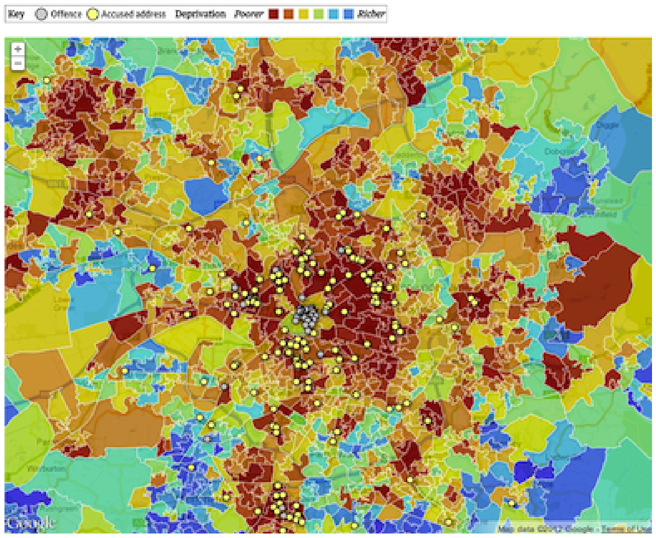


Figure 3: Map from the Guardian Datablog article titled *England riots: suspects mapped and poverty mapped* [11]. The map overlays two datasets. The colored jigsaw-like pattern is a poverty map that paints England wards based on poverty rate. Mapped on top is another dataset with the locations of rioting incidents (grey) and rioting suspects (yellow). The poverty map includes about 32,000 polygons with a total of 1.8 million vertices. The rioting dataset includes about 2100 incidents.

of a performance analysis for a more complex dataset.

4 Scaling to massive and complex polygon datasets

Given the ability to map large point datasets, it was natural that we soon received requests for similar support for maps with complex line and polygon geometries. To illustrate the next level of optimizations that were necessary, consider another post from the Guardian Datablog.

The map in Figure 3 was embedded in the article *England riots: suspects mapped and poverty mapped*, published after the 2011 riots in cities across England. The map shows the wards in England colored by poverty rate. Overlaid are incidents of rioting and the addresses of rioting suspects. The article observes correlations between poverty, disenchantment, and subsequent unrest. The poverty dataset included about 32,000 polygons and a total of about 1.8 million vertices. We now look at some of the optimizations we implemented to support such an intricate map.

Line and polygon simplification: As mentioned in the last section, to reduce rendering time, we limit the payload for each tile. However, sampling of items alone does not suffice for high quality line and polygon datasets that can have hundreds of vertices per individual polygon. For example, in Figure 3, there are 18 wards whose boundaries are described by more than 500 vertices and 960 wards whose boundaries are described by more than 200 vertices. Not all vertices are necessary when rendering the polygons, especially when zoomed out. Our first approach was to use well-understood line simplification algorithms (e.g., Douglas-Peucker [4]). While appropriate for lines, such approaches proved insufficient for polygons: simplifying each boundary independently may result in glaring holes between adjacent polygons on a space-filling map like Figure 3.

Our simplification algorithm instead is based on the idea of tile projection. Briefly, given a level of detail, we project lines onto a discrete 256 x 256 grid that represents the eventual tile. If adjacent points project onto the same grid location, they can be collapsed. In addition to being efficient (linear time versus the $O(n \log n)$ running time of the Douglas-Peucker algorithm), this simple approach ensures that (1) lines and polygons can be rendered onto tiles with no perceivable loss in fidelity, (2) there are guaranteed to be no holes between adjacent polygons, and (3) lines are simplified differently and appropriately at different levels of detail.

Effort-based response caching: Though linear in the number of vertices, line simplification can still be computationally expensive. This is especially the case when there are multiple large polygons in tiles at low levels of detail. In practice, we consider a request to be expensive if it takes more than 100ms to compute the payload for

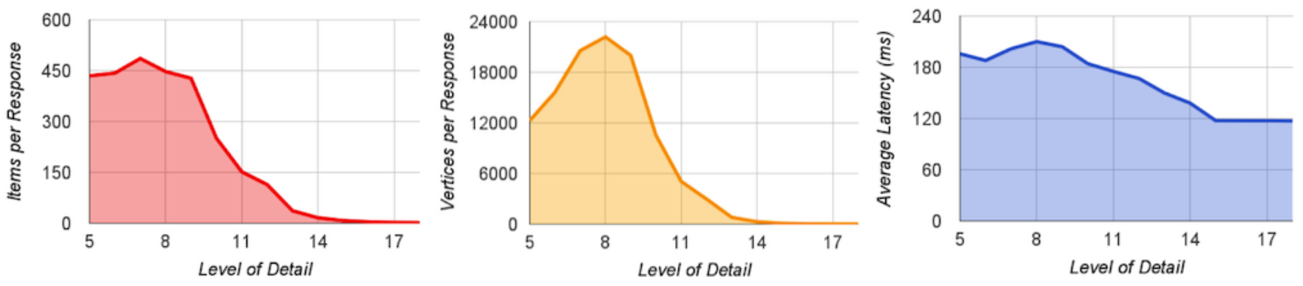


Figure 4: Distribution of average items per response, vertices per response, and GFT server latency with changes in the level of detail for the map in Figure 3. The latency was measured in an experimental setup. To isolate the performance of our own backend from the tile rendering delegated to the Google Maps infrastructure, we do not include the tile rendering time in the measured latencies. As the charts indicate, we never return more than 500 items per tile, never more than 22,000 vertices per tile, and are able to do so within 220 ms (100 ms plus approx 120 ms network latency).

the tile (retrieval + filtering + styling + simplification). At high request rates (more than 1000 per sec), this can significantly increase the CPU load and thereby lead to further degradation in response times. To facilitate scaling to high loads, we cache tile responses. If computing the payload for a tile response takes more than 100ms, we cache the response for a small amount of time. Thus, we only cache potentially expensively responses in heavy load situations.

Effort-based response caching is necessary to support a map like Figure 3 on the main page of a prominent newspaper. We generally avoid aggressive caching of responses on our servers to reduce memory use. We also limit public caching of tile responses due to limitations in the Google Maps API and our need to support near real-time updates on user tables.

Performance Analysis: We now look at the performance implications of using sampling, simplification and in-memory processing for the poverty map in Figure 3. The charts in Figure 4 show the distribution of response sizes and latencies with changing levels of detail. From the first chart, we see that sampling ensures that we never return more than 500 items per tile and that sampling comes into play when the level of detail is below 9. From the middle chart, we see that the average number of vertices per tile tops out at 21,000 at level of detail 8. Simplification plays a critical role below level of detail 8: the sample 450-500 polygons per tiles can be rendered, but with far fewer vertices. The last chart shows the response times (excluding tile rendering time and with caching disabled) as measured at a remote client. We estimate a network delay of about 120 ms. Thus, we have a maximum server processing time of about 100ms, and at higher levels of detail, it barely takes any time. Since the tile rendering payload is limited in size, the tiles are created efficiently. We are able to deliver fairly complex maps with low latencies, thereby ensuring interactivity.

Pre-computing Covers: Lastly, we consider geometries that are much more complex than those in the poverty map. Consider the maps in Figure 5 that were published by The Nature Conservancy (TNC). As an environmental organization, one of the TNCs goals is promoting awareness about critical environmental issues. Their website today includes a large number of maps and charts that display various properties of the different *eco-regions* of the world. The underlying eco-regions dataset (customized in two different ways in Figure 5) includes about 14,500 polygons with a total of 4.3 million vertices. These polygons are significantly more complex than those of English wards in Figure 3. In contrast with the mostly convex boundaries of the English wards, the eco-region boundaries follow natural features that define more convoluted shapes. Specifically, the TNC maps include 716 regions that have more than 1000 vertices describing their boundaries and 59 that have more than 10,000 vertices.

The TNC datasets are classic examples of high quality datasets that only get sporadic traffic. As mentioned

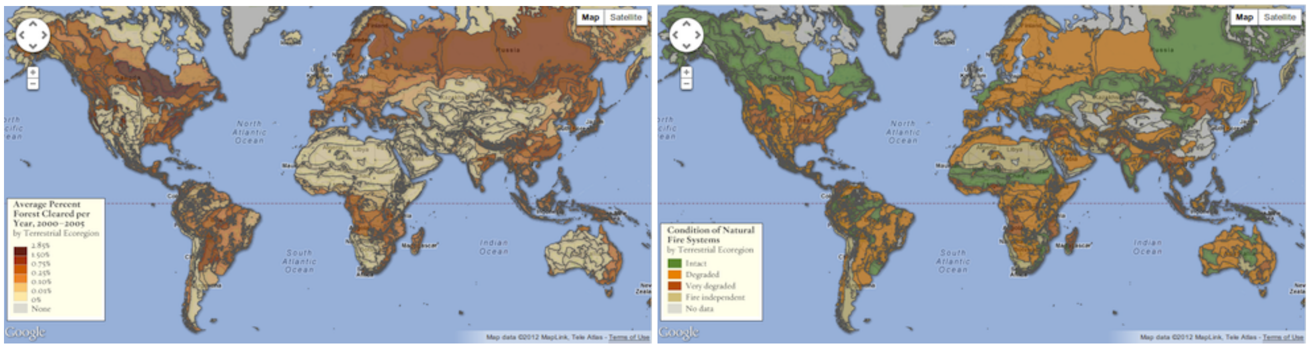


Figure 5: The Nature Conservancy maps showing the percent of forest cleared and conditions for natural fires in various eco-regions of the world [15]. The underlying eco-regions dataset has about 14,500 complex polygons with over 4.3 million vertices.

in Section 3, the column indices are loaded on demand. When the map is accessed for the very first time (or after a long time), the dataset has to be loaded into memory to create the compact in-memory indices. The first step in constructing the crucial spatial index is to compute the Hilbert *cover* for each of the polygons (details in [5]). Computing covers is an optimization problem and can be very expensive when geometries are large and irregular, as these are. As a result, it can take many seconds before the index can be computed and compacted. This leads to bad user experience as the map has to either be left empty or with a *in-progress* indicator. To address this challenge, we pre-compute the covers for large polygon datasets as a background process. When the map is accessed for the first time, the covers are directly loaded to construct the spatial index. This enables us to limit tile response times to less than a few hundred milliseconds in the vast majority of cases. Further reducing the delays when loading complex datasets is a topic on ongoing work.

5 Scaling by Merging

As stated at the outset, an important goal of GFT is to allow users to piece together disparate data sets to obtain new insights. The mechanism we currently offer to achieve that is to create *merged tables*. A merged table is essentially a view with an equi-join over any number of tables on a common key. Merged tables can be manipulated as any other table, including the ability to create embedded maps. Merged tables can lead to interesting modes of collaboration where other users can be granted privileges on partial views of the underlying data. Merged tables in GFT are live, i.e., not materialized. Thus updates made to the underlying tables are propagated to the merged table and queries over the merge are rewritten and executed over their base tables.

Consider the example of the TNC global conservation maps website [15]. There are a total of about 60 maps that paint the different eco-regions of the world according to various metrics (two examples are shown in Figure 5). Each of these maps has several thousand polygons and many million vertices. The resources required to serve these maps is significant, especially given that each map does not individually attract sufficient traffic to merit dedicated resources. However, all of these maps are created from only three distinct base tables that define the polygons for *terrestrial* [9], *marine* [14], and *freshwater* eco-regions [1]. The table backing each map is then created by merging one of the three base tables with another table that has columns specific to that particular map.

Merged tables offer many advantages. First, they offer the opportunity for significant reductions in resource use, while also ensuring better overall performance. Queries over merged tables are re-written as queries over the base tables and reuse the spatial index for the base table column. Given the complexity of the polygons, this leads to a large reduction in the memory footprint for storing indices. There is also a reduction in the index cache miss rate and hence the need to reload the datasets. The reduced footprint also frees up the in-memory column store to cache indices for other tables.

Second, merged tables are well suited in scenarios where the underlying data originates from different sources. The TNC, for example, works with a number of partner organizations, each of whom provides them with the data for a specific map. The partner organizations can retain ownership and edit privileges over the base tables they contribute, while the TNC is able to merge them with their eco-regions tables to create publishable maps.

Lastly, merged tables offer a better experience when mapping datasets that are likely to be updated. For example, GFT is used to host election maps where electoral districts are colored according to the live vote counts received by various candidates. See [12] and [8] for examples in the context of the recent Turkish national elections and the Iowa caucuses respectively. In such datasets, the geometries of electoral districts remain unchanged, while the vote tallies vary. To set up such a map, the electoral district geometries and vote tallies are stored in two separate tables that are then merged. The indices for the more complex geometry table never change, while only the much more compact vote tallies data has to be periodically reloaded. The details of managing updates is beyond the scope of this paper.

Merged tables have similarly been used in crisis response scenarios to map the availability or status of various resources in disaster zones. For example, GFT was used to track road closures and rolling brownouts in Japan after the 2011 tsunami [3] and road flooding in Vermont during the aftermath of Hurricane Irene [2].

6 Conclusion

Non-technical data users such as journalists, NGOs, social scientists and activists have Big Data challenges that are ignored by the typical focus on analytics and off-line computation. Although their individual tables may be small they can be combined with data from a vast pool of public structured data. Their computation requirements are driven by the need to serve high QPS, low-latency, interactive visualizations of large data sets that may be actively updated. Google Fusion Tables was developed with the needs of these users in mind. It provides them the tools and infrastructure to integrate multiple related data sets, interactively visualize this data and eventually embed the visualizations as part of their storytelling.

Based on real world examples, we discussed the specific challenges we faced to build an infrastructure to support interactive customizable maps over large complex geospatial data sets. In parallel we are also developing infrastructure to enhance the ability of our users to find and combine with other relevant public data sets, either within GFT or as HTML tables on the Web and visualize it in novel ways regardless of data size. More examples of prominent uses of GFT can be found in our gallery [6].

References

[1] R. Abell, M. L. Thieme, C. Revenga, M. Bryer, M. Kottelat, N. Bogutskaya, B. Coad, N. Mandrak, S. C. Balderas, W. Bussing, M. L. J. Stiassny, P. Skelton, G. R. Allen, P. Unmack, A. Naseka, R. Ng, N. Sindorf, J. Robertson, E. Armijo, J. V. Higgins, T. J. Heibel, E. Wikramanayake, D. Olson, H. L. López, R. E. Reis, J. G. Lundberg, M. H. S. Pérez, and P. Petry. Freshwater Ecoregions of the World: A New Map of Biogeographic Units for Freshwater Biodiversity Conservation. *BioScience*, 58(5), May 2008.

[2] Vermont Flooding 2011. http://crisislanding.appspot.com/?crisis=2011_flooding_vermont, August 2011.

[3] http://www.google.com/intl/ja/crisisresponse/japanquake2011_traffic.html, March 2011.

[4] D. Douglas and T. Peucker. Algorithms for the reductions of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2), December 1973.

[5] H. Gonzalez, A. Halevy, C. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen. Google Fusion Tables: Data Management, Integration, and Collaboration in the Cloud. In *SOCC*, 2010.

[6] Google Fusion Tables: Example Gallery. https://sites.google.com/site/fusiontablestalks/stories.

[7] D. Hilbert. Ueber die stetige Abbildung einer Linie auf ein Flchenstck. *Mathematische Annalen*, 38, 1891.

[8] J. Keefe. Making AP election data easy with Fusion Tables. http://johnkeefe.net/making-ap-election-data-easy-with-fusion-tabl, January 2012.

[9] D. M. Olson, E. Dinerstein, E. D. Wikramanayake, N. D. Burgess, G. V. N. Powell, E. C. Underwood, J. A. D'Amico, I. Itoua, H. E. Strand, J. C. Morrison, C. J. Loucks, T. F. Allnutt, T. H. Ricketts, Y. Kura, J. F. Lamoreux, W. W. Wettengel, P. Hedao, and K. R. Kassem. Terrestrial Ecoregions of the World: A New Map of Life on Earth. *BioScience*, 51(11), November 2001.

[10] S. Rogers. Wikileaks Iraq war logs: every death mapped. http://www.guardian.co.uk/world/datablog/interactive/2010/oct/23/wikileaks-iraq-deaths-map, October 2010.

[11] S. Rogers. England riots: suspects mapped and poverty mapped. http://www.guardian.co.uk/news/datablog/interactive/2011/aug/16/riots-poverty-map, August 2011.

[12] C. Sabnis. Visualizing Election Results Directly from the Ballot Box. http://googlepublicsector.blogspot.com/2011/06/visualizing-election-results-directly.html, June 2011.

[13] A. D. Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy. Efficient Spatial Sampling of Large Geographic Tables. In *SIGMOD*, 2012.

[14] M. D. Spalding, H. E. Fox, G. R. Allen, N. Davidson, Z. A. Ferdaña, M. Finlayson, B. S. Halpern, M. A. Jorge, A. Lombana, S. A. Lourie, K. D. Martin, E. Mcmanus, J. Molnar, C. A. Recchia, and J. Robertson. Marine Ecoregions of the World: A Bioregionalization of Coastal and Shelf Areas. *BioScience*, 57(7), July 2007.

[15] The Nature Conservancy: Global Conservation Maps. http://maps.tnc.org/globalmaps.html.