class 4

basic db architectures & layouts

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
design

logical design

physical design

system design

next up: db architectures 101
is it “good” to have modules
is it “good” to have modules

SAME THING IS HAPPENING NOW WITH OPEN SOURCE STACKS

is it “good” to have modules
design/implement numerous possible algorithms + data representations

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

query plan

algorithms/operators

data

data

data
PROJECT MILESTONE 1
TODAY FIRST EXAMPLE:
FROM DATA TO PLANS & ANSWERS
```
select name
from student
where GPA > 3.0
```
```sql
select name
from student
where GPA > 3.0
```
select name from student where GPA>3.0

**Logical Plan**

```
student(id, name, GPA, address, class, ...)
```

```
select GPA>3.0
```
\begin{verbatim}
select name 
from student 
where GPA > 3.0
\end{verbatim}

physical plans
select avg(GPA) from student where class=2017
**select** avg(GPA) 
**from** student 
**where** class=2017
give me all students enrolled in cs165

```sql
select student.name
from student, enrolled, course
where course.name="cs165"
and enrolled.courseld=course.id
and student.id=enrolled.studentld
```
select student.name
from student, enrolled, course
where course.name="cs165"
and enrolled.courseId=course.id
and student.id=enrolled.studentId
```sql
select student.name
from student, enrolled, course
where course.name = 'cs165'
and enrolled.courseId = course.id
and student.id = enrolled.studentId
```

```
select course.name = 'cs165'
```

```
join
student.id = enrolled.studentId
```

```
join
enrolled.courseId = course.id
```

```
project
student.name
```

```
pushing selects down
```
\textbf{select} \ min(A) \ \textbf{from} \ R \ \textbf{where} \ B<10 \ \text{and} \ C<80

\textbf{internal language}

\textbf{logical plan}

\textbf{optimizer}

\textbf{rules/cost model/statistics}

\textbf{physical plan}

\textbf{execution}
\textbf{select} \ \text{min}(A) \ \textbf{from} \ R \ \textbf{where} \ B < 10 \ \text{and} \ C < 80
can DBAs make wrong decisions?

can optimizers make wrong decisions?

db kernel

- tuning
- optimizer
- execution
- storage
memory hierarchy

data layouts

column-stores basics
A system where a database runs includes multiple levels of memory hierarchy:

- **CPU - CPU - CPU - CPU**
- **CPU registers**
- **Caches**
- **Memory**
- **Disk - Disk - Disk - Disk - Disk**

Adding **flash** and **non-volatile memory** to the system enhances performance and reliability.
Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD  Edgar F. Codd Innovations Award

registers
my head
~0

2x
on chip cache
this room
1 min

10x
on board cache
this building
10 min

100x
memory
New York
1.5 hours

100Kx
disk
Pluto
2 years
Registers on chip cache

On board cache

Memory

Disk

CPU

Memory wall

Cache miss: looking for something which is not in the cache

Memory miss: looking for something which is not in memory

SRAM ~1ns

DRAM ~10ns

~100ns
Don’t Miss!

access only what you need

design of storage/access methods/algorithms should minimize: data & instruction misses
page-based access

need to only read $x$… but have to read all of page 1

data value $x$

page 1, page 2, page 3, …

data move

cpu

registers

on chip cache

on board cache

memory

disk
query $x < 5$

(size=120 bytes)
memory level $N$

memory level $N-1$

\[
\begin{array}{cccccc}
5 & 10 & 6 & 4 & 12 & \\
2 & 8 & 9 & 7 & 6 & \\
7 & 11 & 3 & 9 & 6 & \\
\end{array}
\]

page size: 5x8 bytes
query $x < 5$

scan

(size=120 bytes)
memory level N

5 10 6 4 12

memory level N-1

5 10 6 4 12
2 8 9 7 6
7 11 3 9 6
...

page size: 5x8 bytes
query $x<5$

Scan

(size=120 bytes)
memory level N

5 10 6 4 12

4

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

... page size: 5x8 bytes
query $x < 5$

Scan

memory level $N$

(size=120 bytes)

5 10 6 4 12

4

memory level $N-1$

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

page size: 5x8 bytes
query \( x < 5 \)

```
5 10 6 4 12
2 8 9 7 6
7 11 3 9 6
```

Memory level N

(size=120 bytes)

Memory level N-1

Page size: 5x8 bytes

40 bytes
query $x < 5$

Scan from memory level $N$

Size = 120 bytes

Scan from memory level $N-1$

Page size: 5x8 bytes

40 bytes
query \( x < 5 \)

memory level N

(size=120 bytes)

\[
\begin{array}{cccccc}
5 & 10 & 6 & 4 & 12 \\
\end{array}
\]

scan

\[
\begin{array}{cccccc}
2 & 8 & 9 & 7 & 6 \\
4 & 2 \\
\end{array}
\]

memory level N-1

\[
\begin{array}{cccccc}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 \\
7 & 11 & 3 & 9 & 6 \\
\ldots \\
\end{array}
\]

page size: 5x8 bytes
query $x < 5$

Scan

(memory level N)

(size = 120 bytes)

7 11 3 9 6 2 8 9 7 6 4 2

(memory level N-1)

5 1 0 6 4 1 2 2 8 9 7 6 7 1 1 3 9 6 ...

Page size: 5x8 bytes

80 bytes
query $x < 5$

memory level $N$

(size=120 bytes)

scan

7 11 3 9 6 2 8 9 7 6 4 2 3

memory level $N-1$

5 10 6 4 12 2 8 9 7 6 7 11 3 9 6 ...

page size: 5x8 bytes
query $x < 5$

(size=120 bytes)
memory level N

\[
\begin{array}{cccccc}
7 & 1 & 1 & 3 & 9 & 6 \\
2 & 8 & 9 & 7 & 6 & 4 \\
7 & 1 & 1 & 3 & 9 & 6 \\
\end{array}
\]

memory level N-1

\[
\begin{array}{cccccc}
5 & 1 & 0 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 & 7 \\
7 & 1 & 1 & 3 & 9 & 6 \\
\end{array}
\]

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

(size=120 bytes)
memory level N

memory level N-1

\[
\begin{align*}
\text{page size: } 5 \times 8 \text{ bytes}
\end{align*}
\]
an oracle gives us the positions

query \( x < 5 \)

\[ \begin{array}{ccccccc}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 \\
7 & 11 & 3 & 9 & 6 \\
\end{array} \]

(size=120 bytes)

memory level N

memory level N-1

\[ \begin{array}{ccccccc}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 \\
7 & 11 & 3 & 9 & 6 \\
\end{array} \] ...

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

\( \text{oracle} \)

(size=120 bytes)

memory level $N$

\[ 5 \ 10 \ 6 \ 4 \ 12 \quad 4 \]

memory level $N-1$

\[ 5 \ 10 \ 6 \ 4 \ 12 \quad 2 \ 8 \ 9 \ 7 \ 6 \quad 7 \ 11 \ 3 \ 9 \ 6 \quad \ldots \]

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

memory level \( N \)

(size=120 bytes)

memory level \( N-1 \)

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

oracle

5 10 6 4 12

oracle

2 8 9 7 6

4

(size=120 bytes)
memory level $N$

memory level $N-1$

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

memory level N

(size=120 bytes)

5 10 6 4 12

oracle

2 8 9 7 6

oracle

4 2

memory level N-1

5 10 6 4 12

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

(size=120 bytes)

memory level N

5 10 6 4 12

oracle

2 8 9 7 6

oracle

4 2

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

memory level N

(size=120 bytes)

oracle

\[
\begin{bmatrix}
7 & 1 & 1 & 3 & 9 & 6 \\
2 & 8 & 9 & 7 & 6 & \text{4} & \text{2}
\end{bmatrix}
\]

memory level N-1

\[
\begin{bmatrix}
5 & 1 & 0 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 & 7 & 1 & 1 & 3 & 9 & 6 & \text{...}
\end{bmatrix}
\]

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

(memory level N)

\begin{align*}
\text{oracle} & \quad \begin{array}{c}
7 \ 11 \ 3 \ 9 \ 6 \\
2 \ 8 \ 9 \ 7 \ 6 \\
4 \ 2 \ 3
\end{array} \\
\text{(size=120 bytes)}
\end{align*}

(memory level N-1)

\begin{align*}
\begin{array}{c}
5 \ 10 \ 6 \ 4 \ 12 \\
2 \ 8 \ 9 \ 7 \ 6 \\
7 \ 11 \ 3 \ 9 \ 6
\end{array} & \quad \ldots \\
\text{page size: 5x8 bytes}
\end{align*}
an oracle gives us the positions

query $x < 5$

memory level $N$

(size=120 bytes)

oracle

7 11 3 9 6  2 8 9 7 6  4 2 3

memory level $N-1$

5 10 6 4 12  2 8 9 7 6  7 11 3 9 6 ...

page size: 5x8 bytes
scan=120bytes vs oracle=120bytes
(and there is no such thing as an Oracle so Oracle is not for free…)

when does it make sense to have an oracle
**sequential access:**
read one block; consume it completely; discard it; read **next**

**in parallel/prefetching**

**what is next?**

Hardware/software can better predict/buffer sequential pages to be read
random access:
read one block; consume it partially; discard it;
1) read “random” next
2) might have to read it again in the future
level N

buffer pool

level N-1

remember hot blocks

why not use OS caching
os and db will typically refer to **pages**

hardware vendors might also refer to **sectors**
**employee**
(id:int, name:varchar(50), office:char(5), telephone:char(10), city:varchar(30), salary:int)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name1</td>
<td>office1</td>
<td>tel1</td>
<td>city1</td>
<td>salary1</td>
<td>name2</td>
<td>office2</td>
<td>tel2</td>
<td>city2</td>
<td>salary2</td>
</tr>
<tr>
<td></td>
<td>name3</td>
<td>office3</td>
<td>tel3</td>
<td>city3</td>
<td>salary3</td>
<td>name4</td>
<td>office4</td>
<td>tel4</td>
<td>city4</td>
<td>salary4</td>
</tr>
<tr>
<td></td>
<td>name5</td>
<td>office5</td>
<td>tel5</td>
<td>city5</td>
<td>salary5</td>
<td>name6</td>
<td>office6</td>
<td>tel6</td>
<td>city6</td>
<td>salary6</td>
</tr>
<tr>
<td></td>
<td>name7</td>
<td>office7</td>
<td>tel7</td>
<td>city7</td>
<td>salary7</td>
<td>name8</td>
<td>office8</td>
<td>tel8</td>
<td>city8</td>
<td>salary8</td>
</tr>
</tbody>
</table>
|   | name9 | office9 | tel9 | city9 | salary9

**data storage**
blocks < pages < files

**remember:** the way we store data defines the best possible way we can access it
### employee

(id:int, name:varchar(50), office:char(5), telephone:char(10), city:varchar(30), salary:int)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name1, office1, tel1, city1, salary1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>name2, office2, tel2, city2, salary2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>name3, office3, tel3, city3, salary3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>name4, office4, tel4, city4, salary4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>name5, office5, tel5, city5, salary5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>name6, office6, tel6, city6, salary6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>name7, office7, tel7, city7, salary7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>name8, office8, tel8, city8, salary8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>name9, office9, tel9, city9, salary9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
slotted page

free_offset, N, offset1-length1, offset2-length2,…

free space

scan
null
update
var length
…
things to “worry” about
how much data we transfer through the memory hierarchy
how many computations we do
row-store

select A, B, C, D

select A

A B C D

file

stored continuously

each page contains all fields for each entry (i.e., all attributes values)
row-store

column-store

stored continuously

each page contains fields of a single attribute

select $A,B,C,D$

select $A$
~1960s

1970: column storage ideas start appearing

~2000: open source complete system

1985: first rather complete column-store model

2005-now: more ideas and industry adoption of column-store designs

c-store, vertica, vectorwise and then ibm, microsoft, oracle, and more
column-store with materialized IDs

R(A,B,C)

header

row1

row2

row3

good idea
virtual ids/ positional alignment

columns do not need to have the same width

positional lookups/joins

\[ A(i) = A + i \times \text{width}(A) \]
ok so now we can selectively read columns but how do we process them?

option 1

option 2

early tuple reconstruction/materialization

row-store engine

column-store engine
it is not just memory and disk
we want to move as few data items as possible
all the way up to the CPU
select min(C) from R where A<10 & B<20

disk

memory

write the query plan and the code/logic of each operator
do not forget about intermediate results
describe data layouts at each step

(milestone1 of project)

no precise final answer is OK
understanding what matters is key
concepts & designs will be repeated >>1
late reconstruction/materialization

\[
\text{select } \min(C) \text{ from } R \text{ where } A<10 \land B<20
\]
late reconstruction/materialization

```sql
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\[ \text{select } \min(C) \text{ from } R \text{ where } A < 10 \text{ & } B < 20 \]

```c
1: int *input = A
2: for (i=0; i<tuples; i++, input++)
3:     if *input < 10
4:         *output = i
5:         output++
```

disk

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

memory

A < 10
late reconstruction/materialization

```sql
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\[
\text{select } \min(C) \text{ from } R \text{ where } A<10 \text{ & } B<20
\]
late reconstruction/materialization

```sql
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

```sql
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

```sql
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\[
\text{select } \min(C) \text{ from } R \text{ where } A<10 \land B<20
\]

always sequential access patterns
memory contains only what is needed at any point in time
Notes to remember

column-stores vs row-stores
it all starts with how we store the data
still basic concepts are the same
moving data is a major cost component
it is not just about disk…
the whole memory hierarchy matters
keep up with reading!

Read: **Architecture of a Database System** (Sections 1, 2, 3, 4)
by J. Hellerstein, M. Stonebraker and J. Hamilton

Read: **The Design and Implementation of Modern Column-store Database Systems**
by D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, S. Madden
basic db architectures & layouts

DATA SYSTEMS

prof. Stratos Idreos