Query Optimization
Costs-I

Knowledge objectives

1. Explain and exemplify the two main assumption that most DBMSs do on estimating the cardinality of query results
2. Explain the need and problem of gathering statistics
3. Explain how several indexes can be used to solve a complex selection predicate over one table
4. Compare the use of RID lists and bitmaps in solving a complex selection predicate over one table
5. Compare the use of RID lists and multiattribute indexes in solving a complex selection predicate over one table
6. Enumerate three ways to have a table sorted
7. Enumerate seven operations that involve sorting in query processing
8. Explain the merge-sort algorithm
9. Explain how duplicates can be removed from a table depending on the data structure it has (i.e., plain file, B+, Hash, Clustered index)

Understanding objectives

1. Estimate the number of tuples in the output of a query, given the schema of the database and the tuples in the tables
2. Calculate the number of blocks necessary to store a given number of tuples, knowing the size of the attributes of each tuple and the bytes of each block
3. Find which indexes would be used to solve a multi-clause selection predicate over one table
4. Decide if a multi-attribute index can be used to solve a multi-clause selection predicate over one table
5. Calculate the approximate cost of a merge-sort operation, given the available memory, the number of tuples in the table, and the number of tuples that fit in a disk block
6. Calculate the approximate size of a table, given its number of tuples, its structure, and the number of tuples and entries that fit in a disk block

Intermediate results estimation

Cardinality

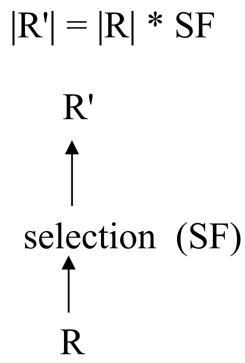
Size

Cost-based optimization steps

1. Generate alternatives in the search space
   a. Join order
   b. Potential algorithms
   c. Available structures (access path)
   ~~d. Materialization or not of intermediate results~~
      - We will assume that they are always materialized
2. Evaluate those alternatives
   1. Intermediate results cardinality and size estimation
   2. Cost estimation
3. Choose the best option
4. Generate the corresponding access plan

Cardinality estimation

- Based on the selectivity factor (0≤SF≤1)
  - General rule: cases in the result / maximum possible cases
    - Next to 0 means very selective (e.g., ID)
    - Next to 1 means little selective (e.g., Sex)
- SF is only needed for selection and join operations
  - Estimated cardinality for selection
    |selection(R)|= SF*|R|
  - Estimated cardinality for join:
    |join(R,S)|= SF*|R|*|S|
  - Estimated cardinality for union:
    - With repetitions:
      |union(R,S)|= |R|+|S|
    - Without repetitions:
      |union(R,S)|= |R|+|S|- |join(R,S)|
  - Estimated cardinality for difference (anti-join):
    |difference(R,S)|= |R| - |join(R,S)|
- Calculated from the leaves to the root
  - Table statistics are needed to estimate cardinalities



Statistics

- DBA is responsible for the statistics to be fresh
- Example of kinds of statistics
  - Regarding relations:
    - Cardinality
    - Number of blocks
    - Average length of records
  - Regarding attributes:
    - Length
    - Domain cardinality (maximum number of different values)
    - Number of existing different values
    - Maximum value
    - Minimum value
- Main hypothesis in most DBMS
  - Uniform distribution of values for each attribute
  - Independence of attributes

Statistics computation

Oracle

DBMS_STATS.GATHER_TABLE_STATS(<esquema>, <table>);

```
DBMS_STATS.GATHER_TABLE_STATS(“username”, “departments”);
DBMS_STATS.GATHER_TABLE_STATS(“username”, “employees”);
```

PostgreSQL

ANALYZE [<table_and_columns>];

```
ANALYZE departments;
ANALYZE employees(dni, name);
```

https://www.postgresql.org/docs/current/sql-analyze.html

Selectivity factor of a selection (I)

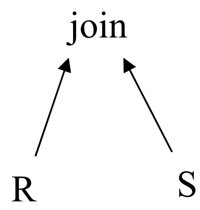
- Assuming equi-probability of values
  - SF(A=c) = 1/ndist(A)
- Assuming uniform distribution, the attribute is a real number and $A \in [min,max]$
  - SF(A>c) = (max-c)/(max-min)
    - SF(A>c) = 0 (if $c \geq max$)
    - SF(A>c) = 1 (if $c < min$)
  - SF(A>v) = ½
  - SF(A<c) = (c-min)/(max-min)
    - SF(A<c) = 1 (if $c > max$)
    - SF(A<c) = 0 (if $c \leq min$)
  - SF(A<v) = ½
- Assuming ndist(A) big enough
  - SF($A \leq x$) = SF(A<x)
  - SF($A \geq x$) = SF(A>x)

Selectivity factor of a selection (II)

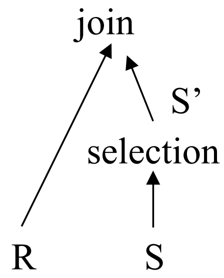
- Assuming P and Q statistically independent
  - SF(P AND Q) = SF(P)·SF(Q)
  - SF(P OR Q) = SF(P)+SF(Q)-SF(P)·SF(Q)
- SF(NOT P) = 1-SF(P)
- SF(A IN (c_1,c_2,…,c_n)) = min(1, n/ndist(A))
- SF(A BETWEEN c_1 AND c_2) = (min(c_2,max) – max(c_1,min))/(max-min) (Not for integers)
- SF(A BETWEEN v_1 AND v_2) = ¼
- SF(A BETWEEN c_1 AND v_2) = ½SF(A>c_1)
- SF(A BETWEEN v_1 AND c_2) = ½SF(A<c_2)

Selectivity factor of a join (I)

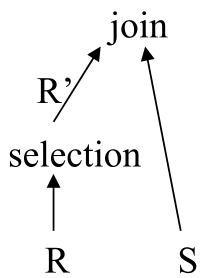
- $SF(R[A=B]S) = 1/|R|$
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



- $SF(R[A=B]S') = 1/|R|$
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



- $SF(R'[A=B]S) = SF_R*(1/|R'|) = 1/|R|$
  - R' = selection(R), having $SF = |R'|/|R| = SF_R$
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



Selectivity factor of a join (II)

- It is difficult to approximate the general case R[AθB]S

- Depending on the comparison operator:

  - SF(R[AxB]S) = 1

  - SF(R[A<>B]S) = 1

  - SF(R[A<B]S) = ½

  - SF(R[A=B]S) = 1/max(ndist(A),ndist(B))
    - Useful without FK, but still domains overlap
      - Good approximation if at least one of the attributes is uniformly distributed
      - More accurate if one domain is subset of another

A. S. Rosenthal. *Note on the expected size of join*. SIGMOD Record, 1981

Intermediate results estimation

- Besides the cost of executing each operation, we also need to know the size of its results (i.e., the cost of writing intermediate results into a temporal file)
  - Record length
    - $\sum$ attribute_length$_i$ (+ control_information)
  - Number of records per block
    - $R_R = \lfloor \text{block\_size/record\_length} \rfloor$
  - Number of blocks per table
    - $B_R = \lceil |R|/R_R \rceil$

Selection of complex predicates

Using RID lists

Using Bitmaps

Using a multi-attribute index

Selection of complex predicates using RID lists

1. Put the predicate in Conjunctive Normal Form (CNF)
    1. Remove negations of parenthesis
        - NOT (A OR B) = NOT A AND NOT B
        - NOT (A AND B) = NOT A OR NOT B
    2. Move disjunctions into the parenthesis
        - (A AND B) OR C = (A OR C) AND (B OR C)
        - (A AND B) OR (C AND D) = (A OR C) AND (A OR D) AND (B OR C) AND (B OR D)
        - (A AND B) OR (C AND B) = (A OR C) AND B
2. Remove disjunctions if possible
    - For each parenthesis, if indexes can be used for all conditions in it
        - Unite the RID lists resulting from accessing those indexes
3. Remove conjunctions if possible
    - For all parenthesis resolved in the previous step, intersect the RID lists produced
4. For each RID (if any) obtained from previous step
    - Go to the table (<u>by following the RID</u>) to check the remaining predicates

Considerations on using RID lists

Let's suppose that we have a predicate in CNF with disjunctions:

(A_1 op_1 v_1 OR A_2 op_2 v_2) AND .... AND (A_p op_p v_p)

being A_i attributes of the same relation and op_i a comparison operator

- If one of the conditions inside the parenthesis does not allow an index to be used, we cannot use any other index
- If no index can be used at all, we should perform a table scan
- Some kinds of indexes are not useful for some comparisons
  - B+ is useless for "different from"
  - Hash is useless for "different from" and inequalities

Example of selection using RID lists

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$

$\Downarrow$

(A OR C) AND (A OR D) AND (B OR C) AND (B OR D)

$\cup \qquad \cup$

$\cap$

For each element in the list of RID resulting from the intersection

Go to the table's file and check "(A OR D) AND (B OR D)"

Cost of a selection of complex predicate using RID lists

u = %load·2d = (2/3)·2d
$h = \lceil \log_u |T| \rceil - 1$
O = output table
v = values indicated in the clause
k = repetitions per value of the attribute
$|O| = SF * |T|$

- No index can be used
  - B (or $\lceil 1.5B \rceil$ if the table is in a cluster)
- An index can be used
  - We estimate the cost as one access per row in the output (i.e., |O|)
  - Sum to |O| the cost of using every index
    - B+
      - h + (v*k-1)/u
    - Hash
      - v

Selection using bitmaps

- A bitmap is a set of lists of bits
  - Each list corresponds to a different value in the indexed attribute
  - Each bit indicates the presence/absence of the corresponding value in a row
    - The overall number of bits in the bitmap is $ndist(A) \cdot |T|$
- A bitmap can be used to evaluate complex predicates
  - Set operations ($\cup$, $\cap$) of RIDs are translated into logic operators (OR, AND) of bits
    - Bit operations are much more efficient
- A bitmap cannot be used to compare inequalities

Example of bitmaps instead of RID lists (I)

Ballpoint	Pencil	Pen	Rubber	A4 paper	A3 paper	Chalk	Eraser
1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0

Catalunya	León	Madrid	Andalucía
1	0	0	0
1	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1
0	1	0	0
1	0	0	0
1	0	0	0

Example of bitmaps instead of RID lists (II)

SELECT COUNT(*)

…

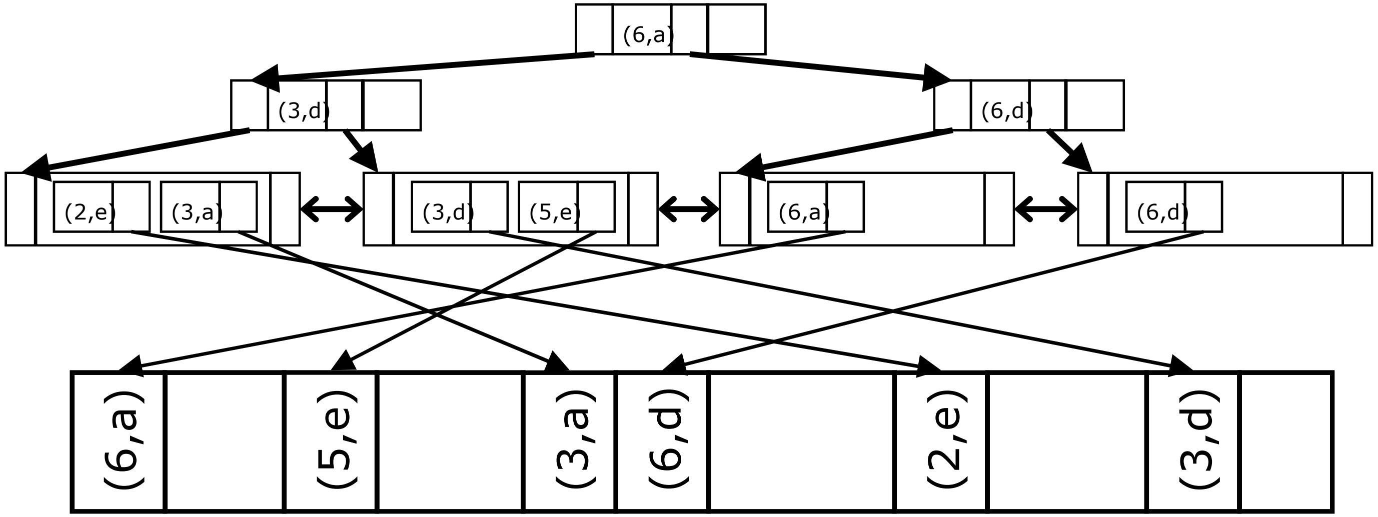
WHERE articleName IN ['Ballpoint','Pencil'] AND region='Catalunya'

Ballpoint		Pencil				Catalunya		
1		0		1		1		1
0		0		0		1		0
0		1		1		0		0
0		0		0		0		0
0	OR	0	=	0	AND	0	=	0
1		0		1		1		1
0		0		0		0		0
0		0		0		0		0
0		0		0		1		0
0		1		1		1		1

Usefulness of multi-attribute tree indexes

- Need more space
  - For each tuple, keeps attributes A_1, .., A_k
  - May result in more levels, worsening access time
- Modifications are more frequent
  - Every time one of the attributes in the index is modified
- It is much more efficient than intersecting RID lists (to evaluate conjunctions)
- Can be used to solve several kinds of queries
  - Equality of all first *i* attributes
  - Equality of all first *i* attributes and inequality (a.k.a. range) of *i+1*
- The order of attributes in the index matters
  - We cannot evaluate condition over A_k, if there is no equality for A_1, .., A_{k-1}

Example of multi-attribute tree index



- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b' YES
  - Num='3' YES
  - Num>'3' AND Let='a' NO
  - Num>'3' AND Let>'b' NO
  - Num>'3' YES
  - Let='e' NO
  - Let>'b' NO
  - Num='3' OR Let='a' NO

Sorting

Cost estimation

Usefulness of sorting algorithms

- ORDER BY
- Duplicates removal
  - DISTINCT
  - UNION
- GROUP BY
- Join
- Difference (anti-join)
- Massive index load

External sorting algorithms

- No index, with M+1 memory pages
  - $2B\cdot\lceil \log_M B \rceil - B$
- B+
  - $\lceil |T|/u \rceil + |T|$
- Clustered
  - $\lceil 1.5B \rceil$
- Hash
  - Useless

$u = \%load\cdot 2d = (2/3)\cdot 2d$

External Merge Sort (I)

- Function *sort()*

Assumption: Data is already in memory
Result: Writes the memory pages (blocks) sorted

Disk accesses: B_T

- Function *scan(T)*

Assumption: We have B_T memory pages
Result: T has been read into memory

Disk accesses: B_T

External Merge Sort (II)

- Function *merge($T_1, \ldots, T_M$)*

min = index (1..M) of the T_i with the minimum current value

first ⇒reads the first block into memory
next ⇒ reads a new block if the memory page is empty
+= ⇒ writes a block if buffer is full

Assumption: T_i are sorted and we have M+1 memory pages

Result: Writes the sorted union of all T_i

t_1 := first(T_1); t_2 := first(T_2); … t_M := first(T_M);
<u>while</u> not (end(T_1) and end(T_2) and … and end(T_M))
    T^{ord} += t_{min};
    t_{min} := next(T_{min});
<u>endWhile</u>

Disk accesses: $2(B_{T_1} + B_{T_2} + \ldots + B_{T_M})$

External Merge Sort (III)

- Function *mergeSort(T)*

Assumption: We have M+1 memory pages
Result: Sorted T

```
if  B_T <= M then
    scan(T); T^ord:=sort();
else
    T_1^ord :=mergeSort(T_1); … T_M^ord :=mergeSort(T_M);
    T^ord :=merge(T_1^ord, …, T_M^ord);
endif
```

Disk accesses: $2B_T \cdot \lceil \log_M B_T \rceil$

$B <= M \rightarrow 2B = 2B \cdot 1$

$M < B <= M^2 \rightarrow 2B + 2B = 2B \cdot 2$

$M^2 < B <= M^3 \rightarrow 4B + 2B = 2B \cdot 3$

$M^3 < B <= M^4 \rightarrow 6B + 2B = 2B \cdot 4$

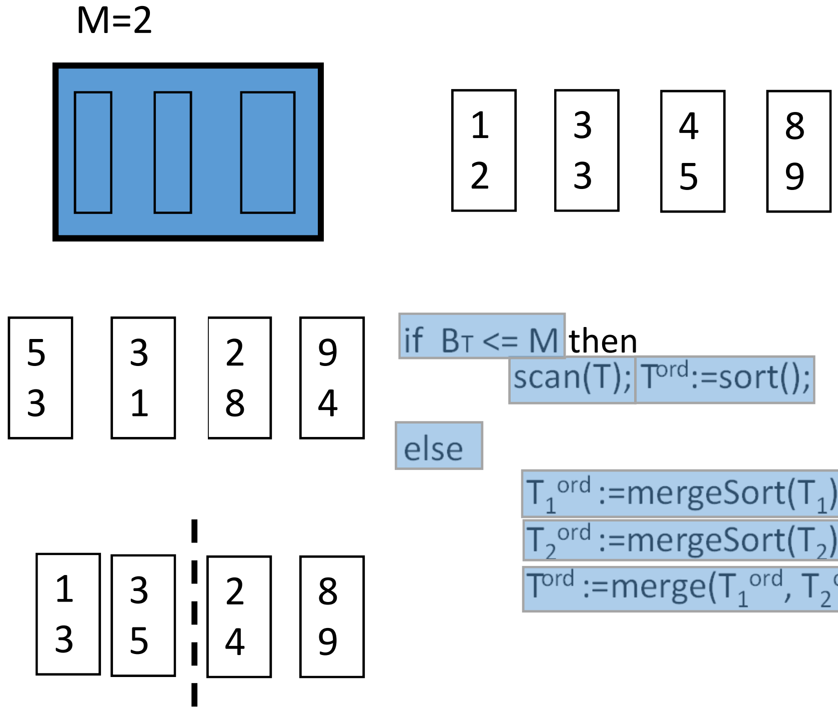
…

B_T is the real number of blocks (taking into account possible cluster). For the sake of simplicity, we don't distinguish weather output has empty spaces or not

Example of External Merge Sort

Accesses

R
R
W
W
R
R
W
W
R
R
R
R



if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

Projection

Cost estimation

Projection algorithms

1. Attribute removal
   a) There is another operation
      - 0
   b) There is no other operation
      - B
2. Duplicate removal
   a) No index, with M+1 memory pages
      - $2B \cdot \lceil \log_M B \rceil - B$
   b) B+, useful if M and R are small with regard to B
      - $\lceil |T|/u \rceil$ (probably $+|T|$)
   c) Clustered
      - $\lceil 1.5B \rceil$
   d) Hash, useful if M and R are small with regard to B
      - $\lceil 1.25(|T|/2d) \rceil$ (probably $+|T|$)

$u = \%load \cdot 2d = (2/3) \cdot 2d$

Closing

Summary

- Intermediate results
    - Cardinality estimation
    - Size estimation
- Cost estimation
    - Selection algorithms
    - Sorting algorithms
    - Projection algorithms

Bibliography

- A. S. Rosental. *Note on the expected size of a join*. SIGMOD Record, 11 (4), 2981
- Y. Ioannidis. *Query Optimization*. ACM Computing Surveys, vol. 28, num. 1, March 1996
- G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, 1998
- J. Sistac. *Sistemes de Gestió de Bases de Dades*. Editorial UOC, 2002
- R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill, 3rd Edition, 2003
- J. Lewis. *Cost-Based Oracle Fundamentals*. Apress, 2006
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007