Database Cracking
and the path towards auto-tuning database kernels

Stratos Idreos
Dutch National Research Center for Mathematics and Computer Science
Every two days we create as much data as much we did from dawn of humanity to 2003

[Eric Schmidt]
Every two days we create as much data as much we did from dawn of humanity to 2003

[Eric Schmidt]
Big Data V’s

- volume
- velocity
- variety
- veracity

data exploration

not always sure what we are looking for (until we find it)
databases

5 decades of research

>$100B industry, growing 10% every year

[Economist, “Data, data everywhere”]
databases

5 decades of research

>$100B industry, growing 10% every year

[Economist, “Data, data everywhere”]

SQL queries

optimize

access

store

database kernel
it is time for a **paradigm shift** in how we design databases systems
database systems great...
declarative processing, back-end to numerous apps
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
expert users - idle time - workload knowledge

but databases have become too heavy and blind!
data systems tailored for data exploration
no workload knowledge

no installation steps

data systems tailored for data exploration
no workload knowledge

no installation steps

minimize data-to-query time

data systems tailored for data exploration
3 ideas

adaptive indexing 7 years, 7 papers

adaptive loading 3 years, 3 papers

dbTouch 0.5 year, 1 paper

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou
3 ideas

- load
  - adaptive loading

- tune
  - adaptive indexing

- query
  - dbTouch

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou
tune = create proper indices offline

performance 10-100X
indexing

load tune query

tune = create proper indices offline
performance 10-100X

but it depends on workload!

which indices to build?
on which data parts?
and when to build them?
sample workload
sample workload → analyze → timeline → load → tune → query
sample workload  analyze  create indices
Complex and time consuming process
human administrators + auto-tuning tools

sample workload → analyze → create indices → query

timeline

complex and time consuming process
Big Data V’s

- Volume
- Velocity
- Variety
- Veracity

**what can go wrong?**

- Not enough space to index all data
- Not enough idle time to finish proper tuning
- By the time we finish tuning, the workload changes
- Not enough money - energy - resources
Big Data V’s

what can go wrong?

- not enough space to index all data
- not enough idle time to finish proper tuning
- by the time we finish tuning, the workload changes
- not enough money - energy - resources
database cracking
database cracking

idle time
workload knowledge
external tools
human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

idle time
workload knowledge
external tools
human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

- initialization
- querying
- indexing

- idle time
- workload knowledge
- external tools
- human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

every query is treated as an advice on how data should be stored
column-store database
a fixed-width and dense array per attribute
column-store database
a fixed-width and dense array per attribute
full indexing example

Q1: select R.A 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
full indexing example

Q1:
select R.A from R
where R.A>10
and R.A<14
full indexing example

Q1: select R.A from R
where R.A > 10
and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14
full indexing example

Q1: select R.A 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

sort

binary search

result

11 12 13 14 16 19
full indexing example

Q1: select R.A 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

sort

binary search

result

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

time + knowledge
cracking example

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

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

column A

4 9 2 7 1 3 8 6

piece 1:
A<=10

13 12 11 16 19 14
cracking example

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

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

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

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

piece1: A <= 10

piece2: 10 < A < 14

piece3: A >= 14

result: 13, 12, 11, 16, 19, 14
gain knowledge on how data is organized

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

column A

<table>
<thead>
<tr>
<th>13</th>
<th>16</th>
<th>4</th>
<th>9</th>
<th>2</th>
<th>12</th>
<th>7</th>
<th>1</th>
<th>19</th>
<th>3</th>
<th>14</th>
<th>11</th>
<th>8</th>
<th>6</th>
</tr>
</thead>
</table>

piece1: A<=10

piece2: 10<A<14

piece3: A>=14

result

4  9  2  7  1  3  8  6

13 12 11

16 19 14
gain knowledge on how data is organized

column A

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

dynamically/on-the-fly within the select-operator

piece1: A<=10

piece2: 10<A<14

piece3: A>=14
cracking example

column A

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>16</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>3</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

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

Q2: select R.A from R where R.A>7 and R.A<=16

dynamically/on-the-fly within the select-operator
cracking example

column A

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

Q2: select R.A from R where R.A>7 and R.A<=16
dynamically/on-the-fly within the select-operator
cracking example

column A

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

Q2: select R.A from R where R.A>7 and R.A<=16

dynamically/on-the-fly within the select-operator
cracking example

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

Q2: select R.A from R where R.A>7 and R.A<=16

dynamically/on-the-fly within the select-operator
cracking example

column A

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

dynamically/on-the-fly within the select-operator

Q2:
select R.A
from R
where R.A > 7
and R.A <= 16

piece1: A <= 10
piece2: 10 < A < 14
piece3: A >= 14

piece1: A <= 7
piece2: 7 < A <= 10
piece3: 10 < A < 14
piece4: 14 <= A <= 16
piece5: A > 16
cracking example

column A

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

Q2:
select R.A from R
where R.A>7
and R.A<=16

dynamically/on-the-fly within the select-operator

piece1: A<=10

piece2: 10<A<14

piece3: A>=14

piece1: A<=7

piece2: 7<A<=10

piece3: 10<A<14

piece4: 14<=A<=16

piece5: A>16
the more we crack, the more we learn

column A

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

dynamically/on-the-fly within the select-operator

Q2:
select R.A
from R
where R.A>7
and R.A<=16

piece1: A<=10

piece2: 10<A<14

piece3: A>=14

result

piece1: A<=7
piece2: 7<A<=10
piece3: 10<A<14
piece4: 14<=A<=16
piece5: A>16
select [15,55]
select [15,55]
select [15,55]
select [15,55]

10 20 30 40 50 60

select [15,55]
select [15,55]

10 20 30 40 50 60

select [15,55]
touch at most two pieces at a time

pieces become smaller and smaller

select [15,55]

10 20 30 40 50 60
implemented in **monetdb**

open-source column-store

database kernel

- optimizer
- reconstruct
- select
- join
- update
- aggr

code footprint
monetdb 2M
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

Database Cracking CIDR 2007

Response time (secs)

Query sequence (x1000)

Scan
Crack
Full Index
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

Database Cracking CIDR 2007
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement

Database Cracking CIDR 2007
continuous adaptation

set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

Database Cracking CIDR 2007

Cumulative average response time (secs) vs Query sequence

- Full Index
- Scan
- Crack

Continuous adaptation
continuous adaptation

set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

Cumulative average response time (secs)

Query sequence

Continuous adaptation

Scan
Full Index
Crack
continuous adaptation

set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

10K queries later,
Full Index still has not
amortized the initialization costs

Database Cracking CIDR 2007
traditional databases
monolithic/full indexing

workload analysis
index building
query processing

offline indexing

online indexing
traditional databases
monolithic/full indexing

workload analysis
index building
query processing

offline indexing

online indexing

database cracking
partial/adaptive/continuous indexing

adaptive indexing
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
select R.A from R where R.A>10 and R.A<14
select R.A from R where R.A > 10 and R.A < 14

select max(R.A), max(R.B), max(S.A), max(S.B) from R, S
where v1 < R.C < v2 and v3 < R.D < v4
and v5 < R.E < v6 and k1 < S.C < k2 and k3 < S.D < k4
and k5 < S.E < k6 and R.F = S.F
select R.A from R where R.A > 10 and R.A < 14

select max(R.A), max(R.B), max(S.A), max(S.B) from R, S
where v1 < R.C < v2 and v3 < R.D < v4
and v5 < R.E < v6 and k1 < S.C < k2 and k3 < S.D < k4 and k5 < S.E < k6
and R.F = S.F
cracking tangram

*base data*

as queries arrive...

**table 1**

```
A  B  C  D
```

**table 2**

```
A  B  C  D
```
cracking tangram

base data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

as queries arrive...

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>
cracking tangram

**base data**

**table 1**

A B C D

**table 2**

A B C D

**as queries arrive...**

**partial materialization**

**table 1**

A B C D

**table 2**

A B C D

**base data**

**table 1**

A B C D

**table 2**

A B C D
cracking tangram

*base data*

**table 1**

```
A   B   C   D
```

```
A   B   C   D
```

**table 2**

```
A   B   C   D
```

```
A   B   C   D
```

*as queries arrive...*

**table 1**

```
A
```

```
B
```

```
C
```

```
D
```

**table 2**

```
A
```

```
B
```

```
C
```

```
D
```

partial materialization

partial indexing
cracking tangram

**base data**

<table>
<thead>
<tr>
<th>table 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

**as queries arrive...**

<table>
<thead>
<tr>
<th>table 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

**partial materialization**  
**partial indexing**  
**continuous adaptation**
cracking tangram

base data

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
cracking tangram

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

partial materialization
partial indexing
continuous adaptation
storage adaptation
cracking tangram

**base data**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**table 1**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**table 2**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**partial materialization**

**partial indexing**

**continuous adaptation**

**storage adaptation**

**no tuple reconstruction**
cracking tangram

**base data**

**table 1**

A  B  C  D

A  B  C  D

**table 2**

A  B  C  D

A  B  C  D

**as queries arrive...**

**table 1**

A  B  C  D

A  B  C  D

**table 2**

A  B  C  D

A  B  C  D

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
cracking tangram

**base data**

**table 1**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

**table 2**

| A | B | C | D |

**as queries arrive...**

**table 1**

| A | B | C | D |

**table 2**

| A | B | C | D |

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
cracking tangram

**base data**

table 1

```
A B C D
```

table 2

```
A B C D
```

**as queries arrive...**

**table 1**

```
A
B
C
D
```

**table 2**

```
A
B
C
D
```

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
- sort in caches
cracking tangram

**base data**
- **table 1**: A, B, C, D
- **table 2**: A, B, C, D

**as queries arrive...**
- **table 1**: A, B, C, D
- **table 2**: A, B, C, D

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
- sort in caches
- crack joins
cracking tangram

base data

table 1

A B C D

as queries arrive...

A B C D

table 1

A B C

table 2

A B C

q1 q2

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
lightweight locking
cracking tangram

base data

<table>
<thead>
<tr>
<th>table 1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>table 1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

query random

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
lightweight locking
stochastic cracking
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- robustness (PVLDB12)
- algorithms (PVLDB11)
- storage restrictions (SIGMOD09)
- benchmarking (TPCTC10)

concurrency control (PVLDB12)
cracking databases

- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- concurrency control (PVLDB12)
- algorithms (PVLDB11)
- b-trees (HP Labs)
- storage restrictions (SIGMOD09)
- robustness (PVLDB12)
- benchmarking (TPCTC10)
- hadoop (Yale/Saarland)
- basics (CIDR07)
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- concurrency control (PVLDB12)
- algorithms (PVLDB11)
- robustness (PVLDB12)
- storage restrictions (SIGMOD09)
- benchmarking (TPCTC10)
- b-trees (HP Labs)
- hadoop (Yale/Saarland)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- concurrency control (PVLDB12)
- hadoop (Yale/Saarland)
- >1 columns (SIGMOD09)
- robustness (PVLDB12)
- b-trees (HP Labs)
- algorithms (PVLDB11)
- storage restrictions (SIGMOD09)
- benchmarking (TPCTC10)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- robustness (PVLDB12)
- concurrency control (PVLDB12)
- b-trees (HP Labs)
- algorithms (PVLDB11)
- storage-restrictions (SIGMOD09)
- benchmarking (TPCTC10)
- hadoop (Yale/Saarland)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
position alignment

lookup

\[ A(i) = A + i \times \text{width}(A) \]
**Positional alignment**

**lookup**

A(i) = A + i * width(A)

**query**

max(B) where A<10
A(i) = A + i \times \text{width}(A)

\text{tuple 1} \quad \text{tuple 2} \quad \text{tuple 3} \quad \ldots

\text{query} \quad \max(B) \text{ where } \mathbf{A} < 10

max(B) where \mathbf{A} < 10
**positional alignment**

**lookup**

\[ A(i) = A + i \times \text{width}(A) \]

**query**

\[ \text{max}(B) \text{ where } A < 10 \]
**position alignment**

**lookup**

$$A(i) = A + i \cdot \text{width}(A)$$

**query**

$$\text{max}(B) \text{ where } A < 10$$

**positional join**

```
A(i) = A + i * width(A)

max(B) where A < 10
```

---

**Sideways Cracking, SIGMOD 09**
positional alignment

lookup

A(i) = A + i \times \text{width}(A)

query

\text{max}(B) \text{ where } A<10

positional join

**A**

\[
\begin{align*}
a1 \\
a2 \\
a3 \\
a4 \\
a5 \\
a6 \\
a7 \\
a8 \\
a9 \\
a10 \\
\end{align*}
\]

**B**

\[
\begin{align*}
a1 \\
a2 \\
a3 \\
a4 \\
a5 \\
a6 \\
a7 \\
a8 \\
a9 \\
a10 \\
\end{align*}
\]

**pos**

\[
\begin{align*}
1 \\
4 \\
6 \\
9 \\
\end{align*}
\]

**tuple 1**

**tuple 2**

**tuple 3**

...
positional alignment

A(i) = A + i * width(A)

lookup
tuple 1
tuple 2
tuple 3
...

query
max(B) where A < 10

positional join

A<10

max(B) with cracking

pos

with cracking

b1
b2
b3
b4
b5
b6
b7
b8
b9
b10

b1
b2
b3
b4
b5
b6
b7
b8
b9
b10

b1
b4
b6
b9
sideways cracking
sideways cracking
sideways cracking
sideways cracking

query
sideways cracking

query
**sideways cracking**

query

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sideways cracking

query

A

B

C

D
sideways cracking

query

A  B  C  D
sideways cracking

query
sideways cracking

A

B

C

D

query
sideways cracking

query
sideways cracking

query
replace tuple reconstruction with cracking

log crack actions and replay to align columns dynamically

query
TPC-H

MonetDB  -  Sel. Crack  -  MySQL  -  Presorted
Presorted  -  Sid. Crack  -  Presorted

Response time (milli secs)

Query sequence

TPC-H Query 15
TPC-H

MonetDB  Presorted  Sel. Crack  Sid. Crack  MySQL  Presorted

Preparation cost
~3 hours

Fully tuned MonetDB

Response time (milli secs)

Query sequence

TPC-H Query 15

Selection cracking
TPC-H

Response time (milli secs)

0 70 100 150 200 250 300 330 1000 3000 10000

Query sequence

MonetDB  Presorted
Sel. Crack  Sid. Crack
MySQL  Presorted

Plain MonetDB
Selection cracking
MonetDB with sideways cracking

Fully tuned MonetDB
Preparation cost ~3 hours
TPC-H

MonetDB  -  Sel. Crack  -  MySQL
Presorted  -  Sid. Crack  -  Presorted

Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H Query 15

Response time (milli secs)

Query sequence

Plain MonetDB
Selection cracking
MonetDB with sideways cracking
TPC-H

- MonetDB
- Sel. Crack
- Sid. Crack
- MySQL
- Presorted

Preparation cost
~3 hours

Response time (milli secs)

Query sequence

TPC-H Query 15

- Fully tuned MonetDB
- Selection cracking
- MonetDB with sideways cracking
TPC-H

MonetDB  Presorted  MySQL  Presorted

Selection cracking
MonetDB with sideways cracking

Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H Query 15

Response time (milli secs)
Query sequence

764
420

MonetDB

Presorted

Selection cracking
MonetDB with sideways cracking

TPC-H
TPC-H

reducing data-to-query time

- Fully tuned MonetDB
  - Preparation cost ~3 hours

- TPC-H Query 15

- Response time (milli secs) vs Query sequence

- Selection cracking
  - MonetDB with sideways cracking

- TPC-H with plain MonetDB
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- concurrency control (PVLDB12)
- robustness (PVLDB12)
- algorithms (PVLDB11)
- storage restrictions (SIGMOD09)
- benchmarking (TPCTC10)
- hadoop (Yale/Saarland)
- b-trees (HP Labs)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
stochastic cracking

robustness: maintain performance levels or have graceful degradation when input or status changes
adaptive indexing
Response time: X

adaptive indexing
Response time: X

Q2  Q1  Q5  Q4  Q3

adaptive indexing
Response time: X  

Response time: 1000X
column with 100 unique integers [1,100]
bad query pattern

column with 100 unique integers [1,100]
bad query pattern

column with 100 unique integers [1,100]
bad query pattern

column with 100 unique integers [1,100]
bad query pattern

$q_1 < 2$
$q_2 < 3$

column with 100 unique integers $[1,100]$
bad query pattern

$q_1 < 2$
$q_2 < 3$

column with 100 unique integers $[1,100]$
bad query pattern

<2 <3 <4

q1 q2 q3

column with 100 unique integers [1,100]
bad query pattern

<2  <3  <4

q1  q2  q3

column with 100 unique integers [1,100]

N  N-1  N-2
bad query pattern

<2  <3  <4
q1  q2  q3

column with 100 unique integers [1,100]

blindly adapting to queries is not always a good idea
query driven + stochastic
query driven + stochastic

to be cracked
query driven + stochastic

$q$  random

to be cracked
query driven + stochastic

progressive cracking
query driven + stochastic

progressive cracking

$q \mid: <v_1$

to be cracked
query driven + stochastic

progressive cracking

$q \leq \langle v_1

\text{crack + filter } \langle v_1

\text{to be cracked}$
query driven + stochastic

progressive cracking

\( q \): \(< v_1 \)

\( \text{random} \)

\( \text{random} \)

\( \text{swap} \)

\( \text{to be cracked} \)

\( \text{crack} + \text{filter} \): \(< v_1 \)
query driven + stochastic

progressive cracking

$q \Rightarrow v_1$

random

swap

scan + filter $\leq v_1$

crack + filter $\leq v_1$
query driven + stochastic

progressive cracking

q1: <v1
q2: <v2

to be cracked
query driven + stochastic

progressive cracking

q1: \(<v1\>
q2: \(<v2\>

random

scan + filter \(<v2\>

to be cracked
query driven + stochastic

progressive cracking

q1: <v1
q2: <v2

swap

random

crack + filter <v2

scan + filter <v2

to be cracked
query driven + stochastic

to be cracked

progressive cracking

q1: <v1
q2: <v2

swap
crack + filter <v2

scan + filter <v2

Stochastic Cracking, PVLDB 12
cracking on Skyserver (4TB)
(Sloan Digital Sky Survey, www.sdss.org)

cracking answers 160,000 queries while full indexing is still halfway creating one index
cracking on Skyserver (4TB) (Sloan Digital Sky Survey, www.sdss.org)

reducing data-to-query time

cracking answers 160,000 queries while full indexing is still halfway creating one index
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- concurrency control (PVLDB12)
- hadoop (Yale/Saarland)
- >1 columns (SIGMOD09)
- robustness (PVLDB12)
- b-trees (HP Labs)
- storage restrictions (SIGMOD09)
- algorithms (PVLDB11)
- benchmarking (TPCTC10)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
loading
copy data inside the database

database now has full control
load | tune | query

copy data inside the database

database now has full control

slow process...not all data might be needed all the time
database vs. unix tools

1 file, 4 attributes, 1 billion tuples
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

break down db cost

- Loading: 7%
- Query Processing: 93%

single query cost (secs)

- DB
- Awk

0 550 1,100 1,650 2,200
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

loading is a major bottleneck
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

break down db cost
- Loading: 93%
- Query Processing: 7%

... but writing/maintaining scripts is hard too
adaptive loading

load/touch only what is needed and only when it is needed
but raw data access is expensive

tokenizing - parsing - no indexing - no statistics

challenge: fast raw data access
Adaptive Loading, CIDR 11

query plan
Adaptive Loading, CIDR 11

query plan

scan
Adaptive Loading, CIDR 11

Query plan:
- Scan
- DB
Adaptive Loading, CIDR 11

- query plan
- scan

loading
access raw data adaptively on-the-fly
access raw data adaptively on-the-fly
access raw data adaptively on-the-fly

query plan

scan

files

cache

selective parsing
file indexing
file splitting
online statistics

loading
MySQL CSV Engine
MySQL DBMS
DBMS X
PostgreSQL
PostgresRaw PM + C

Execution Time (sec)

Q20  Q19  Q18
Q17  Q16  Q15
Q14  Q13  Q12
Q11  Q10  Q9
Q8   Q7   Q6
Q5   Q4   Q3
Q2   Q1   Load

Load: NoDB, SIGMOD 2012
reducing data-to-query time

Execution Time (sec)

MySQL

DBMS X

PostgreSQL

PostgresRaw PM + C
towards auto-tuning data kernels
so what's next?

adaptive (load-store-execute) cracking (+ AI, + OS, + ML)

compression

multi-core cracking

row-store cracking

multidimensional cracking

aggregations

...and many more...
interactive data systems
querying

load tune query
querying

load tune query

SQL interface
querying

load  tune  query

SQL interface
correct and complete answers
querying

complex and slow - not fit for exploration

SQL interface

correct and complete answers
just touch the data you need
just touch the data you need

this is not about query building
it is about query processing
column1

64000

data
HCI + databases
what does this mean for db kernels?
what does this mean for db kernels?
select R.a from R

what does this mean for db kernels?

process only what you touch
explore: touch, observe and react

the system does not have control of the data flow
the user dictates which is the next tuple

hierarchies of samples
incremental and adaptive operators
adaptive indexing - adaptive loading
an exploration tool
get a quick feeling about your data
focus on interesting areas

$HCI + databases$

come play with the dbTouch demo on iPad!
3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

Stratos Idreos
CWI, Amsterdam

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change

[Darwin, Megginson]
3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

Stratos Idreos
CWI, Amsterdam

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change

[Darwin, Megginson]
3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change
[Darwin, Megginson]

Thank you!
Stratos Idreos
CWI, Amsterdam