Aggregate Query Rewriting in Multidimensional Databases

Leonardo Tininini
CNR - Istituto di Analisi dei Sistemi e Informatica "Antonio Ruberti," Italy

INTRODUCTION

An efficient query engine is certainly one of the most important components in data warehouses (also known as OLAP systems or multidimensional databases) and its efficiency is influenced by many other aspects, both logical (data model, policy of view materialization, etc.) and physical (multidimensional or relational storage, indexes, etc). As is evident, OLAP queries are often based on the usual metaphor of the data cube and the concepts of facts, measures and dimensions and, in contrast to conventional transactional environments, they require the classification and aggregation of enormous quantities of data. In spite of that, one of the fundamental requirements for these systems is the ability to perform multidimensional analyses in online response times. Since the evaluation from scratch of a typical OLAP aggregate query may require several hours of computation, this can only be achieved by pre-computing several queries, storing the answers permanently in the database and then reusing them in the query evaluation process. These precomputed queries are commonly referred to as materialized views and the problem of evaluating a query by using (possibly only) these precomputed results is known as the problem of *answering/rewriting queries using views*. In this paper we briefly analyze the difference between query answering and query rewriting approach and why query rewriting is preferable in a data warehouse context. We also discuss the main techniques proposed in literature to rewrite aggregate multidimensional queries using materialized views.

BACKGROUND

Multidimensional data are obtained by applying aggregations and statistical functions to elementary data, or more precisely to data groups, each containing a subset of the data and homogeneous with respect to a given set of attributes. For example, the data "Average duration of calls in 2003 by region and call plan" is obtained from the so-called fact table, which is usually the product of complex source integration activities (Lenzerini, 2002) on the raw data corresponding to each phone call in that year. Several groups are defined; each consisting of calls made in the same region and with the same call plan, and finally applying the average aggregation function on the duration attribute of the data in each group. The pair of values (region, call plan) is used to identify each group and is associated with the corresponding average duration value. In multidimensional databases, the attributes used to group data define the dimensions, whereas the aggregate values define the measures.

The term multidimensional data comes from the well-known metaphor of the data cube (Gray, Bosworth, Layman, & Pirahesh, 1996). For each of n attributes, used to identify a single measure, a dimension of an n-dimensional space is considered. The possible values of the identifying attributes are mapped to points on the dimension's axis, and each point of this n-dimensional space is thus mapped to a single combination of the identifying attribute values and hence to a single aggregate value. The collection of all these points, along with all possible projections in lower dimensional spaces, constitutes the so-called data cube. In most cases, dimensions are structured in hierarchies, representing several granularity levels of the corresponding measures (Jagadish, Lakshmanan, & Srivastava, 1999). Hence a time dimension can be organized into days, months and years; a territorial dimension into towns, regions and countries; a product dimension into brands, families and types. When querying multidimensional data, the user specifies the measures of interest and the level of detail required by indicating the desired hierarchy level for each dimension. In a multidimensional environment querying is often an exploratory process, where the user "moves" along the dimension hierarchies by increasing or reducing the granularity of displayed data. The drill-down operation corresponds to an increase in detail, for example, by requesting the number of calls by region and month, starting from data on the number of calls by region or by region and year. Conversely, roll-up allows the user to view data at a coarser level of granularity (Agrawal, Gupta, & Sarawagi, 1997; Cabibbo & Torlone, 1997).

Multidimensional querying systems are commonly known as OLAP (Online Analytical Processing) Systems, in contrast to conventional OLTP (Online Transactional Processing) Systems. The two types have several con-

trasting features, although they share the same requirement of fast "online" response times. In particular, one of the key differences between OLTP and OLAP queries is the number of records required to calculate the answer. OLTP queries typically involve a rather limited number of records, accessed through primary key or other specific indexes, which need to be processed for short, isolated transactions or to be issued on a user interface. In contrast, multidimensional queries usually require the classification and aggregation of a huge amount of data (Gupta, Harinarayan, & Quass, 1995) and fast response times are made possible by the extensive use of pre-computed queries, called materialized views (whose answers are stored permanently in the database), and by sophisticated techniques enabling the query engine to exploit these pre-computed results.

MAIN THRUST

The problem of evaluating the answer to a query by using pre-computed (materialized) views has been extensively studied in literature and generically denoted as answering queries using views (Levy, Mendelzon, Sagiv, & Srivastava, 1995; Halevy, 2001). The problem can be informally stated as follows: given a query Q and a collection of views V over the same schema s, is it possible to evaluate the answer to Q by using (only) the information provided by V? A more rigorous distinction has also been made between view-based query rewriting and query answering, corresponding to two distinct approaches to the general problem (Calvanese, De Giacomo, Lenzerini, & Vardi, 2000; Halevy, 2001). This is strictly related to the distinction between view definition and view extension, which is analogous to the standard distinction between schema and instance in database literature. Broadly speaking, view definition corresponds to the way the query is syntactically defined, for example to the corresponding SQL expression, while its extension corresponds to the set of returned tuples, that is, the result obtained by evaluating the view on a specific database instance.

Query Answering vs. Query Rewriting

Query rewriting is based on the use of view definitions to produce a new rewritten query, expressed in terms of available view names and equivalent to the original. The answer can then be obtained by using the rewritten query and the view extensions (instances). Query answering, in contrast, is based on the exploitation of both view definitions and extensions and attempts to determine the best possible answer, possibly a subset of the exact answer, which can be extracted from the view extensions (Abiteboul & Duschka, 1998; Grahne & Mendelzon, 1999).

In general, query answering techniques are preferable in contexts where exact answers are unlikely to be obtained (e.g., integration of heterogeneous data sources, like Web sites), and response time requirements are not very stringent. However, as noted in Grahne & Mendelzon (1999), query answering methods can be extremely inefficient, as it is difficult or even impossible to process only the "useful" views and apply optimization techniques such as pushing selections and joins. As a consequence, the rewriting approach is more appropriate in contexts such as OLAP systems, where there is a very large amount of data and fast response times are required (Goldstein & Larson, 2001), and for query optimization, where different query plans need to be maintained in the main memory and efficiently compared (Afrati, Li, & Ullman, 2001).

Rewriting and Answering: An Example

Consider a fact table Cens, of elementary census data on the simplified schema: (Census_tract_ID, Sex, Empl_status, Educ_status, Marital_status) and a collection of aggregate data representing the resident population by sex and marital status, stored in a materialized view on the schema V: (Sex, Marital_status, Pop_res). For simplicity, it is assumed that the dimensional tables are "collapsed" in the fact table Cens. A typical multidimensional query will be shown in the next section. The view V is computed by a simple count(*)-group-by query on the table Cens.

```
CREATE VIEW V AS
SELECT Sex, Marital_status, COUNT(*) AS Pop_res
FROM Cens
GROUP BY Sex, Marital_status
```

The query Q expressed by

```
SELECT Marital_status, COUNT(*)
FROM Cens
GROUP BY Marital_status
```

corresponding to the resident population by marital status can be computed without accessing the data in Cens, and be rewritten as follows:

```
SELECT Marital_status, SUM(Pop_res)
FROM V
GROUP BY Marital_status
```

Note that the rewritten query can be obtained very efficiently by simple syntactic manipulations on Q and V and its applicability does not depend on the records in V. Suppose now some subsets of (views on) Cens are available, corresponding to the employment statuses stu-