Overview of Data Exploration Techniques

Stratos Idreos, Olga Papaemmanouil, Surajit Chaudhuri
data exploration

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

data has always been big

---

volume  velocity  variety  veracity
content structure

user interaction

middleware

database kernel
Part 3*

*for Part 1 and 2 please look at the websites of the tutorial co-authors
too many preparation options lead to complex installation

schema  storage  load  indexes  query

timeline

expert users - idle time - workload knowledge
users/applications
declarative interface
ask what you want

db system

DBA
need to choose the proper system
how can we prepare if we do not know what we are up against? (loading, indexing, storage, …)
how can we prepare if we do not know what we are up against? (loading, indexing, storage, …)

data systems kernels tailored for data exploration

no preparation - easy to use - fast
tune = create proper indices offline
performance 10-100X
tune = create proper indices offline
performance 10-100X

but it depends on the workload!
which indices to build?
on which data parts?
and when to build them?
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

**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
database cracking
database cracking

- idle time
- workload
- 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
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
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
Q1: select R.A from R where R.A > 10 and R.A < 14

![Diagram showing column A with values and sorting process](attachment:image.png)

**Sort**: Values are sorted in ascending order.

**Binary Search**: The values are searched in a sorted list to find the range of R.A values that satisfy the condition.

**Result**: The values 11, 12, 13, 14, 15, 16, 19 are the output of the query.
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
Q1: 
select R.A
from R
where R.A>10
and R.A<14

piece1: 
A<=10
Q1: select R.A from R where R.A > 10 and R.A < 14

piece1: $A \leq 10$

piece2: $10 < A < 14$
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

![Diagram showing column A with values and split into three pieces based on inequalities A <= 10, 10 < A < 14, and A >= 14.]

Database Cracking CIDR 2007
Q1:
select R.A from R where R.A > 10 and R.A < 14

result

piece1: A <= 10

piece2: 10 < A < 14

piece3: A >= 14

Database Cracking CIDR 2007

gain knowledge on how data is organized
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

result
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

Database Cracking CIDR 2007
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
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
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
### 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
**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
the more we crack, the more we learn

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

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

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

Database Cracking CIDR 2007
select [15,55]
select [15,55]
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]
set-up

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

almost no initialization overhead
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

Response time (secs)
Query sequence (x1000)
set-up

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

Cumulative average response time (secs)

Continuous adaptation

Query sequence

Full Index
Scan
Crack

Database Cracking CIDR 2007
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
<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
</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

Table 1

Updates

Joins

Concurrency control
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- storage-restrictions (SIGMOD09)
- robustness (PVLDB12)
- algorithms (PVLDB11)
- time-series (SIGMOD14)
- adaptive storage (SIGMOD14)
- concurrency control (PVLDB12)
- multi-cores (SIGMOD15)
- benchmarking (TPCTC10)
- hadoop (Yale/Saarland)
- b-trees (HP Labs)
cracking tanagram

<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>

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>

base data

**table 1**

**table 2**
cracking tangram

base data

table 1

A B C D

as queries arrive...

table 1

A B C D

A B C D

table 2

A B C D

table 2

A B C D
cracking tangram

as queries arrive...

partial materialization
cracking tangram

base data
table 1

as queries arrive...
table 1

partial materialization
partial indexing
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation

continuous adaptation

continuous adaptation
cracking tangram

base data

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

as queries arrive...

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

partial materialization

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

partial indexing

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

continuous adaptation

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

storage adaptation

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

table 1

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

table 2

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

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

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

As queries arrive...

Table 1

Table 2

Base data

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>
</table>

as queries arrive...

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

**table 1**

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

partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

**table 2**

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

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

**table 2**

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

as queries arrive...

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

as queries arrive...

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

as queries arrive...

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

**base data**

**table 1**

**table 2**

**partial materialization**

**partial indexing**

**continuous adaptation**

**storage adaptation**

**no tuple reconstruction**

**adaptive alignment**

**sort in caches**
base data
table 1

as queries arrive...
table 1

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

table 2

table 2

cracking tangram
cracking tangram

base data

A
B
C
D

as queries arrive...

A
B
C
D

table 1

table 1

A
B
C
D

A
B
C
D

table 1

table 1

A
B
C
D

A
B
C
D

table 2

partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

lightweight locking

crack joins
cracking tangram

as queries arrive...

base data

table 1

A B C D

A B C D

A B C D

A B C D

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

Adaptive loading

Adaptive storage

Databases

Vision, declarative and flexible systems

Sampling

DbTouch
loading

load  |  tune  |  query

copy data inside the database
database now has full control

slow process...
not all data might be needed all the time
1 file, 4 attributes, 1 billion tuples
1 file, 4 attributes, 1 billion tuples

break down db cost
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

break down db cost

loading is a major bottleneck
1 file, 4 attributes, 1 billion tuples

break down db cost

loading is a major bottleneck

but writing/maintaining scripts does not scale
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
query plan
query plan
scan
query plan

scan

db
query plan

scan

loading
access raw data adaptively on-the-fly
Adaptive Loading, CIDR 11

- query plan
  - scan
  - files
  - cache

access raw data adaptively on-the-fly
selective parsing
file indexing
file splitting
online statistics

query plan

scan

files cache

access raw data
adaptively on-the-fly

loading

Adaptive Loading, CIDR 11
NoDB, SIGMOD 2012
reducing data-to-query time
adaptive indexing

adaptive loading

adaptive storage

dbTouch

vision declarative and flexible systems

sampling
row-store

A B C D

column-store

A B C D
1. INTRODUCTION

Modern state-of-the-art database systems are designed around a single data storage layout. This is a fixed decision that drives the whole design of the architecture of a database system. For example, traditional row-store systems store data one row at a time [20] while modern column-store systems store data one column at a time [1]. However, none of those choices is a universally good solution; different workloads require different storage layouts and data access methods in order to achieve good performance.

Database systems vendors provide different storage engines under the same software suite to efficiently support workloads, H.2.4

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Figure 1 illustrates an example of how even a well-tuned high-performance DBMS cannot efficiently cope with varying workloads. In this example, we test two state-of-the-art commercial systems, a row-store DBMS (DBMS-R) and a column-store DBMS (DBMS-C). We report the time needed to run a single analytical select-project-aggregate query in a modern machine. Figure 1 shows that none of those two state-of-the-art systems is a universally good solution; for different classes of queries (in this case depending on the number of attributes accessed), a different system performs best.

The way data is stored out requiring any tuning or workload knowledge.

Figure 1: Inability of state-of-the-art database systems to maintain optimal behavior across different workload patterns.

<table>
<thead>
<tr>
<th>Attributes Accessed (%)</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

DBMS-R

DBMS-C

DBMS-C

DBMS-R

no fixed optimal solution
rows & columns

row-store
A B C D

column-store
A B C D

hybrid-store
A B C D
which layout is best?
which layout is best?
which layout is best?
which layout is best?

<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... too many combinations to maintain in parallel
query cost

\[ q(L) = \sum_{i=1}^{\mid L \mid} \max(cost_i^{IO}, cost_i^{CPU}) \]

for a given query we can know which layout is best
the one that will cause the fewer cache misses
if we know all queries up front we can choose the layouts

adaptive storage:
continuously adapt layouts based on incoming queries
but computing all possible combinations is expensive…

query

select A+B+C+D from R where A<10 and E>10

1. deal only with attributes referenced in queries
2. handle select clause separately from where clause
3. start from pure column-store and build up
4. stop when no improvement possible

H20, SIGMOD 14
adaptive indexing

adaptive loading

adaptive storage

dbTouch

vision declarative and flexible systems

sampling
querying

load  tune  query

SQL interface

correct and complete answers

dbTouch, CIDR 2013
querying

complex and slow - not fit for exploration

SQL interface
correct and complete answers

dbTouch, CIDR 2013
just touch the data you need
just touch the data you need

this is not about query building
it is about query processing

dbTouch, CIDR 2013
what does this mean for db kernels?
select R.a from R

what does this mean for db kernels?
What does this mean for db kernels?

select R.a from R
from touch to query processing

touch location (x)

size(width)

row ID = (tuples*x)/size

storage
select avg(R.c) 
where R.a=S.b and S.b<20
select avg(R.c) 
where R.a=S.b and S.b<20
select avg(R.c)
where R.a=S.b and S.b<20

avg(R.c)

R.a=S.b

σ(<20)
visual objects

samples hierarchy
visual objects

samples hierarchy

base data

initial sample

dbTouch, CIDR 2013
visual objects

samples hierarchy

create incrementally and on demand

base data

initial sample

dbTouch, CIDR 2013
adaptive indexing

adaptive loading

adaptive storage

dbTouch

vision declarative and flexible systems

sampling
sampling outside the engine

application

sampling logic/query rewrite

Kernel

Aqua, VLDB 1999
BlinkDB, Eurosys 2013
sampling with SciBORG

queries → Kernel → maintain hierarchy of biased samples

data →
sampling with SciBORG

continuously reorganized based on the workload

impression 1
impression 2
impression 3
...
adaptive indexing

adaptive loading

adaptive storage

dbTouch

sampling

vision declarative and flexible systems
building systems declaratively

vision: being able to define system components in a higher level language without significant performance penalty

RodentStore, CIDR 2009
data systems today allow us to answer queries fast

data systems for exploration should allow us to find fast which queries to ask

+ approximate processing techniques
data systems today allow us to answer queries fast

data systems for exploration should allow us to find fast which queries to ask

+ approximate processing techniques

thank you!