Developing a Dynamic Materialized View Index for Efficiently Discovering Usable Views for Progressive Queries

Chao Zhu*, Qiang Zhu*, Calisto Zuzarte**, and Wenbin Ma**

Abstract—Numerous data intensive applications demand the efficient processing of a new type of query, which is called a progressive query (PQ). A PQ consists of a set of unpredictable but inter-related step-queries (SQ) that are specified by its user in a sequence of steps. A conventional DBMS was not designed to efficiently process such PQs. In our earlier work, we introduced a materialized view based approach for efficiently processing PQs, where the focus was on selecting promising views for materialization. The problem of how to efficiently find usable views from the materialized set in order to answer the SQs for a PQ remains open. In this paper, we present a new index technique, called the Dynamic Materialized View Index (DMVI), to rapidly discover usable views for answering a given SQ. The structure of the proposed index is a special ordered tree where the SQ domain tables are used as search keys and some bitmaps are kept at the leaf nodes for refined filtering. A two-level priority rule is adopted to order domain tables in the tree, which facilitates the efficient maintenance of the tree by taking into account the dynamic characteristics of various types of materialized views for PQs. The bitmap encoding methods and the strategies/algorithms to construct, search, and maintain the DMVI are suggested. The extensive experimental results demonstrate that our index technique is quite promising in improving the performance of the materialized view based query processing approach for PQs.

Keywords—Database, query processing, query optimization, progressive query, materialized view, index

1. INTRODUCTION

The rapid growth of numerous data intensive applications (e.g., astronomy, biology, and social media) has led to significant research being focused on the problem of analyzing a large amount of data in databases. In such data intensive applications, a new type of query, which is called a progressive query (PQ), is required [49]. Unlike a conventional query, a progressive query is defined as a query that is formulated in more than one step (progressively), where each step is called a step-query (SQ) [49]. A user submits his/her first SQ $sq_1$ on one or more external (existing) tables in the database. Based on the result $R_1$ of $sq_1$, the user submits a second SQ $sq_2$.

* The preliminary results of this work were presented at the 2012 Conference of the IBM Centre for Advanced Studies on Collaborative Research -- CASCON’12 [48] in Toronto between Nov, 5–7, 2012. Research was partially supported by the IBM Canada Software Lab and The University of Michigan.

Manuscript received February 26, 2013; first revision July 26, 2013; accepted September 3, 2013.

Corresponding Author: Chao Zhu (zhuchaon1@gmail.com)  
* Dept. of Computer and Information Science, the University of Michigan, Dearborn, MI 48128, USA (zhuchaon1@gmail.com, qzhu@umich.edu)  
** IBM Canada Software Lab, Markham, Ontario, Canada (calisto@ca.ibm.com, wenbinm@ca.ibm.com)
using $R_1$ and/or additional external tables as its input. In general, an SQ may use the result(s) of its previous SQ(s) and/or external tables as its input. After the last SQ is submitted, a PQ is completed. Hence, the user gradually approaches his/her desired result by issuing a number of related SQs to the database. The relationships among the SQs and referenced external tables for a PQ can be depicted as a dependency graph (DG) [49]. Based on the structures of their DGs, PQs can be classified as single-input linear, multiple-input linear, and non-linear ones.

As an illustration, let us consider the following example: assume that a user wants to buy a car. In the first step, he/she searches all of the cars that are available on a website (i.e., SQ 1). However, there are too many results that show up. Hence, in the second step, he/she adds a search condition to only show cars that were manufactured in the USA (i.e., SQ 2). After analyzing the result, he/she selects a “Ford” car. Therefore, in the third step, he/she checks the details (e.g., prize, configuration, etc.) of all the “Ford” cars to make a final decision (i.e., SQ 3).

From the above example, we can see that the main characteristic of a PQ is that the SQs of a PQ cannot be known beforehand. Each SQ is formulated based on the result(s) of the previous SQ(s). Hence, to execute such unpredictable SQs, the results of the previous SQs of each in-process PQ have to be kept in the system (as one type of [temporary] materialized view) until the PQ is completed. On the other hand, it is desirable to retain some popular results (i.e., those that are frequently utilized) for SQs in the system (as another type of [critical] materialized view) even after their corresponding PQs are completed so that the SQs of future PQs can utilize these results to improve their processing efficiency. To achieve this goal, we introduced a dynamic materialized view based approach for processing PQs in [46]. A benefit estimation model was developed to determine if the result table for the SQ of a completed PQ should still be kept in the system as a (critical) materialized view. Both the popular results of SQs from completed PQs and the results of previous SQs from in-process PQs are kept as materialized views in a view storage/set (VS).

In [46], we mainly focused on discussing how to select promising materialized views for processing PQs (i.e., studying the view materialization selection problem). However, as more and more materialized views are selected and saved in the VS, how to efficiently discover and use the relevant materialized views from the VS for answering the SQs of a PQ becomes an important issue. Since view matching (searching) is a time consuming task, if the saved materialized views are not properly managed (e.g., each view in the VS is examined sequentially to match the currently executing SQ) the view matching/searching cost is very expensive and the performance of the PQ seriously suffers. Therefore, a new PQ oriented materialized view managing technique is required. Our target is to develop a view access method to efficiently discover usable views for answering SQs.

In this paper, we present a new index technique, which is called the dynamic materialized view index (DMVI), to index the materialized views in the VS and we used this index to efficiently discover/search for usable materialized views for answering a PQ (i.e., solving the view searching problem). When a new view $v$ is added to the VS, a search path for $v$ is created in the DMVI and the relevant information (address, query expression, bitmaps, etc.) of $v$ is stored at the end (leaf) of the path. Many views may share the same search path. When an SQ $sq$ arrives, the system goes through the proper search path to discover usable views for answering $sq$. In addition, a bitmap based method is applied at the leaf of the path in the DMVI for further pruning unusable views. By using the DMVI, the views in the VS can be efficiently managed and searched for answering the SQs of the PQs. Algorithms and strategies for constructing the DMVI,