scans vs indexes

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
**b-tree** - dynamic tree - always balanced

![Diagram of a b-tree with keys 1, 2, 3..., 12, 15, 17, 20, ..., 35, 50, ...]
select ... from R where A<v and ....

(secondary) **index vs scan**: the eternal battle
Just having indexes in the system is or can be useless... or even bad for performance.

Knowing when to use an index is key.
Primary index vs secondary vs scan?

Just having indexes in the system is or can be useless… or even bad for performance.

Knowing when to use an index is key.
design/implement numerous possible algorithms + data representations

choose the best data source, algorithms and path for each query
scan
scan

secondary index scan
A secondary index on A values out of order with base data

A query that select on A and then needs B

Intermediate out of order
A
a1
a2
a3
a4
a5
B
b1
b2
b3
b4
b5
C
c1
c2
c3
c4
c5

A
a5
a3
a2
a1
a4

secondary index on A
values out of order with base data

a query that select on A and then needs B
intermediate out of order
A | B | C
---|---|---
a1 | b1 | c1
a2 | b2 | c2
a3 | b3 | c3
a4 | b4 | c4
a5 | b5 | c5

Secondary index on A values out of order with base data.

A query that select on A and then needs B.

Intermediate out of order.
covering index:
contains all columns needed for a set of queries

no need to go to base data but…
random access to traverse the tree & need to sort result

Vs.

sequential access pattern but needs to access all data
**the standard solution**

1) maintain statistics,
2) optimizer chooses access path depending on estimated selectivity
the standard solution
1) maintain statistics,
2) optimizer chooses access path depending on estimated selectivity

what is wrong with that
Motivation

Execution time (sec) vs. Result selectivity (%)

- **Index Scan**
- **Full Scan**

Graph shows linear increase in execution time with increasing result selectivity for both Index Scan and Full Scan methods.
Motivation

TPC-H (SF10) 2/2

Normalized execution time

Original
Tuned

Q1, Q3, Q5, Q7, Q9, Q11, Q13, Q16, Q19, Q22

Index Scan
Full Scan
Motivation

TPCH (SF10) 2/2

Result selectivity (%)

Normalised execution time

TPC-H Query

ROBUSTNESS
can we just recompute the statistics?

![Graph showing execution time vs. result selectivity for various statistics collection methods.]

- basic stats
- per column
- for pair

- Full Scan
- Index Scan
- Optimizer decision
- Avg. statistics collection

Execution time (sec) vs. Result selectivity (%)
can we just recomput the statistics?

Computation of statistics is too expensive

If the workload is volatile with lots of updates
Then either we pay and recompute all the time
Or we have non-robust performance

... 

Or we need something else
2012, somewhere in Germany

if I keep 30 data systems researchers “trapped” in a castle for a week, we might be able to define “robust query processing” and find a few solutions
robust query processing (best definition to date by Goetz)
graceful degradation when the environment changes
Can we avoid bad access path selection (secondary index vs scan) when we have stale (or no) statistics?
\textbf{select} \text{min}(A) \textbf{from} \ R \textbf{where} B<10 \text{ and } C<80
SWITCH SCAN
while index probing
switch to scan
if cardinality > estimation

good: avoids worst case
bad: performance cliff

SMOOTH SCAN
goal avoid performance cliff
close to optimal

Design smooth scan
smooth scan
gradually morph from index scan to full scan

for each qualifying tupleID
mode1 fetch the respective page and get the value
mode2 check all values in a fetched page
mode3 fetch and check adjacent pages as well
mode3+ increase # of pages fetched
some design points

tuple cache: avoid producing the same result value
page cache: avoid reading the same page twice
result cache: produce result in indexed order
when to morph
in order to achieve a smooth behavior

**optimizer** start when selectivity > estimation

**SLA** respect an upper threshold

**selectivity** morph when selectivity increases by z

**pessimistic** morph with every new probe
The diagram compares the execution time (in seconds) of different TPC-H queries for PostgreSQL and PostgreSQL with Smooth Scan. The queries Q1 (98%), Q4 (<1%), and Q14 (<1%) show significantly lower execution times for PostgreSQL with Smooth Scan compared to PostgreSQL. Query Q6 (2%) shows a much higher execution time for PostgreSQL with Smooth Scan.

- **TPC-H Query**
  - Q1 (98%)
  - Q4 (<1%)
  - Q6 (2%)
  - Q14 (<1%)

- **Execution time (sec)**
  - 0 to 1400

- **Software**
  - PostgreSQL
  - PostgreSQL with Smooth Scan
Read: **Access Path Selection in Main-Memory Optimized Data Systems: Should I Scan or Should I Probe?**
Mike. Kester, Manos Athanassoulis, and Stratos Idreos
ACM **SIGMOD** International Conference on Management of Data, 2017

Browse: **Smooth Scan: Statistics-Oblivious Access Paths**
Renata Borovica, Stratos Idreos, Anastasia Ailamaki, Marcin Zukowski and Campbell Fraser
IEEE International Conference on Data Engineering (ICDE), 2015

Extra: **Efficient mid-query re-optimization of sub-optimal query execution plans**
Navin Kabra and David DeWitt
ACM **SIGMOD** International Conference on Management of Data, 1998
scans vs indexes

DATA SYSTEMS

prof. Stratos Idreos