EECS 262a
Advanced Topics in Computer Systems
Lecture 17

C-Store / DB Cracking
March 28th, 2016

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

- C-Store: A Column-oriented DBMS*
  Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, Stan Zdonik. Appears in *Proceedings of the ACM Conference on Very Large Databases*(VLDB), 2005
- Database Cracking+
  Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Appears in the *3rd Biennial Conference on Innovative Data Systems Research* (CIDR), January 7-10, 2007
- Wednesday: Comparison of PDBMS, CS, MR
- Thoughts?

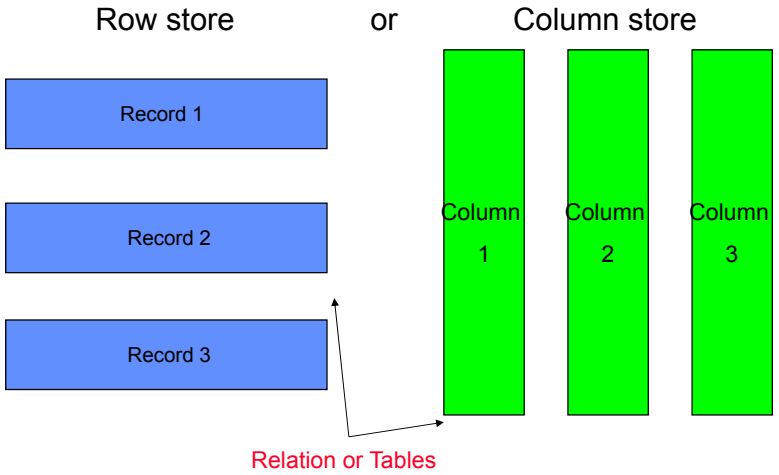
*Some slides based on slides from Jianlin Feng, School of Software, Sun Yat-Sen University
+Some slides based on slides from Stratos Idreos, CWI Amsterdam, The Netherlands

Relational Data: Logical View

Name	Age	Department	Salary
Bob	25	Math	10K
Bill	27	EECS	50K
Jill	24	Biology	80K

C-Store: A Column-oriented DBMS

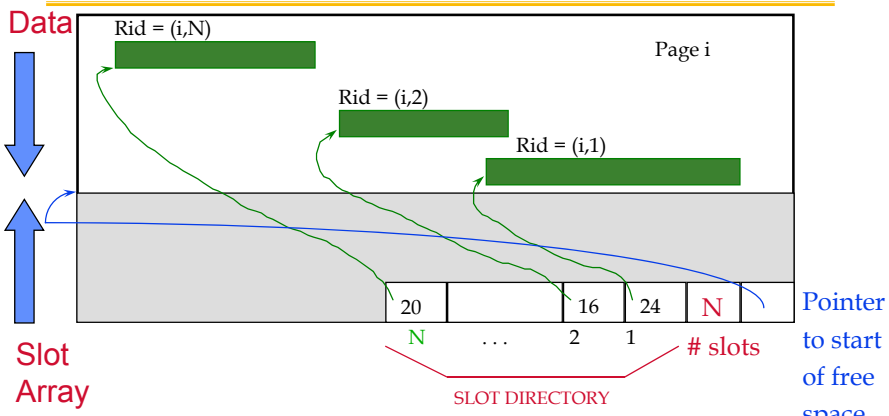
- Physical layout:



Row Stores

- On-Line Transaction Processing (OLTP)
  - ATM, POS in supermarkets
- Characteristics of OLTP applications:
  - Transactions that involve small numbers of records (or tuples)
  - Frequent updates (including queries)
  - Many users
  - Fast response times
- OLTP needs a write-optimized row store
  - Insert and delete a record in one physical write
- Easy to add new record, but might read unnecessary data (wasted memory and I/O bandwidth)

Row Store: Columns Stored Together



Record id = <page id, slot #>

Current DBMS Gold Standard

- Store columns from one record contiguously on disk
- Use B-tree indexing
- Use small (e.g. 4K) disk blocks
- Align fields on byte or word boundaries
- Conventional (row-oriented) query optimizer and executor (technology from 1979)
- Aries-style transactions

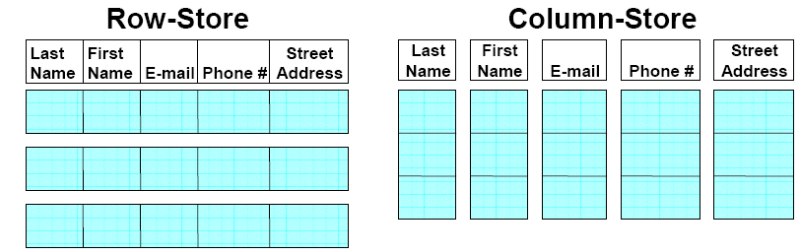
OLAP and Data Warehouses

- On-Line Analytical Processing (OLAP)
  - Flexible reporting for Business Intelligence
- Characteristics of OLAP applications
  - Transactions that involve large numbers of records
  - Frequent Ad-hoc queries and infrequent updates
  - A few decision-making users
  - Fast response times
- Data Warehouses facilitate reporting and analysis
  - Read-Mostly
- Other read-mostly applications
  - Customer Relationship Management (Siebel, Oracle)
  - Catalog Search in E-Commerce (Amazon.com, Bestbuy.com)

Column Stores

- Logical data model: Relational Model
- Key Intuition: Only read relevant columns
  - Example: Ad-hoc queries read 2 columns out of 20
- Multiple prior column store implementations
  - Sybase IQ (early '90s, bitmap index)
  - Addamark (i.e., SenSage, for Event Log data warehouse)
  - KDB (Column-stores for financial services companies)
  - MonetDB (Hyper-Pipelining Query Execution, CIDR'05)
- Only read necessary data, but might need multiple seeks

Row and Column Stores

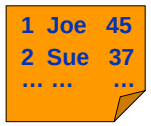


+ Easy to add a new record
– Might read in unnecessary data

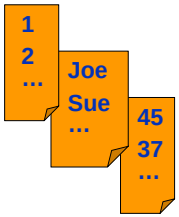
+ Only need to read in relevant data
– Tuple writes might require multiple seeks

Read-Optimized Databases

SQL Server
DB2
Oracle



Sybase IQ
MonetDB
KDB
C-Store

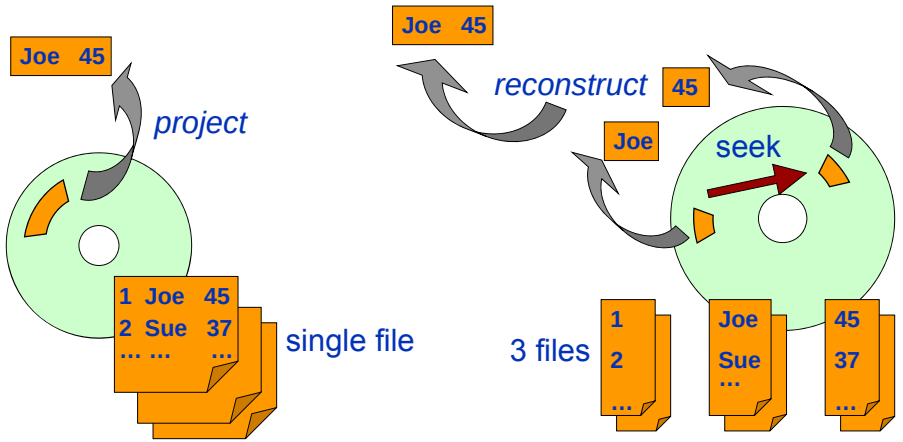


row stores **column stores**

- Effect of column-orientation on performance?
  - Read-less, seek more so depends on prefetching, query selectivity, tuple width, competing query traffic

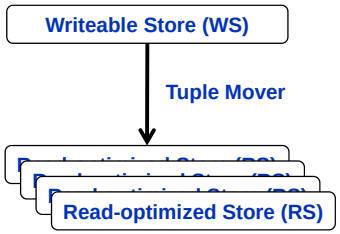
Rows versus Columns

row stores **column stores**



C-Store Technical Ideas

- Column Store with some "novel" ideas (below)
- Only materialized views on each relation (perhaps many)
- Active data compression
- Column-oriented query executor and optimizer
- Shared-nothing architecture
- Replication-based concurrency control and recovery



Architecture of Vertica C-Store

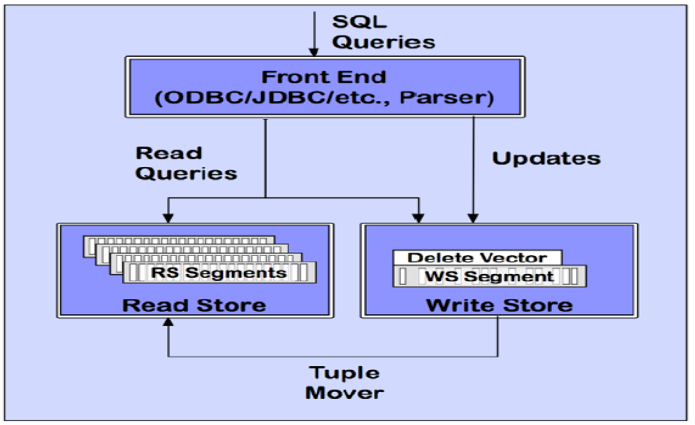


Figure 1: The Vertica Database Architecture

Basic Concepts

- A logical table is physically represented as a set of projections
- Each projection consists of a set of columns
  - Columns are stored separately, along with a common sort order defined by SORT KEY
- Each column appears in at least one projection
- A column can have different sort orders if it is stored in multiple projections

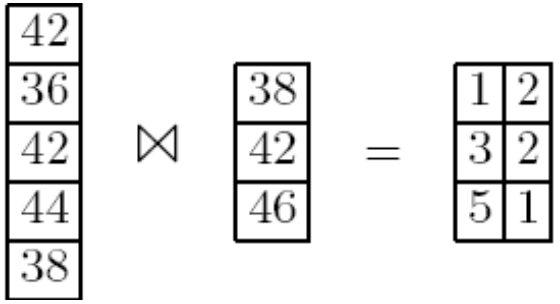
Example C-Store Projection

- LINEITEM(shipdate, quantity, retflag, suppkey | shipdate, quantity, retflag)
  - First sorted by shipdate
  - Second sorted by quantity
  - Third sorted by retflag
- Sorting increases locality of data
  - Favors compression techniques such as Run-Length Encoding (see Elephant paper)

C-Store Operators

- Selection
  - Produce bitmaps that can be efficiently combined
- Mask
  - Materialize a set of values from a column and a bitmap
- Permute
  - Reorder a column using a join index
- Projection
  - Free operation!
  - Two columns in the same order can be concatenated for free
- Join
  - Produces positions rather than values

Example: Join over Two Columns

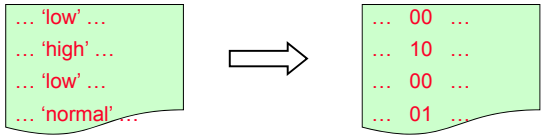


Column Store has High Compressibility

- Each attribute is stored in a separate column
  - Related values are compressible (versus values of separate attributes)
- Compression benefits
  - Reduces the data sizes
  - Improves disk (and memory) I/O performance by:
    - » reducing seek times (related data stored nearer together)
    - » reducing transfer times (less data to read/write)
    - » increasing buffer hit rate (buffer can hold larger fraction of data)

Compression Methods

- Dictionary



- Bit-pack
  - Pack several attributes inside a 4-byte word
  - Use as many bits as max-value
- Delta
  - Base value per page
  - Arithmetic differences
- No Run-Length Encoding (unlike Elephant paper)

C-Store Use of Snapshot Isolation

- Snapshot Isolation for Fast OLAP/Data Warehousing
  - Allows very fast transactions without locks
  - Can read large consistent snapshot of database
- Divide into RS and WS stores
  - Read Store is Optimized for Fast Read-Only Transactions
  - Write Store is Optimized for Transactional Updates
- Low Water Mark (LWM)
  - Represents earliest epoch at which read-only transactions can run
  - RS contains tuples added before LWM
- High Water Mark (HWM)
  - Represents latest epoch at which read-only transactions can run

Other Ideas

- K-safety: Can handle up to K-1 failures
  - Every piece of data replicated K times
  - Different projections sorted in different ways
- Join Tables
  - Construct original tuples given covering projects
  - Vertica gave up on Join Tables – too expensive, require super-projections

Evaluation?

Query	C-Store	Row Store	Column Store
Q1	0.03	6.80	2.24
Q2	0.36	1.09	0.83
Q3	4.90	93.26	29.54
Q4	2.09	722.90	22.23
Q5	0.31	116.56	0.93
Q6	8.50	652.90	32.83
Q7	2.54	265.80	33.24

- Series of 7 queries against C-Store vs two commercial DBMS
  - C-Store faster In all cases, sometimes significantly
- Why so much faster?
  - Column Representation – avoid extra reads
  - Overlapping projections – multiple orderings of column as appropriate
  - Better data compression
  - Query operators operating on compressed data

Summary

- Columns outperform rows in listed workloads
  - Reasons:
    - » Column representation avoids reads of unused attributes
    - » Query operators operate on compressed representation, mitigating storage barrier problem
    - » Avoids memory-bandwidth bottleneck in rows
- Storing overlapping projections, rather than the whole table allows storage of multiple orderings of a column as appropriate
- Better compression of data allows more orderings in the same space
- Results from other papers:
  - Prefetching is key to columns outperforming rows
  - Systems with competing traffic favor columns

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

BREAK

DB Physical Organization Problem

- Many DBMS perform well and efficiently only after being tuned by a DBA
- DBA decides:
  - *Which indexes to build?*
  - *On which data parts?*
  - *and when to build them?*

Timeline

- Sample workload
- Analyze performance
- Prepare estimated physical design
- Queries

Very complex and time consuming process!

What about:

- Dynamic, changing workloads?
- Very Large Databases?

Database Cracking

- Solve challenges of dynamic environments:
  - *Remove all tuning* and physical design steps, but still get similar performance as a fully tuned system
- How?
- Design new auto-tuning kernels
  - DBA with cracking (operators, plans, structures, etc.)

Database Cracking

- No monitoring
- No preparation
- No external tools
- No full indexes
- No human involvement
- *Continuous on-the-fly physical reorganization*
  - *Partial, incremental, adaptive indexing*
- *Designed for modern column-stores*

Cracking Example

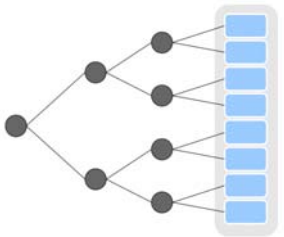
- Each query is treated as an advice on how data should be stored
  - Triggers physical re-organization of the database

Q1: select * from R
where R.A > 10
and R.A < 14

Column A
13
16
4
9
2
12
7
1
19
3
14
11
8
6

Cracking Design

- The first time a range query is posed on an attribute *A*, a cracking DBMS makes a copy of column *A*, called the *cracker column* of *A*
- A cracker column is continuously physically re-organized based on queries that need to touch attribute such as the result is in a contiguous space
- For each cracker column, there is a *cracker index*

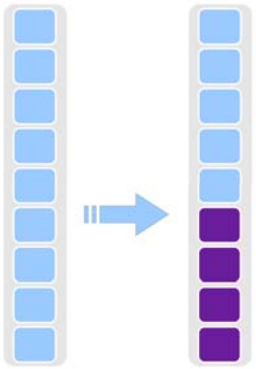


Cracking Algorithms

- Two types of cracking algorithms based on select's where clause

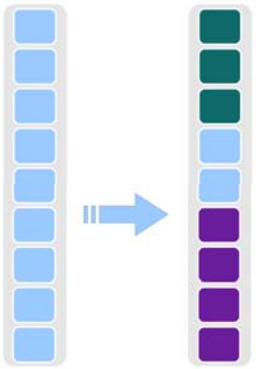
where X < #

Split a piece into two new pieces



where # < X < #

Split a piece into three new pieces

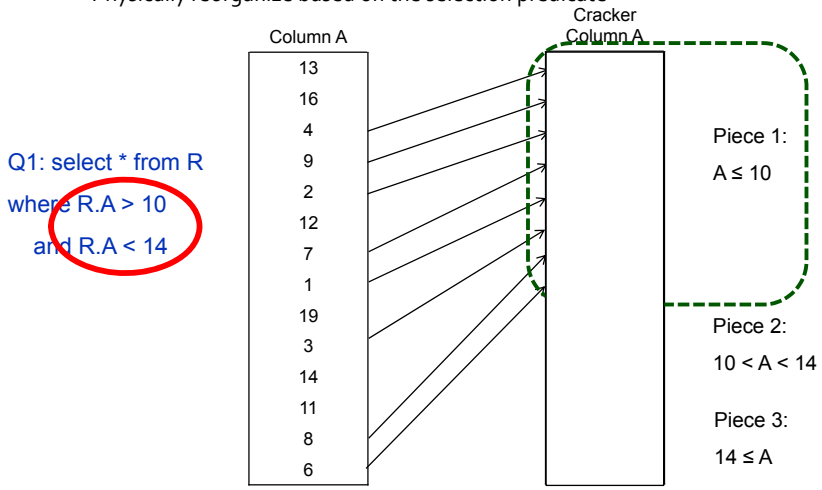


Cracker Select Operator

- Traditional select operator
  - Scans the column
  - Returns a new column that contains the qualifying values
- The cracker select operator
  - Searches the cracker index
  - Physically re-organizes the pieces found
  - Updates the cracker index
  - Returns a slice of the cracker column as the result
- More steps but faster because analyzes less data

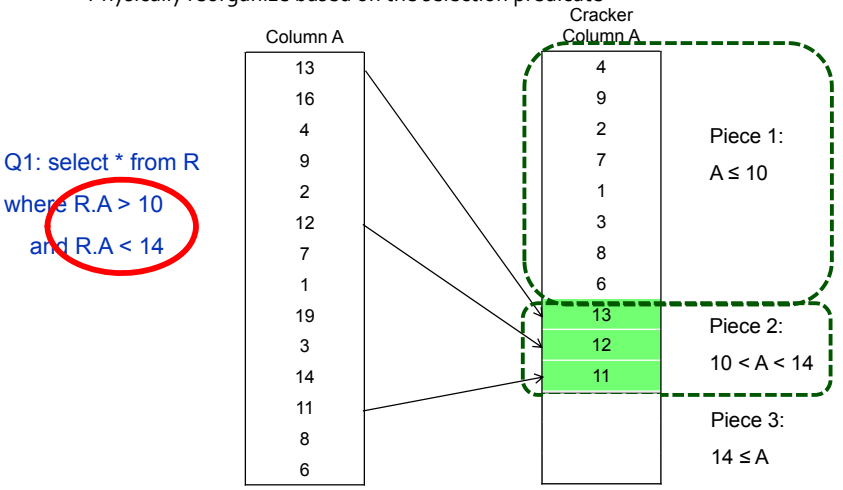
Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate



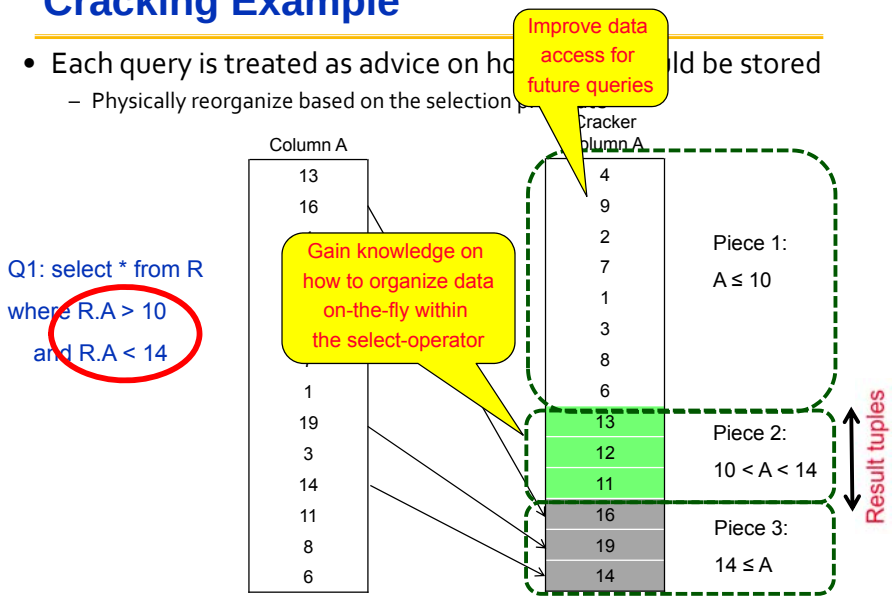
Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate



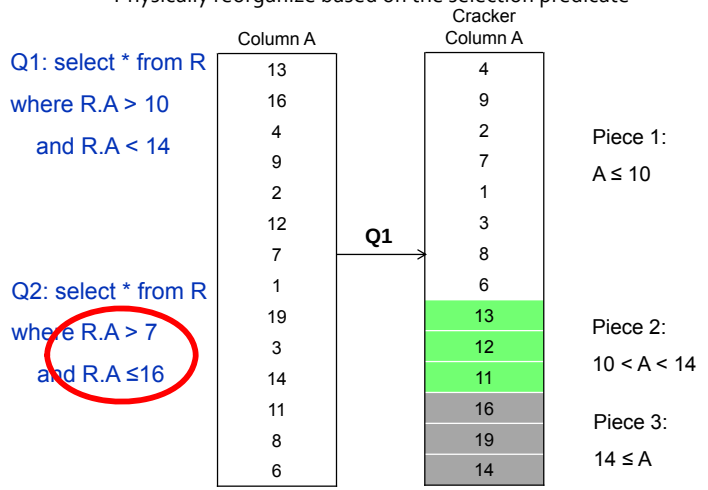
Cracking Example

- Each query is treated as advice on ho ld be stored
  - Physically reorganize based on the selection p



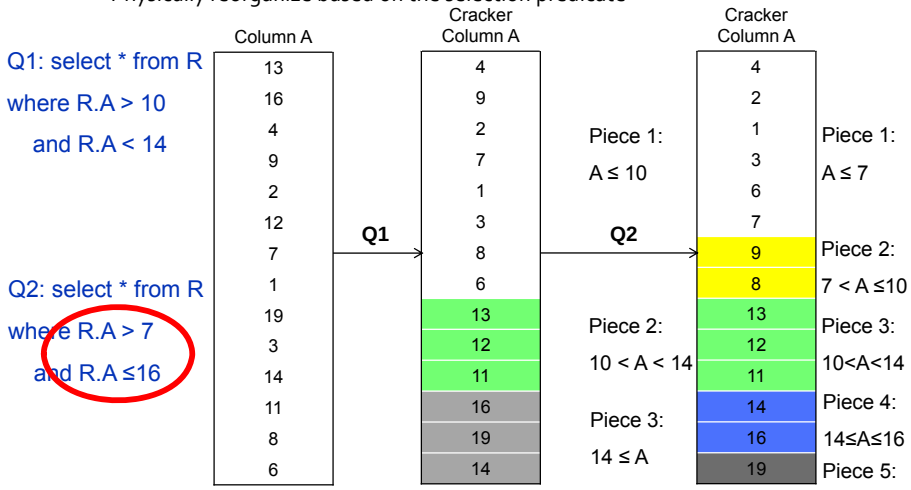
Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate



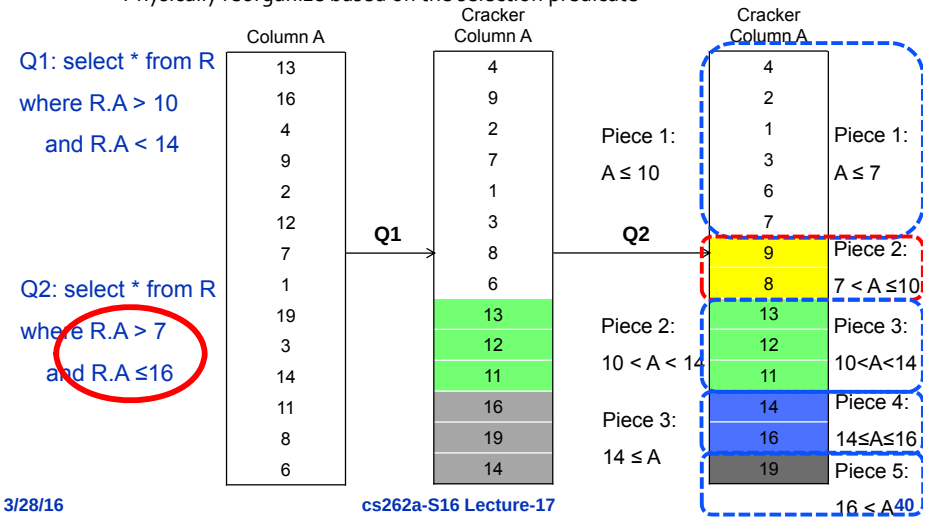
Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate



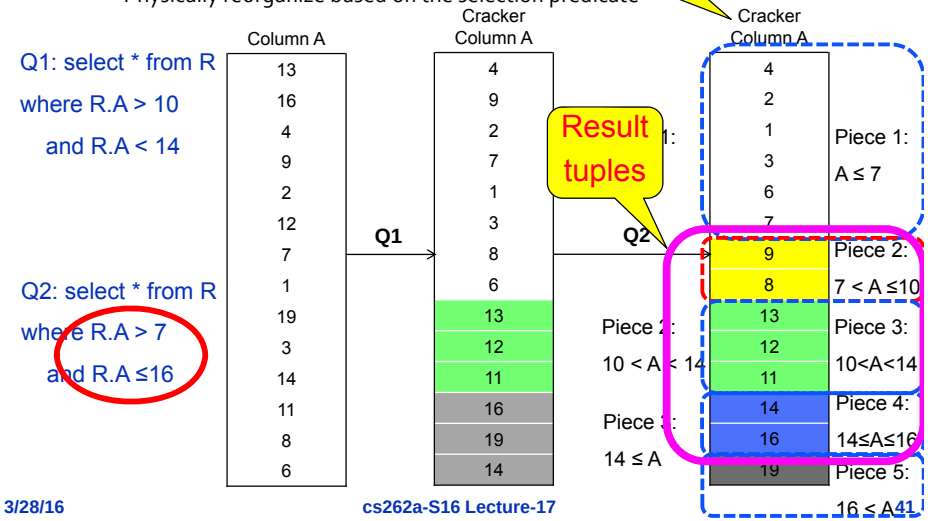
Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate

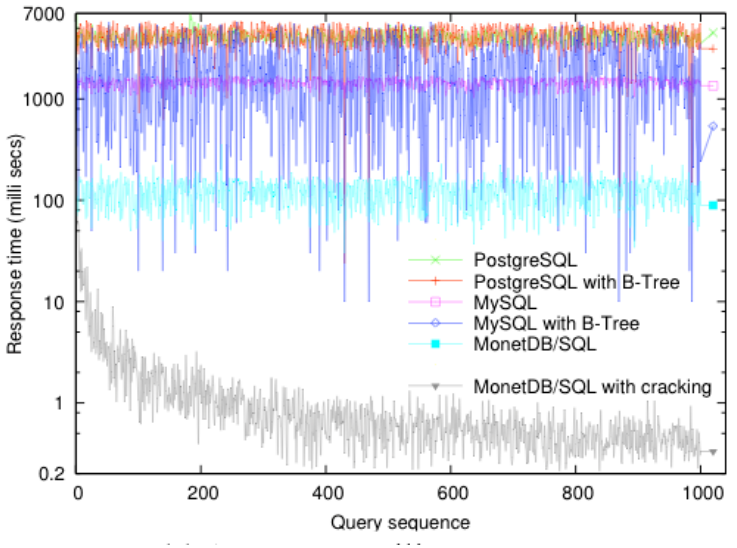


Cracking Example

- Each query is treated as advice on how data should be stored
  - Physically reorganize based on the selection predicate

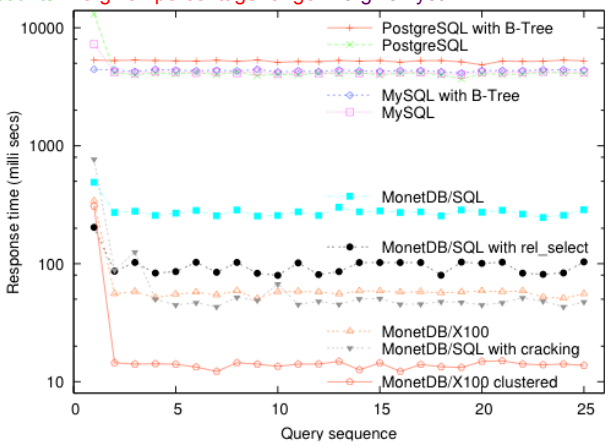


Self-Organizing Behavior (Count(*) range query)



Self-Organizing Behavior (TPC-H Query 6)

- TPC-H is an ad-hoc, decision support benchmark
  - Business oriented ad-hoc queries, concurrent data modifications
- Example:
  - Tell me the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year
- Workload:
  - Database load
  - Execution of 22 read-only queries in both single and multi-user mode
  - Execution of 2 refresh functions



Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?