CS505 Intermediate Topics in Databases, Fall 2012, Project 2 page

Background

Among many architectures for databases, one that we will implement in this project is *columnar distribution* (also known as vertical partitioning). The idea of columnar distribution is based on the familiar property of tables: When X is a set of attributes of table R, and A is a set of attributes and X --> A holds in R, then XA, X(R-A) is a loss-less decomposition. That is the join of the projection of R to XA and to X(R\A) is loss-less.

So here is what we can do. We add a new key attribute (in effect a serial number of a record). Then we can losslessly decompose our table R into two-attribute tables. The scheme of each of these tables is (*k*,*a*) where *k* is the serial number attribute, and *a* is an attribute. Here is an example. Say the scheme of my relation *Cust* is (lname, fname, add). Then I add the serial number attribute k, and get a loss-less decomposition of *Cust* into 3 tables, namely projections to (k,lname), (k,fname) and (k,add).

It may look like we actually require more space to store the original table (after all the key attribute is repeated), but we get various benefits. First, maybe we will be able to compress the resulting tables if there are many repeats of attribute values. But a much bigger benefit comes from the query processing. Say we have 50 attributes in a table (nothing unusual in a large database, especially if we map a non-1NF relation on a relational table). Now, assume we run a query which involves in the answer (i.e. in the SELECT list and in the WHERE condition) only (again, say) 5 attributes. Then, in processing, all we need is 5 of these 2-attribute tables, namely those which involve the attributes mentioned in the query!

The user inputs the table, but we keep in the database projections only. When a query comes in, we *transform* that query Q into another query Q' which pertains *only* to attributes that are mentioned in this query. There is some parsing and rewriting of the query. But altogether we expend much less effort (and space).

Let us look at an example. We have the scheme as above. Columnar decomposition of out table Cust is Cust_lname, Cust_fname, Cust_address. The name of added key attribute is k. Let us look at the following query to the table Cust:

```
SELECT lname fname
FROM Cust
WHERE (lname LIKE m*) AND NOT(fname LIKE v*)
```

In reality this query will be transformed (in the columnar context) into the following query:

```
SELECT t1.lname t2.fname
FROM Cust_lname AS t1 Cust_fname AS t2
WHERE (t1.lname LIKE m*) AND (NOT (t2.fname LIKE v*)) and t_1.key = t_2.key)
```

Run the example and convince yourself that the answer is what you want.

The subject of this second project is to implement the SQL query processing when the tables are maintained as columnar distribution.

The idea is to create the illusion that the tables are stored as sets of records, but in reality store tables as collections of two-attribute tables. Of course we will consider a very simple case (although the whole scheme works in the general case!). Here is what you will implement

1. First, we need to make sure that the user will be able to insert the data. We will do this in a manner very similar to the first project. The user will be able to insert a table in the following format. The first row is the metadata (attribute names and their types). The remaining rows will be data. You will assume that there are no nulls (error message when a null value occurs). The value separator will be ; and row separator will be *. You will assume that the maximum number of columns is 10.
2. Once the data (a single table) is accepted, you will transform it into columnar partition as described above. To do so, you will add the column called k, and then project to the two-attribute tables; one of these attributes is the key you created
3. You will accept only simple queries (although the real columnar-based systems accept arbitrary SQL). To make it (slightly) harder than in Project 1, the queries you accept will have selection clauses which are conjunctions of:
    1. Equalities of the form A = a (A is an attribute name, a, an attribute value),
    2. Simple LIKE statements (A LIKE s*) where s is a string
    3. Negations of the literals of the previous two kinds

Software requirements

1. You need to build an interface that accepts from the user data on tables. It could resemble MS Access interface, or you can introduce some other mechanism.
2. A user will be able to enter a query
3. You will show the transformed query
4. You will also show the answer to transformed query (which, of course, should be identical to the answer to the original query)

This project is worth 30% of the credit for the part II of this course. Devote sufficient effort to this project!

Submission requirements

- You must submit the code, documentation, and test cases in hard copy or email by **Monday , November 26, 2012**.
- I might ask you to demonstrate your program *at my discretion*. If so, I will let you know by **Wednesday, November 28**. I will base my decision on your code and documentation. If needed, demonstration will be made that same week.

B.t.w. The company called Vertica was formed around that idea and eventually was bought by HP for 9-digit number.

Set up: August 28, 2012

Last updated: October 23, 2012