Announcement

Updated schedule

10/12 Guest lecture from TiDB

...  
10/24 Guest lecture from Oracle

...  
11/7 Midterm review
11/9 Midterm exam

Course project

Proposal deadline is 10/24. Please start to form teams (2–4 people) asap.
Access Path Selection in a Relational Database Management System

P. Kocher, R. B. Hamann, P. A. Leslie, V. V. Prasad
IBM research Division, San Jose, California 95193

Abstract: In a high level query and data manipulation language such as SQL, request
are stored monotonically without restric-
tions on the order or form of the request. This
New System 2 chosen access paths for both
simple, unstructured, and complex
queries. In many cases, the queries concern
a relation of related data as a function
algorithm is an experimental database management system
written at IBM Research Division. Access Path
Selection in System 2 has been studied and
built by members of the IBM San Jose
Research Laboratory.

1. Introduction

System 2 is an experimental database manage-
ment system written at the IBM Research Division.
Access Path Selection in System 2 has been studied
and built by members of the IBM San Jose
Research Laboratory since 1977. The software was developed
without access path selection and is not generally available out-
side the IBM Research Division.

This paper concerns itself with rela-
tional data model, terminology, and description
of the System 2 Access Path
Selection algorithm. Access Path
Selection in System 2 can be traced both from an
algorithmic, and implementation perspective.

In System 2 a user need not know how the
indexes are physically stored and why
access paths are available. A SQL query
only requires the user to specify anything about
the access paths that he wishes to be used for
given access path selection. This paper is divided
into four sections:

1. Introduction
2. Processing of an SQL statement
3. Execution Plan.
4. Conclusion

1. Introduction

Access Path Selection in System 2 is

Copyright © 1979 by the ACM, Inc. Subject to
permission, reproduction and redistribution
of digital copies of this article are permitted
without fee provided this copyright note and
the full citation are included. The ACM's
Digital Library articles are the sole property
of the ACM. The ACM's Digital Library
is administered by the ACM. SIGMOD
Conference is published by the American
Database Association, a division of the
Association for Computing Machinery.

SIGMOD 1979
Agenda

System R

Query Optimization in R
  • Cost estimation
  • Plan enumeration
System R

1. Parsing
2. Optimization
3. Code generation
4. Execution
Query Optimization

SQL query -> Query Parser -> unoptimized plan

Query Optimizer
- Plan generator
- Plan cost estimator

System Catalog

-> optimized plan
Query Optimization in System R
System R Storage Architecture

Cost = IO cost + Computation cost
     = \#I/Os + W * RSICARD

RSICARD = \#tuples through the RSI interface

Goal: enumerate execution plans and pick the one with the lowest cost
<table>
<thead>
<tr>
<th>Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NCARD(T)</strong></td>
<td># tuples in T</td>
</tr>
<tr>
<td><strong>TCARD(T)</strong></td>
<td># of pages containing tuples in T</td>
</tr>
</tbody>
</table>
| **P(T)**            | Fraction of segment pages that hold tuples of T.  
                      \[ P(T) = \frac{\text{TCARD(T)}}{\# \text{non-empty pages in the segment}} \] |
| **ICARD(I)**        | # distinct keys in the index I                                    |
| **NINDEX(I)**       | # pages in index I                                                |
| High key value and  |                                                                    |
| low key value       |                                                                    |
| Modern systems      | Keep histogram on table attributes.                               |
Access Paths

Segment Scans
- A segment contains disk pages that can hold tuples from multiple relations
- Segment scan is a sequential scan of all the pages

<table>
<thead>
<tr>
<th>Page 1</th>
<th>A{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 2</td>
<td>A{...}</td>
</tr>
<tr>
<td>Page 3</td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 4</td>
<td>A{...}</td>
</tr>
<tr>
<td></td>
<td>B{...}</td>
</tr>
</tbody>
</table>
Access Paths

Segment Scans
- A segment contains disk pages that can hold tuples from multiple relations
- Segment scan is a sequential scan of all the pages

Index Scan
- Clustered index scan
- Non-clustered scan
- Scan with starting and stopping key values

<table>
<thead>
<tr>
<th>Page 1</th>
<th>A{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 2</td>
<td>A{...}</td>
</tr>
<tr>
<td>Page 3</td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 4</td>
<td>A{...}</td>
</tr>
<tr>
<td></td>
<td>B{...}</td>
</tr>
</tbody>
</table>
Predicates

Sargable predicates (**Search ARGuments-able**)
- Predicates that can be filtered by the RSS
- **I.e., column comparison-operator value**
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a **boolean factor**
Predicates

Sargable predicates *(Search ARGuments-able)*
- Predicates that can be filtered by the RSS
- I.e., *column comparison-operator value*
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a *boolean factor*

Examples of non-sargable
- function(column) = something
- column1 + column2 = something
- column + value = something
- column1 > column2
Predicates

Sargable predicates (Search ARGuments-able)

- Predicates that can be filtered by the RSS
- I.e., column comparison-operator value
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a boolean factor

A predicate matches an index if

1. Predicate is sargable
2. Columns referenced in the predicate match an initial subset of attributes of the index key

Example: B-tree Index on (name, age)

- predicate1: name=‘xxx’ and age=‘17’ match
- predicate2: age=‘17’ not match
Computation cost: RSICARD

Calculate the selectivity factor $F$ for each boolean factor/predicate
Computation cost: RSICARD

Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists
  - $F = 1/ICARD(index)$  # distinct keys
- else
  - $1/10$
Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists
  $$F = \frac{1}{\text{ICARD(index)}}$$  # distinct keys
- else
  $$\frac{1}{10}$$

**column1 = column2**
- $$\frac{1}{\text{Max(ICARD(column1 index), ICARD(column2 index))}}$$
Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists
  $$F = \frac{1}{\text{ICARD(index)}} \quad \# \text{distinct keys}$$
- else
  $$\frac{1}{10}$$

**column1 = column2**

$$\frac{1}{\text{Max(ICARD(column1 index), ICARD(column2 index))}}$$

Assumes each key value in the index with the smaller cardinality has a matching value in the other index.

For each record in relation 1, $\frac{(\text{NCARD2} / \text{ICARD2})}{\text{tuples in relation 2}}$ will satisfy the predicate.

Total number of selected tuples =

$$\frac{(\text{NCARD2} \times \text{NCARD1})}{\text{ICARD2}}$$

$$F = \frac{1}{\text{ICARD2}}$$
Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists
  \[ F = \frac{1}{ICARD(index)} \quad \# \text{distinct keys} \]
- else
  \[ 1/10 \]

**column1 = column2**
- \[ 1 / \text{Max}(ICARD(column1 \text{ index}), ICARD(column2 \text{ index})) \]

**column > value**
- \[ F = \frac{\text{high key value} - \text{value}}{\text{high key value} - \text{low key value}} \]
Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists: $F = 1/\text{ICARD(index)}$  # distinct keys
- else: $1/10$

**column1 = column2**
- $1 / \max(\text{ICARD(column1 index), ICARD(column2 index)})$

**column > value**
- $F = (\text{high key value} - \text{value}) / (\text{high key value} - \text{low key value})$

**pred1 and pred2**
- $F = F(\text{pred1}) \times F(\text{pred2})$

**pred1 or pred2**
- $F = F(\text{pred1}) + F(\text{pred2}) - F(\text{pred1}) \times F(\text{pred2})$

**Not pred**
- $F = 1 - F(\text{pred})$
IO cost

Calculate the number of pages access through IO
IO cost

Calculate the number of pages access through IO

**segment scan**

- \( IO = \frac{TCARD(T)}{P} \)  
  # segment pages
IO cost

Calculate the number of pages access through IO

**segment scan**
- \( \text{IO} = \frac{\text{TCARD}(T)}{P} \)  
  # segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)
- \( \text{IO} = 1 \text{ data page} + 1-3 \text{ index page} \)
IO cost

Calculate the number of pages access through IO

**segment scan**

- IO = TCARD(T)/P  \# segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)

- IO = 1 data page + 1-3 index page

**clustered index matching**

- IO = F(preds) * (NINDEX(I) + TCARD(T)) \# index pages & \# data pages
IO cost

Calculate the number of pages access through IO

**segment scan**

• \( \text{IO} = \frac{\text{TCARD}(T)}{P} \)  
  \# segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)

• \( \text{IO} = 1 \) data page + 1-3 index page

**clustered index matching**

• \( \text{IO} = F(\text{preds}) \times (\text{NINDEX}(I) + \text{TCARD}(T)) \)  
  \# index pages & \# data pages

**non-clustered index matching**

• \( \text{IO} = F(\text{preds}) \times (\text{NINDEX}(I) + \text{NCARD}(T)) \)  
  \# index pages & \# data page accesses
IO cost

Calculate the number of pages access through IO

**segment scan**
- IO = TCARD(T)/P  # segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)
- IO = 1 data page + 1-3 index page

**clustered index matching**
- IO = F(preds) * (NINDEX(I) + TCARD(T))  # index pages & # data pages

**non-clustered index matching**
- IO = F(preds) * (NINDEX(I) + NCARD(T))  # index pages & # data page accesses

**clustered index no matching**
- IO = NINDEX(I) + TCARD(T)
Final Cost

Cost = IO cost + Computation cost

= #I/Os + W * RSICARD
Access Path Selection for Joins

R \bowtie S

Method 1: nested loops
  • Tuple order within a relation does not matter

Method 2: merging scans
  • Both relations sorted on the join key
Access Path Selection for Joins

\[ R \bowtie S \]

Method 1: nested loops
- Tuple order within a relation does not matter

Method 2: merging scans
- Both relations sorted on the join key

Tuple order is an interesting order if specified by
- Group by
- Order by
- Equi-join key

Search space too large!
Search Space – Join Order

left-deep tree

Convention: right child is the inner relation
Search Space – Join Order

Convention: **right child is the inner relation**

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Convention: **right child is the inner relation**
For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Convention: **right child is the inner relation**

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining.

Bushy tree may produce cheaper plans but are rarely considered due to the explosion of search space.
Partial group by can also reduce cost

Example:

```
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples
Search Space – Group By

Partial group by can also reduce cost

Example:

```sql
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples
Modern query optimizers consider storage hierarchy and multicore? Retain fast performance in the huge search space?

- Multi-user, multi-core, multi-tier storage, different operators, etc.

What about distributed system?
Modify query plans in the middle of execution?
Consider the following schema and SQL query

Relation R (a, b): 10 million tuples, R.a is the primary key
Relation S (c, d): 100 million tuples, S.c is a foreign key referring to R.a

```
SELECT *
FROM R, S WHERE R.a = S.c AND R.d = 5;
```

c) [10 points] Please estimate the number of rows in the output relation using the techniques in Selinger’79 (Lecture 6, Query Optimization).
Submit review for

Mike Stonebraker, et al. C-store: a column-oriented DBMS, VLDB 2005