Reminder

• HW1 due on Friday
Hints for HW1

Triple-representation of data:
• $T(\text{subject}, \text{predicate}, \text{object})$
  – E.g. (“Alice”, “age”, 30)
Hints for HW1

Triple-representation of data:
• $T$(subject, predicate, object)
  – E.g. (“Alice”, “age”, 30)
• Convenient for *semistructured data*:
  – Some attributes may be missing
  – Some subjects have extra attributes
  – Some attributes may be duplicates
Hints for HW1

Triple-representation of data:
• T(subject, predicate, object)
  – E.g. (“Alice”, “age”, 30)
• Convenient for *semistructured data*:
  – Some attributes may be missing
  – Some subjects have extra attributes
  – Some attributes may be duplicates

Structured data:
• Known schema: Person(name, age, city, …)
• How do we map semistructured to structured?
test=# select * from triples;

<table>
<thead>
<tr>
<th>key</th>
<th>attr</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Alice</td>
</tr>
<tr>
<td>1</td>
<td>age</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>city</td>
<td>Seattle</td>
</tr>
<tr>
<td>2</td>
<td>name</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>age</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>city</td>
<td>Portland</td>
</tr>
<tr>
<td>1</td>
<td>profession</td>
<td>SE</td>
</tr>
</tbody>
</table>

(7 rows)
Hints for HW1

test=# select * from triples;
key | attr | val
-----+-------+------
1    | name  | Alice
1    | age   | 20
1    | city  | Seattle
2    | name  | Bob
2    | age   | 30
2    | city  | Portland
1    | profession | SE
(7 rows)

? test=# select * from Person;
id | name | age | city
----+------|-----|------
1   | Alice| 20  | Seattle
2   | Bob  | 30  | Portland
(2 rows)
Hints for HW1

test=# select * from triples;

<table>
<thead>
<tr>
<th>key</th>
<th>attr</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Alice</td>
</tr>
<tr>
<td>1</td>
<td>age</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>city</td>
<td>Seattle</td>
</tr>
<tr>
<td>2</td>
<td>name</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>age</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>city</td>
<td>Portland</td>
</tr>
<tr>
<td>1</td>
<td>profession</td>
<td>SE</td>
</tr>
</tbody>
</table>

(7 rows)

create table Person as
(select x.key as id,
  x.val as name,
  y.val as age,
  z.val as city
from triples x,
  triples y,
  triples z
where x.key = y.key
  and x.key = z.key
  and x.attr='name'
  and y.attr='age'
  and z.attr='city');
Hints for HW1

Other options are possible:

- E.g. insert only the keys first, then update, separately, each field:

  update Person set name = … where …
  update Person set age = … where …
  update Person set city = … where …
Hints for HW1

test=# select * from triples;
key | attr | val
-----+-----------+---------
1 | name | Alice
1 | age | 20
1 | city | Seattle
2 | name | Bob
2 | age | 30
2 | city | Portland
1 | profession | SE
(7 rows)

? test=# select * from Empl;
id | name | profession
-----+-----------+---------
1 | Alice | SE
2 | Bob | 
(2 rows)
Hints for HW1

test=# select * from triples;
key | attr | val
-----+-----+------
1 | name | Alice
1 | age  | 20
1 | city | Seattle
2 | name | Bob
2 | age  | 30
2 | city | Portland
1 | profession | SE
(7 rows)

create table Empl as
(select x.key as id,
  x.val as name,
  y.val as profession
from triples x left outer join
  triples y
on x.key=y.key
and y.attr='profession'
where x.attr='name');
Hints for HW1

• Data is not code!
  – There is no perfect way to import DBLP
  – Where it’s ambiguous, use common sense
Hints for HW1

• Data is not code!
  – There is no perfect way to import DBLP
  – Where it’s ambiguous, use common sense

• NULLs or No-data: choose your poison
  – NULLs are bad: drop records that need NULL
  – Dropping records is bad: insert NULLs if needed
Hints for HW1

• Data is not code!
  – There is no perfect way to import DBLP
  – Where it’s ambiguous, use common sense

• NULLs or No-data: choose your poison
  – NULLs are bad: drop records that need NULL
  – Dropping records is bad: insert NULLs if needed

• Data cleaning = hacking of the worst kind
  – How to you check if a field is a homepage?
  – Use some regular expression…
Outline for Today

• Quick review of page/record formats
• PAX paper
• C-Store paper
  – Sec. 1,2 only
  – Will discuss the slides concerning Sec. 4 some other time
• Go back to last lecture (lecture 5):
  – finish discussing indexes
References

• Ailamaki et al. *Weaving Relations for Cache Performance*, VLDB’2001

• Daniel Abadi, et al., *The Design and Implementation of Modern Column-Oriented Database Systems* Foundations and Trends in Databases
Storage

[Architecture of a Database System, Hellerstein, Stonebraker, Hamilton]
Review

• Design choice: **One OS file for each relation**
  – Option 1: DBMS creates one big file with “files” inside
  – Option 2: DBMSs uses disk directly, with “files” inside

• The OS (or DBMS) provides an API of the form
  – Seek to some position (or “skip” over B bytes)
  – Read/Write B bytes
Review: Working with Pages

• Reading/writing to/from disk
  – Seeking takes a long time!
  – Reading sequentially is fast

• To simplify buffer manager, want to cache a collection of same-sized objects

• Solution: Read/write pages of data
  – A page should correspond to a disk block
Review: Page Formats

• 1 page = 1 disk block = fixed size (e.g. 8KB)

• Records:
  – Fixed length
  – Variable length

• Record id = RID
  – Typically RID = (PageID, SlotNumber)
Page Format: Fixed-length Records

Fixed-length records: packed representation
Divide page into slots. Each slot can hold one tuple
Record ID (RID) for each tuple is (PageID, SlotNb)

<table>
<thead>
<tr>
<th>Slot₁</th>
<th>Slot₂</th>
<th>Slotₙ</th>
<th>Free space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

Number of records
Page Format: Variable-length Records

Header contains slot directory
+ Need to keep track of nb of slots
+ Also need to keep track of free space (F)
Record Formats: Fixed-length Fields

Fixed-length records => Each field has a fixed length (i.e., it has the same length in all the records)

| Field 1 | Field 2 | ... | ... | Field K |

Information about field lengths and types is in the catalog
Record Formats: Variable-length Fields

Variable length records

Field 1  Field 2  ...  ...  Field K

Record header

Remark: NULLS require no space at all (why ?)
An Intermediate Format: PAX

• PAX = Partition Attributes Across

• Addresses memory access bottleneck (not the disk bottleneck)
Current Scheme: Slotted Pages

Formal name: NSM (N-ary Storage Model)

- Records are stored sequentially
- Offsets to start of each record at end of page

 RID | SSN  | Name  | Age |
-----|------|-------|-----|
 1   | 1237 | Jane  | 30  |
 2   | 4322 | John  | 45  |
 3   | 1563 | Jim   | 20  |
 4   | 7658 | Susan | 52  |
 5   | 2534 | Leon  | 43  |
 6   | 8791 | Dan   | 37  |

PAGE HEADER

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>RH1</th>
<th>RID</th>
</tr>
</thead>
</table>
| Jane | 30  | RH1 | 1237
| John | 4322| RH2 | 4322
| Jim  | 1563| RH3 | 7658
| Susan| 52  | RH4 | 8791

Ailamaki VLDB’01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Predicate Evaluation using NSM

select name from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
Predicate Evaluation using NSM

select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
Predicate Evaluation using NSM

NSM pushes non-referenced data to the cache

select name
from R
where age > 50

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
Predicate Evaluation using NSM

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt]
Predicate Evaluation using NSM

select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
Need New Data Page Layout

- Eliminates unnecessary memory accesses
- Improves inter-record locality
- Keeps a record’s fields together
- Does not affect I/O performance

and, most importantly, is...

low-implementation-cost, high-impact

Ailamaki VLDB’01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition Attributes Across (PAX)

**NSM PAGE**

<table>
<thead>
<tr>
<th>PAGE HEADER</th>
<th>RH1</th>
<th>1237</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>30</td>
<td>RH2</td>
</tr>
<tr>
<td>45</td>
<td>RH3</td>
<td>1563</td>
</tr>
<tr>
<td>7658</td>
<td>Susan</td>
<td>52</td>
</tr>
</tbody>
</table>

**PAX PAGE**

<table>
<thead>
<tr>
<th>PAGE HEADER</th>
<th>1237</th>
<th>4322</th>
</tr>
</thead>
<tbody>
<tr>
<td>1563</td>
<td>7658</td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td>Susan</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

Partition data *within* the page for spatial locality

Ailamaki VLDB’01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition Attributes Across (PAX)

Partition data *within* the page for spatial locality

Ailamaki VLDB'01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition Attributes Across (PAX)

Partition data *within* the page for spatial locality

Ailamaki VLDB'01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition data *within* the page for spatial locality

Ailamaki VLDB'01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition Attributes Across (PAX)

Partition data *within* the page for spatial locality

Ailamaki VLDB’01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Partition data within the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
Predicate Evaluation using PAX

```
select name
from R
where age > 50
```

Fewer cache misses, low reconstruction cost

Ailamaki VLDB’01 [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
Predicate Evaluation using PAX

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Fewer cache misses, low reconstruction cost
A Real NSM Record

NSM: All fields of record stored together + slots

Ailamaki VLDB’01  [http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt](http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt)
PAX: Detailed Design

PAX: Group fields + amortizes record headers

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt
PAX - Summary

• Improves processor cache locality
• Does not affect I/O behavior
  – Same disk accesses for NSM or PAX storage
  – No need to change the buffer manager

• Today:
  – Most (all?) commercial engines use a PAX layout of the disk
  – Beyond disk: Snowflake partitions tables horizontally into files, then uses column-store inside each file (hence, PAX)
Column-Store

• Store an entire attribute in a different file

• While the idea had been around before PAX, getting all the details right in order to extract the extra performance took a long time
From Row-Store to Column-Store

Rows stored contiguously on disk (+ tuple headers)

Columns stored contiguously on disk (no headers needed)
### C-Store Illustration

**Row-based (4 pages)**

<table>
<thead>
<tr>
<th>Page</th>
<th>A 1</th>
<th>A 2</th>
<th>A 2</th>
<th>A 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A 2</td>
<td>B 2</td>
<td>B 2</td>
<td>B 4</td>
</tr>
<tr>
<td></td>
<td>C 4</td>
<td>C 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Column-based (4 pages)**

<table>
<thead>
<tr>
<th>Page</th>
<th>A</th>
<th>A</th>
<th>1</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>C</td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

C-Store also avoids large tuple headers.

**Page**
Column-Oriented Databases

• Main idea:
  – **Physical storage**: complete vertical partition; each column stored separately: R.A, R.B, R.A
  – **Logical schema**: remains the same R(A,B,C)

• Main advantage:
  – **Improved transfer rate**: disk to memory, memory to CPU, better cache locality
Basic Trade-Off

• **Row stores**
  – Quick to update entire tuple (1 page IO)
  – Quick to access a single tuple

• **Column stores**
  – Avoid reading unnecessary columns
  – Better compression

• **Entire system needs a different design**
  – Not only storage manager
  – To achieve high performance
From Row to Column Storage (Modern Designs)

Figure 1.1: Physical layout of column-oriented vs row-oriented databases.

Basic tradeoffs:
- Reading all attributes of one record, v.s.
- Reading some attributes of many records
Figure 1.2: Performance of C-Store versus a commercial database system on the SSBM benchmark, with different column-oriented optimizations enabled.
Architectural Changes in C-Store

• We will discuss the remaining slides in a later lecture, after we study the query processor for a row-store
Key Architectural Trends (Sec. 1)

- Virtual IDs
- Block-oriented and vertical processing
- Late materialization
- Column-specific compression
Key Architectural Trends (Sec. 1)

• Virtual IDs
  – Offsets (arrays) instead of keys

• Block-oriented and vertical processing
  – Iterator model: one tuple → one block of tuples

• Late materialization
  – Postpone tuple reconstruction in query plan

• Column-specific compression
  – Much better than row-compression (why?)
Vectorized Processing

Review:

• Volcano-style iterator model
  – Next() method
  – Pipelining

• Materialization of all intermediate results

• Discuss in class:

\[
\text{select avg(A) from R where A < 100}
\]
Vectorized Processing

• Vectorized processing:
  – Next() returns a block of tuples (e.g. N=1000) instead of single tuple

• Pros:
  – No more large intermediate results
  – Tight inner loop for selection and/or avg

• Discuss in class:

  select avg(A) from R where A < 100
Compression (Sec. 4)

• What is the advantage of compression in databases?

• Discuss main column-at-a-time compression techniques
Compression (Sec. 4)

• What is the advantage of compression in databases?

• Discuss main column-at-a-time compression techniques
  – Row-length encoding: F,F,F,F,M,M \rightarrow 4F,2M
  – Bit-vector (see also bit-map indexes)
Compression (Sec. 4)

Row-based (4 pages)

<table>
<thead>
<tr>
<th>Page</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>A 2</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>A 2</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>B 2</td>
<td>B 2</td>
<td>B 4</td>
</tr>
<tr>
<td>B 2</td>
<td>B 2</td>
<td>B 4</td>
</tr>
<tr>
<td>C 4</td>
<td>C 4</td>
<td>C 4</td>
</tr>
<tr>
<td>C 4</td>
<td>C 4</td>
<td>C 4</td>
</tr>
</tbody>
</table>

Column-based (4 pages)

<table>
<thead>
<tr>
<th>Page</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>A 2</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>A 2</td>
<td>A 2</td>
<td>A 2</td>
</tr>
<tr>
<td>B 2</td>
<td>B 2</td>
<td>B 4</td>
</tr>
<tr>
<td>B 2</td>
<td>B 2</td>
<td>B 4</td>
</tr>
<tr>
<td>C 4</td>
<td>C 4</td>
<td>C 4</td>
</tr>
<tr>
<td>C 4</td>
<td>C 4</td>
<td>C 4</td>
</tr>
</tbody>
</table>

Compressed (2 pages)

<table>
<thead>
<tr>
<th>Page</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4XA</td>
<td>2XB</td>
<td>2XC</td>
</tr>
<tr>
<td>1X1</td>
<td>4X2</td>
<td>5X4</td>
</tr>
</tbody>
</table>
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a'} \land D='d')(R(A,B,C,D,...))$

  Early materialization:
  – Retrieve positions with 'a' in column A:
    2, 4, 5, 9, 25…
  – Retrieve those values in column D:
    'x', 'd', 'y', 'd', 'd'…
  – Retain only positions with 'd':
    4, 9, …
  – Lookup values in column B:
    B[4], B[9], …

• Late materialization
  – Retrieve positions with 'a' in column A:
    2, 4, 5, 9, 25…
  – Retrieve positions with 'd' in column D:
    3, 4, 7, 9, 12, ..
  – Intersect: 4, 9, …
  – Lookup values in column B:
    B[4], B[9], …
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a'} \wedge D='d')(R(A,B,C,D,...))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25...
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12, ...
  – Intersect: 4, 9, ...
  – Lookup values in column B: $B[4], B[9], ...$
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a'} \land D='d')(R(A,B,C,D,...))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…

• Late materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12,…
  – Intersect: 4, 9,…
  – Lookup values in column B: $B[4], B[9],…$
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a'\land D='d'}(R(A,B,C,D,...)))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retrieve those values in column D: ‘x’, ‘d’, ‘y’, ‘d’, ‘d’, ...
  – Retain only positions with ‘d’: 4, 9, ...
• Late materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12, ...
  – Intersect: 4, 9, …
Late Materialization (Sec. 4)

- What is it?
- Discuss $\Pi_B(\sigma_{A='a'} \land D='d')(R(A,B,C,D,...))$
- Early materialization:
  - Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25...
  - Retain only positions with ‘d’: 4, 9, ...
  - Lookup values in column B: B[4], B[9], ...

- Late materialization:
  - Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25...
  - Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12, ...
  - Intersect: 4, 9, ...
  - Lookup values in column B: B[4], B[9], ...
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A=\text{a}} \land D=\text{d}')(R(A,B,C,D,\ldots))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retrieve those values in column D: ‘x’, ‘d’, ‘y’, ‘d’, ‘d’, ...
  – Retain only positions with ‘d’: 4, 9, ...
  – Lookup values in column B: B[4], B[9], ...
• Late materialization
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a' \land D='d'}(R(A,B,C,D,...))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25...
  – Retain only positions with ‘d’: 4, 9, ...
  – Lookup values in column B: B[4], B[9], ...
• Late materialization
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25...
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12,..
Late Materialization (Sec. 4)

• What is it?
• Discuss \( \Pi_B(\sigma_{A='a'} \land D='d')(R(A,B,C,D,...)) \)

• Early materialization:
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retain only positions with ‘d’: 4, 9, …
  – Lookup values in column B: B[4], B[9], …

• Late materialization
  – Retrieve positions with ‘a’ in column A: 2, 4, 5, 9, 25…
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12,…
  – Intersect: 4, 9, …
Late Materialization (Sec. 4)

• What is it?
• Discuss $\Pi_B(\sigma_{A='a'} \land D='d')(R(A,B,C,D,\ldots))$
• Early materialization:
  – Retrieve positions with ‘a’ in column A:  2, 4, 5, 9, 25…
  – Retain only positions with ‘d’: 4, 9, …
  – Lookup values in column B: B[4], B[9], …
• Late materialization
  – Retrieve positions with ‘a’ in column A:  2, 4, 5, 9, 25…
  – Retrieve positions with ‘d’ in column D: 3, 4, 7, 9, 12,…
  – Intersect: 4, 9, …
  – Lookup values in column B: B[4], B[9], …
Late Materialization (Sec. 4)

Ex: \( \text{SELECT R.b from R where R.a=X and R.d=Y} \)

Early materialization

Late materialization

\[ R \]
\[ a \quad b \quad c \quad d \]

\[ \pi \]
\[ \sigma \]

\[ R \]
\[ a \quad b \quad c \quad d \]

Extract values

\[ 1 \quad 1 \quad 0 \quad 1 \]
\[ 0 \quad 0 \quad 1 \quad 0 \]
\[ \sigma \]
\[ 1 \quad 1 \quad 0 \quad 1 \]
\[
\]
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```
Jive Join (Sec. 4)

```sql
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```

Tuple positions

Add new indexes

One column will be out of order
Jive Join (Sec. 4)

SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id

emp.dept_id

One column will be out of order

department_id

Tuple positions

42 36
42 44
38

38 42 46
36

1 2
2 4
3 2
5 1

2 1
4 2
2 3
1 4

Add new indexes

1 4
2 1
2 3
4 2

sort

CSEP 544 - Spring 2021
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```

Tuple positions

<table>
<thead>
<tr>
<th>emp.dept_id</th>
<th>dept.id</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td>44</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td>5</td>
</tr>
</tbody>
</table>

One column will be out of order

<table>
<thead>
<tr>
<th>dept.name???</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Johnson</td>
</tr>
<tr>
<td>Jones</td>
</tr>
</tbody>
</table>

Add new indexes

Fetch dept.name
Jive Join (Sec. 4)

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```
Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
  and 5 < R.a < 20 and 40 < R.b < 50
  and 30 < S.a < 40
```
Late Materialization

```sql
select sum(R.a) from R, S
where R.c = S.b
  and 5 < R.a < 20 and 40 < R.b < 50
  and 30 < S.a < 40
```
select sum(R.a) from R, S
where R.c = S.b
    and 5<R.a<20 and 40<R.b<50
    and 30<S.a<40
Late Materialization

select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40

40, 50
Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
and 5<R.a<20 and 40<R.b<50
and 30<S.a<40
```
Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
  and 5<R.a<20 and 40<R.b<50
  and 30<S.a<40
```
select sum(R.a) from R, S
where R.c = S.b
and 5 < R.a < 20 and 40 < R.b < 50
and 30 < S.a < 40
Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
  and 5<R.a<20 and 40<R.b<50
  and 30<S.a<40
```
Late Materialization

```
select sum(R.a) from R, S
where R.c = S.b
  and 5 < R.a < 20 and 40 < R.b < 50
  and 30 < S.a < 40
```
More Details

• Sort columns according to some criterion
  – Helps with range queries on that column
  – Helps compressing that column
  – But need to sort all the other columns the same way

• Create additional (redundant) "views", called “projections”, by sorting on different columns
Vertica Data Model Details

Data organized into *projections*: 
Sorted subsets of the attributes 
Each table has one super projection 
Includes all table attributes

Original Data

<table>
<thead>
<tr>
<th>sale_id</th>
<th>cid</th>
<th>cust</th>
<th>date</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>Andrew</td>
<td>01/01/06</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>Chuck</td>
<td>01/05/06</td>
<td>$98</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>Nga</td>
<td>01/02/06</td>
<td>$90</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>Matt</td>
<td>01/03/06</td>
<td>$101</td>
</tr>
<tr>
<td>5</td>
<td>89</td>
<td>Ben</td>
<td>01/01/06</td>
<td>$103</td>
</tr>
<tr>
<td>1000</td>
<td>89</td>
<td>Ben</td>
<td>01/02/06</td>
<td>$103</td>
</tr>
<tr>
<td>1001</td>
<td>11</td>
<td>Andrew</td>
<td>01/03/06</td>
<td>$95</td>
</tr>
</tbody>
</table>

Split in two projections

Super projection sorted by date
Non-super projection containing only (cust, price) attributes, sorted by cust

From: The Vertica Analytic Database: CStore 7 Years Later. Lamb et. Al. VLDB’12
Parallel Processing

- Segment data horizontally across nodes
- Organize as column store on each node

Super projection sorted by date & segmented by hash(sale_id)
Non-super projection containing only(cust, price) attributes, sorted by cust, segmented by hash(cust)
Vertica Data Model Details

Split in two projections

- date
- price
- cust
- cid
- sale_id

Segmented on several nodes

Node 1

<table>
<thead>
<tr>
<th>date</th>
<th>price</th>
<th>cid</th>
<th>cust</th>
<th>sale_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/02/06</td>
<td>$90.00</td>
<td>27</td>
<td>Nga</td>
<td>3</td>
</tr>
<tr>
<td>01/03/06</td>
<td>$95.00</td>
<td>11</td>
<td>Andrew</td>
<td>1001</td>
</tr>
<tr>
<td>01/03/06</td>
<td>$101.00</td>
<td>28</td>
<td>Matt</td>
<td>4</td>
</tr>
</tbody>
</table>

Node 2

<table>
<thead>
<tr>
<th>date</th>
<th>price</th>
<th>cid</th>
<th>cust</th>
<th>sale_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/06</td>
<td>$100.00</td>
<td>11</td>
<td>Andrew</td>
<td>1</td>
</tr>
<tr>
<td>01/01/06</td>
<td>$103.00</td>
<td>89</td>
<td>Ben</td>
<td>5</td>
</tr>
<tr>
<td>01/02/06</td>
<td>$103.00</td>
<td>89</td>
<td>Ben</td>
<td>1000</td>
</tr>
<tr>
<td>01/05/06</td>
<td>$98.00</td>
<td>17</td>
<td>Chuck</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cust</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew</td>
<td>$95.00</td>
</tr>
<tr>
<td>Andrew</td>
<td>$100.00</td>
</tr>
<tr>
<td>Chuck</td>
<td>$98.00</td>
</tr>
<tr>
<td>Nga</td>
<td>$90.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cust</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td>$103.00</td>
</tr>
<tr>
<td>Ben</td>
<td>$103.00</td>
</tr>
<tr>
<td>Matt</td>
<td>$101.00</td>
</tr>
</tbody>
</table>
Vertica Data Partitioning

• Cross-node partitioning called “segmentation”
  – Hash-partitioning
  – Other expression

• Each node assigned multiple local segments
  – To facilitate elasticity
  – Enables moving segments as cluster size changes

• Can also replicate all tuples in projection
Vertica Intra-Node Partitioning

• C-store proposed intra-node data partitioning
  – Similar to other parallel DBMS such as Teradata

• In contrast, Vertica divides each on-disk structure into logical regions at runtime and processing the regions in parallel

• Vertica also supports explicit data partitioning
  – Partitions segments within nodes into smaller pieces
  – CREATE TABLE … PARTITION BY <expr>
  – Benefits:
    • Fast deletion
    • Pruning of partitions during query execution
Segmentation = horizontal partitioning across nodes
→ Each projection has own segmentation
→ More segments than nodes for elasticity
Partition = horizontal within a node
→ Same partition for all projections & nodes
ROS = Read Optimized Store
Each column’s data within its ROS container is stored as a single file
→ Total of 28 files of user data
Updates

• What is the issue?

• How does the paper address this?
Updates

• What is the issue?
  – Updates in a sorted column require reordering of the entire column, and the other columns as well

• How does the paper address this?
Updates

• What is the issue?
  – Updates in a sorted column require reordering of the entire column, and the other columns as well

• How does the paper address this?
  – Update to Write Optimized Store (WOS)
  – Queries on Read Optimized Store (ROS)
C-Store/Vertica Design

Figure 1. Architecture of C-Store

From: C-Store: A Column-oriented DBMS. Stonebraker et. Al. VLDB’05
Read and Write Optimized Stores

- **Write Optimized Store (WOS)**
  - In memory data: buffer delete/insert/update operations
  - Column vs row does not matter

- **Tuples never modified in place**
  - Use “delete vector” to track deleted tuples
  - Eventually removed by tuple mover during ROS merge

- **Tuple mover**
  - Move between WOS and ROS
  - When moving tuples out, creates a new ROS container
  - Merges ROS files together
    - Better compression & faster processing (fewer files to merge)
Read and Write Optimized Stores

• **Read Optimized Store (ROS)**
  – Multiple ROS containers
  – Stored on standard file system
  – Logically contains some number of complete tuples sorted by the *projection’s* sort order, stored as a pair of files per column: position index & data
    • The position index = only metadata per disk block
  – Column files may be independently retrieved
Final Thoughts

Simulating a Column-Store in a Row-Store DBMS:

• **Vertical partitioning**
  – Two-column tables: (key, attribute)

• **Index-only plans**
  – Create a B+ tree index on each attribute
  – Answer queries using indexes only, without reading actual data

• **Materialized views**
  – Each view contains a subset of columns