Exercises Data Warehousing
Indexing

Solutions

1. Consider the following relation `Cars`:

`Brand`	`Type`	`Color`	`Risk`
Opel	Corsa	Grey	Low
Opel	Corsa	Red	Medium
Peugeot	206	Black	Medium
BMW	A	Black	High

   (a) Construct a bitmap index for the attributes `Brand` and `Color` for this table.
   Solution

Brand

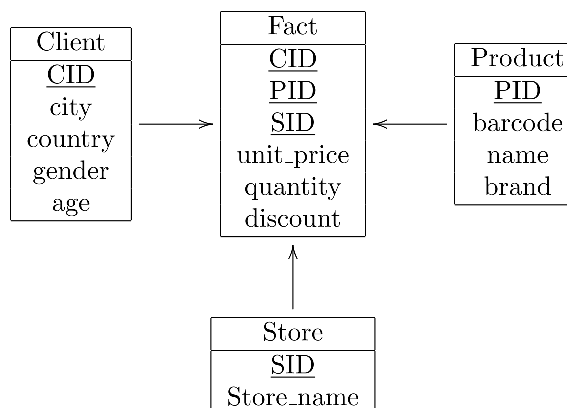
`Opel`	`Peug.`	`BMW`
1	0	0
1	0	0
0	1	0
0	0	1

Color

`Grey`	`Red`	`Black`
1	0	0
0	1	0
0	0	1
0	0	1

   (b) Indicate how these two bitmap indices can be used to answer the query: *Give the total number of red Opel cars with a medium risk score.*
   Solution: First we intersect the bitmap for `Opel` with the bitmap index for `Red` by taking the logical **AND** of the bitmaps, resulting in the bitmap: (0 1 0 0). Then, for all 1-entries, we directly access the corresponding tuple in the relation, check the condition `Risk=medium`, and if this condition is satisfied, we count the tuple. Hence, in this case, we access the second tuple only. As the tuple satisfies the condition, it is counted. The final result 1 is returned.

2. Consider the following Star schema for a ROLAP database. A fact (c, p, s, up, q, d) represents that customer c bought q units of product p in store s at unit price up and received a discount d.



   (a) Discuss the advantages and disadvantages of the following indices; for instance, describe under what circumstances and for which types of queries are these indices interesting.
      i. a bitmap join index for the attribute country from the table Client into the fact table (mapping countries to tuples in the fact table that are about this country);
      Solution: The advantage of this index is that it will speed up queries that slice on a country as it will allow to immediately select the disk blocks in which facts are stored that are needed to answer the query. One caveat, however: if there is only an index on country, it is likely that this index will not be sufficiently selective because, depending on the block size, every block may contain every country. Hence, in such a situation, unless the index can be used in combination with other bitmap or bitmap-join indices, the index will not be helpful. So, in summary, this index is probably only useful in combination with other bitmap indices.
      ii. a bitmap index on table Client for the attribute gender; and,
      Solution: Again a similar argumentation as for the previous index applies, because there are only two possible values for gender. Therefore, a bitmap index on gender will only be useful in combination with other bitmap indices on the same table, such as for instance on city.

iii. a bitmap index for the attribute unit_price in table Fact.
Solution: This index is clearly suboptimal as there will be many different values in unit_price. In such a situation it is always better to use for instance a btree index. Furthermore, in this case unit_price is a measure, not a dimension. Hence it will be unlikely that there will be many queries making selections on the basis of unit_price, which renders an index on this attribute useless.

(b) Consider the following query:

```
SELECT Client.Gender, SUM(Fact.quantity)
FROM Fact, Client
WHERE Fact.CID=Client.CID AND
      (Client.country = "The Netherlands"
       OR Client.country = "Belgium")
group by Client.Gender
```

Select one of the indices from (a) and explain how this index may help to answer this query efficiently.
Solution: The only candidate is the first bitmap-join index. This index allows to select only those facts that are about Belgium and The Netherlands. For these facts, however, we will have to lookup the associated clients in the client dimension table because we need to know the gender in order to get access to the gender. Hence, it is unlikely that any of the indices above will actually help, unless the selection `(Client.country = "The Netherlands" OR Client.country = "Belgium")` is very selective and there is a (BTree) index on CID in the client table that allows to retrieve tuples in the Client table by CID instead of by a full table scan.

3. Deduplication is an important task in data cleaning. To do deduplication it is useful to have a measure of distance between string values such as names, addresses, phone numbers, etc. The edit distance is one such distance measure. Compute the edit distance between “Brussel” and “Bruges”.

Solution:

		B	**R**	**U**	**G**	**E**	**S**
	0	1	2	3	4	5	6
B	1	**0**	1	2	3	4	5
R	2	1	**0**	1	2	3	4
U	3	2	1	**0**	1	2	3
S	4	3	2	1	**1**	2	3
S	5	4	3	2	**2**	2	3
E	6	5	4	3	3	**2**	3
L	7	6	5	4	4	4	**3**

The edit distance is **3** as indicated in the botton right corner. By following the path in **bold** from the top left corner to the right bottom corner we can identify the required operations. The first three **0** mean that no operation is required since the 3 first letters are equal. The next **1** means that the **S** was replaced by a **G**. The first **2** means that the second **S** was deleted. The second **2** means that no operation was needed since the two characters are equal to **E**. Finally, the last **3** means that the **L** was replaced by an **S**. Notice that the path in bold is just **one** among multiple paths of length **3**.

4. Consider a cube with three dimensions A, B, and C, and one measure M. Suppose that the complete cube needs to be materialized for the aggregation function M. That is, given the base table B(IDA, IDB, IDC, M), we need to compute:

```
SELECT IDA, IDB, IDC, sum(M)
FROM B
GROUP BY CUBE(IDA, IDB, IDC)
```

Show how you could apply the pipe-sort algorithm to compute the cube. Assume that sorting a relation X takes SORT(X) time, and scanning the table takes FTS(X) (FTS stands for “full table scan”). Express the time needed by the query plan developed by the pipe-sort algorithm with the time needed by a brute-force solution that computes the aggregations for all grouping sets separately. What is the gain? (Express costs and gain in terms of SORT(X) and FTS(X) for X any of the aggregation tables constructed along the way)

Solution: The pipe-sort algorithm is based on the idea that when a relation is sorted, for example, in attributes ABC it is also sorted on attributes AB and A. Therefore, with a single scan it is possible to compute the aggregation on ABC, AB, A and {} in a pipe-line fashion.

For the `GROUP BY CUBE` we need to compute the aggregation for all $2^n = 8$ subsets of ABC. By sorting on ABC we have can compute the aggregation of 4 of these subsets. By sorting on BC we can compute the aggregation on BC and B. Finally, we need to compute the aggregation on both AC and C. The cost for computing the cube would be

$$\begin{array}{l} \text{SORT(ABC)} + \text{FTS(ABC)} \rightarrow \text{compute ABC, AB, A, \{\}} \\ \text{SORT(BCA)} + \text{FTS(BCA)} \rightarrow \text{compute BC, B} \\ \text{SORT(CAB)} + \text{FTS(CAB)} \rightarrow \text{compute CA, C} \\ \text{SORT(CA)} \rightarrow \text{obtain AC} \end{array}$$

Compare this with respect to the brute-force solution SORT(X) + FTS(X) for all eight subsets of ABC.

5. Suppose that a cube Sales(A, B, C, D, Amount) has to be fully materialized. The cube contains 64 tuples. Sorting takes the typical $n \log(n)$ time. Every GROUP BY with k attributes has 2^k tuples.

   (a) Compute the cube using the PipeSort algorithm.

   Solution: A possible evaluation plan to fully materialize the cube is given in Fig. 1.

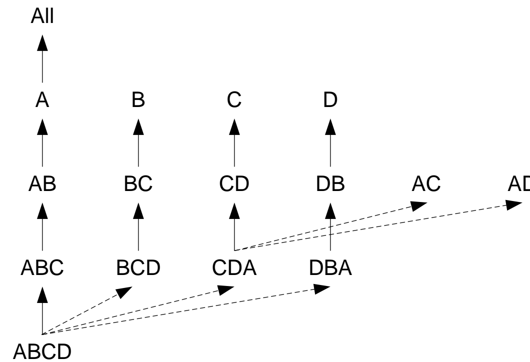


Figure 1: Evaluation plan to fully materialize the cube Sales(A, B, C, D, Amount)

   (b) Compute the gain of applying the PipeSort compared to the cost of computing all the views from scratch.

   Solution: There are 16 views. Since the cube has 64 tuples, suppose that the views with 3 attributes (such as BCD) have 32 tuples and the views with 2 attributes (such as AC) have 16 tuples. Suppose also that scanning a table of n tuples costs n time units, and that sorting a table of n tuples costs $n \log(n)$ time units.

   The cost of applying the PipeSort algorithm is as follows (see Fig. 1). The first scan of 64 tuples costs 64 units. Then, the following two operations are performed three times: a sort of the table for computing a view with three attributes, and a pipeline for computing all views derived for that. The cost of these operations is $3 \times (32 \log(32) + 32)$. Finally, the following two operations are performed two times: a sort of the table for computing a view with two attributes, and a pipeline for computing all views derived for that. The cost of these operations is $2 \times (16 \log(16) + 16)$. Thus, the total cost of the PipeSort algorithm is

$$64 + 3 \times (32 \log(32) + 32) + 2 \times (16 \log(16) + 16) = 64 + 576 + 152 = 792$$

   On the other hand, the cost of computing each of the 16 views independently would imply a sort operation and scanning of the table. Thus, the total cost would be

$$\begin{array}{l} 4 \times (32 \log(32) + 32) + 6 \times (16 \log(16) + 16) + \\ \qquad 4 \times (8 \log(8) + 8) + 8 = 768 + 456 + 128 + 8 = 1360 \end{array}$$

   Therefore, in this example, the cost of the view computation with the PipeSort algorithm will be 58% of the cost of computing the views independently.