Access Path Selection in Main-Memory Optimized Data Systems:

Should I Scan or Should I Probe?

Michael S. Kester  Manos Athanassoulis  Stratos Idreos
Ask for what you want...

```
SELECT x
FROM table_A
WHERE
  y < 10;
```

...the system chooses how to retrieve it.
SQL

Optimizer

Data

Results
SELECT x
FROM table_A
WHERE
  y < 10;
SELECT x
FROM table_A
WHERE
  y < 10;
SELECT x
FROM table_A
WHERE y < 10;

select /* TPC_H Query 17 */
sum(l_extendedprice) / 7.0 as avg_yearly
from
lineitem,
part
where
p_partkey = l_partkey
and p_brand = '[BRAND]' 
and p_container = '[CONTAINER]' 
and l_quantity < ( 
select
0.2 * avg(l_quantity)
from
lineitem
where
l_partkey = p_partkey
)
select /* TPC_H Query 17 */
sum(l_extendedprice) / 7.0 as avg_yearly
from
lineitem,
part
where
p_partkey = l_partkey
and p_brand = 'BRAND'
and p_container = 'CONTAINER'
and l_quantity < ( select
0.2 * avg(l_quantity)
from
lineitem
where
l_partkey = p_partkey
)
```sql
SELECT x
FROM table_A
WHERE y < 10;
```

```sql
SELECT/* TPC_H Query 17 */
sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where p_partkey = l_partkey
and p_brand = 'BRAND'
and p_container = 'CONTAINER'
and l_quantity < (select 0.2 * avg(l_quantity)
from lineitem
where l_partkey = p_partkey )
```

```sql
SELECT F1.PARCELID, F1.FDATE FIRE1, F2.FDATE FIRE2,
F2.FDATE - F1.FDATE INTERVAL
FROM FIRES F1, FIRES F2
WHERE F1.PARCELID = F2.PARCELID AND F2.FDATE >
F1.FDATE AND F2.FDATE <= F1.FDATE + 300;
```
SELECT x
FROM table_A
WHERE y < 10;

select /* TPC_H Query 17 */
sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where p_partkey = l_partkey
and p_brand = '[BRAND]' and p_container = '[CONTAINER]' and l_quantity < (
select 0.2 * avg(l_quantity)
from lineitem
where l_partkey = p_partkey )

select user_id, count(*) as how_many
from bboard
where not exists (select 1 from bboard_authorized_maintainers bam
where bam.user_id = bboard.user_id)
and posting_time + 60 > sysdate
group by user_id
order by how_many desc;

SELECT F1.PARCELID, F1.FDATE FIRE1, F2.FDATE FIRE2,
F2.FDATE - F1.FDATE INTERVAL FROM FIRES F1, FIRES F2
WHERE F1.PARCELID = F2.PARCELID AND F2.FDATE > F1.FDATE AND F2.FDATE <= F1.FDATE + 300;

select user_id, count(*) as how_many
from bboard
where not exists (select 1 from bboard_authorized_maintainers bam
where bam.user_id = bboard.user_id)
and posting_time + 60 > sysdate
group by user_id
order by how_many desc;
SELECT x
FROM table_A
WHERE y < 10;

SELECT *
FROM exercise_logs
WHERE (minutes > 10
    AND calories > 50)
    OR heart_rate > 110;

select /* TPC_H Query 17 */
sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem,
part
where p_partkey = l_partkey
and p_brand = '[BRAND]'
and p_container = '[CONTAINER]'
and l_quantity < (select 0.2 * avg(l_quantity)
from lineitem
where l_partkey = p_partkey )

select F1.PARCELID, F1.FDATE FIRE1, F2.FDATE FIRE2,
F2.FDATE - F1.FDATE INTERVAL FROMFIRES F1, FIRES F2
WHERE F1.PARCELID = F2.PARCELID AND F2.FDATE >
F1.FDATE AND F2.FDATE <= F1.FDATE + 300;

SELECT F1.PARCELID, F1.FDATE FIRE1, F2.FDATE FIRE2,
F2.FDATE - F1.FDATE INTERVAL FROM FIRES F1, FIRES F2
WHERE F1.PARCELID = F2.PARCELID AND F2.FDATE >
F1.FDATE AND F2.FDATE <= F1.FDATE + 300;

select user_id, count(*) as how_many
from bboard
where not exists (select 1 from bboard_authorized_maintainers bam
where bam.user_id = bboard.user_id)
and posting_time + 60 > sysdate
group by user_id
order by how_many desc;
Access Path Choices

Full Base Data Scan

Secondary Index Scan
(Auxilliary copy of the data + structure)
Access Path Choices

Full Base Data Scan

Secondary Index Scan
(Auxilliary copy of the data + structure)
Selectivity 0%

Index is best
Scan is best

A fixed threshold
Selectivity Crossover

1960 2017?

Vertica / C-store
Vectorwise / MonetDB
SQLServer
IBM BLU
Oracle
SAP
...

Increasing pressure on indexes
Column(-group) Orientation

Compression
SIMD

Multi-Core

Vectorized Processing

C_1

C_2

C_3

C_4
1:1 Scan to Result

Shared Scans

S_1  S_2  S_3

r_1  r_2  r_3

S_1

r_1  r_2  r_3
$Q_0, Q_1, Q_2, \ldots Q_n$
Are indexes ever needed?

\( Q_0, Q_1, Q_2, \ldots Q_n \)
Are indexes ever needed?
Are indexes ever needed?
Are indexes ever needed?
If so, how should the optimizer choose an access path?
Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California
Selectivity

Scan is best

Historical Division

Index is best

Concurrent Read Queries

0%
Concurrency changes everything.
APS = Index scan

Sequential scan
APS = \text{Index scan} + \ ? + \ ?
APS = Index scan

\[ p \]
APS = \text{Index scan} + p + \text{black arrow}
APS = \[
\frac{\text{△} + ? + ? + ?}{\text{▁▁} + \begin{array}{c}
\text{Calculator} \\
p
\end{array} + \bullet}
\]
APS = \[
\begin{array}{c}
\text{△} + \text{△} + ? + p + ?
\end{array}
\]
APS = \frac{\text{Diagram 1} + \text{Diagram 2} + \text{Diagram 3} + \text{Diagram 4} + \text{Diagram 5}}{p + \text{Diagram 6}}
Multiquery APS = \[\sum p_i\]

Predicate evaluation

Result set writing
Multiquery APS = \[ \text{Diagram} + \text{Diagram} + \text{Diagram} + \text{Diagram} + \text{Diagram} \]
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + \[ \sum p_i \] + Result Writing

Predicate Eval.
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + \( \sum p_i \) + Result Writing
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + \[ \sum p_i \] + Predicate Eval. + Result Writing
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + Predicate Eval. + Result Writing

$\sum p_i$
Selectivity Crossover

Scan is best

Index is best

Concurrent read queries
Selectivity Crossover

Scan is best

Index is best

Concurrent read queries
Selectivity Crossover

100M Tuples | 8 Cores

Scan is best

Index is best

Concurrent read queries
Concurrency matters

- Scan is best
- Index is best

Concurrent read queries
Selectivity Crossover

Scan is best

Concurrent read queries

Large result count
Selectivity Crossover

Scan is best

Index is best

Concurrent read queries

100M Tuples | 8 Cores
100M Tuples | 8 Cores

- Selectivity Crossover
- Index is best
- Scan is best
- Concurrent read queries
- Shared scans help... to a point
100M Tuples | 8 Cores

Scan is best

Index is best

Two x 256

Single Batch

Selectivity Crossover

Concurrent read queries
8 Queries | 8 Cores | Mean of 10 trials

- Selectivity Crossover
- Scan is best
- Index is best

Relation Size

Selectivity Crossover
Selectivity Crossover

8 Queries | 8 Cores | Mean of 10 trials

Scan is best
Index is best

Relation Size

10^4
10^5
10^6
10^7
10^8
10^9
8 Queries | 8 Cores | Mean of 10 trials

Selectivity Crossover

Scan is best

Index is best
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + \( \Sigma p_i \) + Result Writing
Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + Predicate Eval. + Result Writing

\[ \sum p_i \]
Tree Traversal

Leaf Traversal

Result Writing

Sorting

Concurrency + data set size
Hardware Improvements

Dawn of time 2000

0% 1% 10%
Column Stores

Hardware Improvements

Dawn of time 2000 2010
“What-if” questions

2000  2010  2017  Future

Column Stores
Main Memory
Some problems you may be thinking.
Selectivity vs. Concurrent Queries

- **Selectivity**: 0%
- **Concurrent Queries**: 0%

- **Scan is best** when selectivity is high and concurrent queries are low.
- **Index is best** when selectivity is low and concurrent queries are high.

daslab.seas.harvard.edu/evosys
Thank you

daslab.seas.harvard.edu/evosys