Database Systems
- Query Processing and Optimization 2 -

Prof. Dr. Agnès Voisard
Institute of Computer Science,
Databases and Information Systems Group
and Fraunhofer FOKUS

2025
v1

Notes

Notes

Estimation of Costs of Access using Indices

So far we did not consider the effects of indices and hash functions on the cost of evaluating an expression.
Presence of these structures has a significant influence on the choice of a query processing strategy.

- **Indices and hash functions allow fast access to records containing a specific value on the index key.**
- **Indices allow the records of a file to be read in sorted order.**
  Efficient to read records in an order corresponding closely to physical order.
  (If it is the case: *clustering index*, physical clustering of records into blocks).

Estimation of Costs of Access using Indices (cont'd)

Detailed strategy for processing a query: *Access plan* for the query. Includes

- The relational operations to be performed
- The indices to be used
- The order in which tuples are to be accessed
- The order in which operations have to be performed.

Remark: take also into account the overhead to those blocks containing indices when estimating a cost.

Structure of the Query Optimizer

Most systems implement only a few strategies.
For each strategy a cost estimate is computed.

Some systems reduce the number of strategies that need to be fully considered by making heuristic guess of a good strategy.

The optimizer considers every good strategy and terminates as soon as it determines that the cost is greater than the best previously considered strategy.
To simplify the strategy selection task: split a query into several subqueries.
(saved when it appears several time, both in the optimization phase and in the execution phase).

Join Strategies

Factors that influence the selection of an optimal strategy:

- Physical order of tuples in a relation
- Presence of indices and type of index
- Cost of computing a temporary index for processing only one query
  - ◇ Simple iteration
  - ◇ Block-oriented iteration

  (relation processed in a per-block basis instead of a per-tuple basis)
  - ◇ Merge-join
  - ◇ Use of an index, Hash join
  - ◇ + join strategies for parallel processors

Example

Consider the query:

```
SELECT     AccountNum
FROM       DEPOSIT
WHERE BranchName = ‘‘Perryridge’’
  AND Cname = ‘‘Williams’’
  AND  Balance > 1000
```

Example (cont'd)

... and the statistical information:

- 20 tuples of deposit fit in one block
- V(BranchName, deposit) = 50
- V(Cname, deposit) = 200
- V(Balance, deposit) = 5000
- $n_{deposit}$ = 10,000

Assume on deposit:

- Clustering B^+-tree index for BranchName
  tuples with same key are stored together.
- Nonclustering B^+-tree index for Cname.

Assumption: values distributed uniformly.

Example (cont'd)

Comparison of blocks reads.

A. Using the clustering index

50 different values for Branchname.

We expect that 10,000/50 = 200 tuples of the deposit relation pertain to the Perryridge branch.

Read (check) all of them for the WHERE clause.

Clustering index: 200/20 = 10 block read (200 tuples, 20 tuples/block).

Assume B+-tree index stores 20 pointers per node.

B+-tree has between 3 and 5 leaf nodes.

Tree has a depth of 2: read 2 index blocks.

=> This strategy requires 12 block reads.

Example (cont'd)

B. Using the nonclustering index (on customer name)
V(Cname, deposit) = 200.
Estimate: 10,000 / 200 = 50 tuples of deposit belong to Williams.
Nonclustering index: 1 block read required for each tuple.
(50 block reads for the deposit tuples).

If 20 pointers in the B+-tree index, 200 Cnames, tree has between 11 and 20 leaf nodes. Depth 2, 2 block accesses in the index.

=> This strategy requires 52 block reads.

Example (cont'd)

Other way to process the query.

Use the index for Cname to retrieve the *pointers* to records with *Cname* = "*Williams*".
P1 = set of these pointers.

Same thing with P2 = pointers to records with *BranchName* = "*Perryridge*".

P1 $\cap$ P2 = set of pointers to records with both conditions.
They must be retrieved and tested to see if *Balance* $>$ 1000.

Example (cont'd)

This technique requires both indices to be accessed.
4 index blocks must be accessed.
Estimation of blocks to be read from the deposit file:
Estimation of number of pointers in P1 $\cap$ P2.

Estimation: 1 tuple out of 50 x 200 (10,000) has both conditions.
(uniform distribution + independent).
P1 $\cap$ P2 = 1 pointer. => read only 1 block.

Total estimate: 5 blocks.

Example (cont'd)

We did not consider using the *Balance* attribute and the predicate *Balance* $>$ 1000:

- There is no index for balance
- Selection on *Balance* involves a "greater than" comparison.
  In general equality predicates are more selective than "greater than" ones.
  Since we have an equality predicate (2) it is better to use it (selection of fewer tuples).

Remarks on these estimates

when using indices: estimate the complete cost of a strategy in terms of block accesses.

For a given relational algebra expression, it may be possible to formulate several strategies.

What will come next?

1. Welcome to Database Systems
2. Introduction to Database Systems
3. Entity Relationship Design Diagram (ERM)
4. Relational Model
5. Relational Algebra
6. Structured Query Language (SQL)
7. Relational Database Design - Functional Dependencies
8. Relational Database Design - Normalization
9. Online Analytical Processing + Embedded SQL
10. Data Mining
11. Physical Representation - Storage and File Structure
12. Physical Representation - Indexing and Hashing
13. Transactions
14. Concurrency Control Techniques
15. Recovery Techniques
16. Query Processing and Optimization