Maintenance of top-k materialized views

Eftychia Baikousi · Panos Vassiliadis

Published online: 15 October 2009
© Springer Science+Business Media, LLC 2009

Abstract In this paper we present results on the problem of maintaining materialized top-k views and provide results in two directions. The first problem we tackle concerns the maintenance of top-k views in the presence of high deletion rates. We provide a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams. The second problem we have been concerned with has to do with the efficient maintenance of multiple top-k views in the presence of updates to their base relation. To this end, we provide theoretical guarantees for the nucleation (practically, inclusion) of a view with respect to another view and the reflection of this property to the management of updates. We also provide algorithmic results towards the maintenance of a large number of views, via their appropriate structuring in hierarchies of views.

Keywords Top-k views · View refreshment

1 Introduction

The top-k querying problem concerns the retrieval of the top-k results of a ranked query over a database. Specifically, given a relation $R(\text{tid}, A_1, A_2, \ldots, A_m)$ and a query $Q$ over $R$ retrieves the top-k tuples from $R$ having the $k$ highest values according to a scoring function $f$ that accompanies $Q$. Typically, $f$ is a monotone ranking function of the form: $f : \text{dom}(A_1) \times \cdots \times \text{dom}(A_m) \rightarrow \mathbb{R}$.
Related work has extensively dealt with the problem of efficiently computing the top-$k$ results of a query. The first algorithms that occurred in this context are FA [4, 5] and TA [6], with various extensions that followed them for specific contexts (e.g., parallel or distributed computation, etc.). In recent years, in an attempt to achieve improved performance, researchers solve the problem of answering top-$k$ queries via materialized views [2, 9, 10]. In this setting, results of previous top-$k$ queries are stored in the form of materialized views. Then, a new top-$k$ query may be answered through materialized views resulting in better performance than making use only of the base relation from the database. As typically happens with materialized views, though, when the source relation is updated, we need to refresh the contents of all the materialized views in order to reflect the most recent data.

The two main problems that pertain to the maintenance of materialized views are (a) the correct and efficient maintenance of a single view when updates occur to the base relation, and (b) the generalization of the maintenance problem for a large number of materialized views.

Maintaining a single top-$k$ materialized view

Concerning the problem of maintaining a single view, the first—and only—attempt that we are aware of is [19]. To sustain the update rate at the source relation without having to fully re-compute the materialized views, [19] maintain $k_{\text{max}}$ tuples (instead of the necessary $k$) and perform refill queries whenever the contents of the materialized views fall below the threshold of $k$ tuples. Yet, the approach of [19] suffers from the following problems: (a) the method is theoretically guaranteed to work well only when insertions and deletions are of the same probability (in fact, the authors deal with updates in their experiments), (b) there is no quality-of-service guarantee when deletions are more probable than insertions.

In this paper, we compensate for these shortcomings by providing a method that is able to provide quality guarantees when the deletion rate is higher than the insertion rate. The case is not so rare if one considers that the number of persons logged in a web server or a portal presents anticipated high peaks and valleys at specific time points or dates. The first contribution of our work in this paper is to deal with these phenomena efficiently. The solution to the problem is not obvious for the following reasons. First, even if the value distributions of the attributes that participate in the computation of the score are known individually, it is not possible to compute the distribution of their linear combination (i.e., the score) unless they are stable probabilities—e.g., Normal, Cauchy. Second, even if we extend $k$ with extra tuples to sustain the incoming stream of updates that eventually affects the top-$k$ materialized view, the extra tuples increase the possibility that an incoming source update might affect the view, thus resulting in the need to recursively compute this extension. Finally, we need to accommodate statistical fluctuations from the expected values. To resolve all the above, we provide a principled method that operates independently of the statistical properties of the data and the characteristics of the update streams. The method comprises the following steps: (a) a computation of the rate that actually affects the materialized view, (b) a computation of the necessary extension to $k$ in order to handle the augmented number of deletions that occur and (c) a fine tuning part that adjusts this value to take the fluctuation of the statistical properties of this value into consideration.