Chapter 8: Interactive Analytics

Introduced by Joe Hellerstein

Selected Readings:


For decades, most database workloads have been partitioned into two categories: (1) many small “transaction processing” queries that do lookups and updates on a small number of items in a large database, and (2) fewer big “analytic” queries that summarize large volumes of data for analysis. This section is concerned with ideas for accelerating the second category of queries particularly to answer them at interactive speeds, and allow for summarization, exploration, and visualization of data.

Over the years there has been a great deal of buzzword bingo in industry to capture some or all of this latter workload, from “Decision Support Systems” (DSS) to “Online Analytic Processing” (OLAP) to “Business Intelligence” (BI) to “Dashboards” and more generally just “Analytics”. Billions of dollars of revenue have been associated with these labels over time, so marketers and industry analysts worked hard over the years to define, distinguish and sometimes try to subvert them. By now it’s a bit of a mess of nomenclature. The interested reader can examine Wikipedia and assess conventional wisdom on what these buzzwords came to mean and how they might be different; be warned that it will not be a scientifically satisfying exercise.

Here, I will try to keep things simple and somewhat technically grounded.

Human cognition cannot process large amounts of raw data. In order for a human to make sense of data, the data has to be “distilled” in some way to a relatively small set of records or visual marks. Typically this is done by partitioning the data and running simple arithmetic aggregation functions on the partitions think of SQL’s “GROUP BY” functionality as a canonical pattern. Subsequently the data typically needs to be visualized for users to relate it to their task at hand.

The primary challenge we address in this chapter is to make large-scale grouping/aggregation queries run at interactive speeds—even in cases where it is not feasible to iterate through all the data associated with the query.

How do we make a query run in less time than it takes to look at the data? There is really only one answer: we answer the query without looking at (all) the data. Two variants of this idea emerge:

Precomputation: If we know something about the query workload in advance, we can distill the data in various ways to allow us to support quick answers (either accurate or approximate) to certain queries. The simplest version of this idea is to precompute the answers to a set of queries, and only support those queries. We discuss more sophisticated answers below. Sampling: If we cannot anticipate the queries well in advance, our only choice is to look at a subset of the data at query time. This amounts to sampling from the data, and approximating the true answer based on the sample.

The papers in this section focus on one or both of these approaches.

Our first two papers address what the database community dubbed “data cubes” [DataCubes]. Data cubes were originally supported by standalone query/visualization tools called On Line Analytic Processing (OLAP) systems. The name is due to relational pioneer Ted Codd, who was hired as a consultant to promote an early OLAP vendor called Essbase (subsequently bought by Oracle). This was not one of Codd’s more scholarly endeavors.

Early OLAP tools used a pure “precomputation” approach. They ingested a table, and computed and stored the answers to a family of GROUP BY queries over that table:

\[\text{GROUP BY} \]
each query grouped on a different subset of the columns, and computed summary aggregates over the non-grouped numerical columns. For example, in a table of car sales, it might show total sales by Make, total sales by Model, total sales by Region, and total sales by combinations of any 2 or 3 of those attributes. A graphical user interface allowed users to navigate the resulting space of group-by queries interactively, this space of queries is what became known as a data cube\(^2\). Originally, OLAP systems were pitched as standalone “Multidimensional Databases” that were fundamentally different than relational databases. However, Jim Gray and a consortium of authors in the relational industry explained how the notion of a data cube can fit in the relational context \([4]\), and the concept subsequently migrated into the SQL standard as a single query construct: “CUBE BY”. There is also a standard alternative to SQL called MDX that is less verbose for OLAP purposes. Some of the terminology from data cubes has become common parlance—in particular, the idea of “drilling down” into details and “rolling up” to coarser summaries.

A naïve relational precomputation approach for precomputing a full data cube does not scale well. For a table with \(k\) potential grouping columns, such an approach would have to run and store the results for \(2^k\) GROUP BY queries, one for each subset of the columns. Each query would require a full pass of the table.

Our first paper by Harinarayan, Rajaraman and Ullman reduces this space: it chooses a judicious subset of queries in the cube that are worthy of precomputation; it then uses the results of those queries to compute the results to any other query in the cube. This paper is one of the most-cited papers in the area, in part because it was early in observing that the structure of the data cube problem is a set-containment lattice. This lattice structure underlies their solution, and recurs in many other papers on data cubes (including our next paper), as well as on certain data mining algorithms like Association Rules (a.k.a. Frequent Itemsets) \([2]\). Everyone working in the OLAP space should have read this paper.

Our second paper by Zhao, Deshpande and Naughton focuses on the actual computation of results in the cube. The paper uses an “array-based” approach: that is, it assumes the data is stored in an Essbase-like sparse array structure, rather than a relational table structure, and presents a very fast algorithm that exploits that structure. However, it makes the surprising observation that even for relational tables, it is worthwhile to convert tables to arrays in order to run this algorithm, rather than to run a (far less efficient) traditional relational algorithm. This substantially widens the design space for query engines. The implication is that you can decou-

---

\(^2\)Note that this idea did not originate in databases. In statistics, and later in spreadsheets, there is an old, well-known idea of a contingency table or cross tabulation (crosstab).
structure-on-use.

We include the first paper on online aggregation here. To appreciate the paper, it’s important to remember that databases at the time had long operated under a mythology of “correctness” that is a bit hard to appreciate in today’s research environment. Up until approximately the 21st century, computers were viewed by the general populace—and the business community—as engines of accurate, deterministic calculation. Phrases like “Garbage In, Garbage Out” were invented to remind users to put “correct” data into the computer, so it could do its job and produce “correct” outputs. In general, computers weren’t expected to produce “sloppy” approximate results.

So the first battle being fought in this paper is the idea that the complete accuracy in large-scale analytics queries is unimportant, and that users should be able to balance accuracy and running time in a flexible way. This line of thinking quickly leads to three research directions that need to work in harmony: fast query processing, statistical approximation, and user interface design. The inter-dependencies of these three themes make for an interesting design space that researchers and products are still exploring today.

The initial paper we include here explores how to embed this functionality in a traditional DBMS. It also provides statistical estimators for standard SQL aggregates over samples, and shows how stratified sampling can be achieved using standard B-trees via “index striding”, to enable different groups in a GROUP BY query to be sampled at different rates. Subsequent papers in the area have explored integrating online aggregation with many of the other standard issues in query processing, many of which are surprisingly tricky: joins, parallelism, subqueries, and more recently the specifics of recent Big Data systems like MapReduce and Spark.

Both IBM and Informix pursued commercial efforts for online aggregation in the late 1990s, and Microsoft also had a research agenda in approximate query processing as well. None of these efforts came to market. One reason for this at the time was the hidebound idea that “database customers won’t tolerate wrong answers” 3. A more compelling reason related to the coupling of user interface with query engine and approximation. At that time, many of the BI vendors were independent of the database vendors. As a result, the database vendors didn’t “own” the end-user experience in general, and could not deliver the online aggregation functionality directly to users via standard APIs. For example, traditional query cursor APIs do not allow for multiple approximations of the same query, nor do they support confidence intervals associated with aggregate columns. The way the market was structured at the time did not support aggressive new technology spanning both the back-end and front-end.

Many of these factors have changed today, and online aggregation is getting a fresh look in research and in industry. The first motivation, not surprisingly, is the interest in Big Data. Big Data is not only large in volume, but also has wide “variety” of formats and uses which means that it may not be parsed and understood until users want to analyze. For exploratory analytics on Big Data, the combination of large volumes and schema-on-use makes precomputation unattractive or impossible. But sampling on-the-fly remains cheap and useful.

In addition, the structure of the industry and its interfaces has changed since the 1990s. From the bottom up, query engine standards today often emerge and evolve through open source development, and the winning projects (e.g., Hadoop and Spark) become close enough to monopolies that their APIs can dictate client design. At the same time from the top down, hosted data visualization products in the cloud are often vertically integrated: the front-end experience is the primary concern, and is driven by a (often special-purpose) back-end implementation without concern for standardization. In both cases, it’s possible to deliver a unique feature like online aggregation through the stack from engine to applications.

In that context we present one of the more widely-read recent papers in the area, on BlinkDB. The system makes use of what Olken calls “materialized sample views”; precomputed samples over base tables, stored to speed up approximate query answering. Like the early OLAP papers, BlinkDB makes the case that only a few GROUP BY clauses need to be precomputed to get good performance on (stable) workloads. Similar arguments are made by the authors of the early AQUA project on approximate queries [1], but they focused on precomputed synopses (“sketches”) rather than materialized sample views as their underlying approximation mechanism. The BlinkDB paper also makes the case for stratification in its views to capture small groups, reminiscent of the Index Striding in the online aggregation paper. BlinkDB has received interest in industry, and the Spark team has recently proposed augmenting its precomputed samples with sampling on the fly—a sensible mixture of techniques to achieve online aggregation as efficiently as possible. Recent commercial BI tools like ZoomData seem to use online aggregation as well (they call it “query sharpening”).

With all this discussion of online aggregation, it’s worth taking a snapshot of current market realities. In the 25 years since it was widely introduced, OLAP-style precomputation has underpinned what is now a multi-billion dollar BI in-

---

3This was particularly ironic given that the sampling support provided by some of the vendors was biased (by sampling blocks instead of tuples).
dustry. By contrast, approximation at the user interface is practically non-existent. So for those of you keeping score at home based on revenue generation, the simple solution of precomputation is the current winner by a mile. It’s still an open question when and if approximation will become a bread-and-butter technique in practice. At a technical level, the fundamental benefits of sampling seem inevitably useful, and the technology trends around data growth and exploratory analytics make it compelling in the Big Data market. But today this is still a technology that is before its time.

A final algorithmic note: approximate queries via sketches are in fact very widely used by engineers and data scientists in the field today as building blocks for analysis. Outside of the systems work covered here, well-read database students should be familiar with techniques like CountMin sketches, HyperLogLog sketches, Bloom filters, and so on. A comprehensive survey of the field can be found in [3]; implementations of various sketches can be found in a number of languages online, including as user-defined functions in the MADlib library mentioned in Chapter 11.

References


