Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#25: Column Stores

Today's Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Distributed Execution

Wikipedia Example

```
CREATE TABLE useracct (
  userID INT PRIMARY KEY,
  userName VARCHAR UNIQUE,
  ⋮
);
```

```
CREATE TABLE pages (
  pageID INT PRIMARY KEY,
  title VARCHAR UNIQUE,
  latest INT REFERENCES revisions (revID),
);
```

```
CREATE TABLE revisions (
  revID INT PRIMARY KEY,
  pageID INT REFERENCES pages (pageID),
  userID INT REFERENCES useracct (userID),
  content TEXT,
  updated DATETIME
);
```


OLTP

- On-line Transaction Processing:
  - Short-lived txns.
  - Small footprint.
  - Repetitive operations.

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```

```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revisions AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

```
INSERT INTO revisions
VALUES (?,?…,?)
```

OLAP

- On-line Analytical Processing:
  - Long running queries.
  - Complex joins.
  - Exploratory queries.

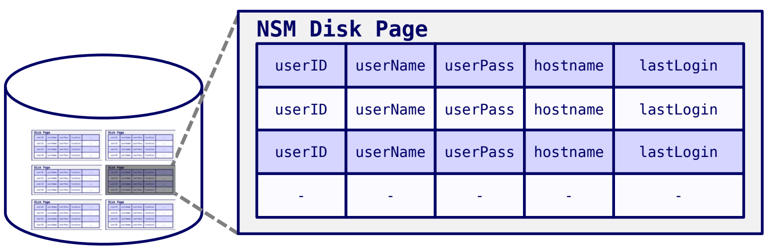
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```


Data Storage Models

- There are different ways to store tuples.
- We have been assuming the ***n*-ary storage model** this entire semester.

n-ary Storage Model

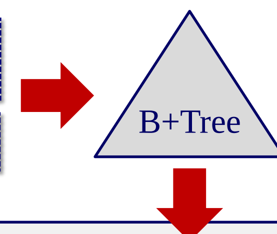
- The DBMS stores all attributes for a single tuple contiguously in a block.

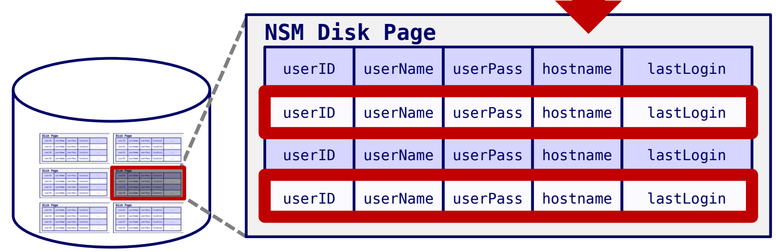


n-ary Storage Model

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```

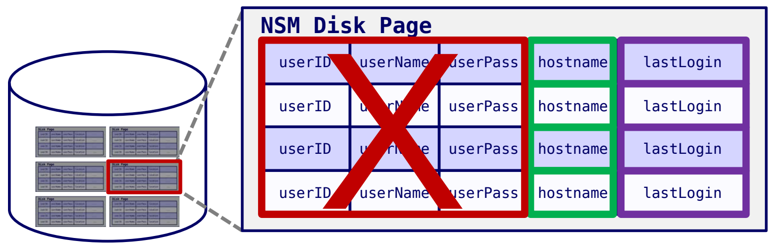
```
INSERT INTO useracct
VALUES (?,?,…?)
```





n-ary Storage Model

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```

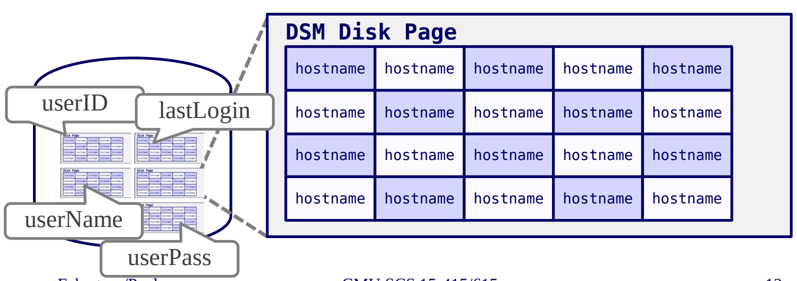


n-ary Storage Model

- Advantages
  - Fast inserts, updates, and deletes.
  - Good for queries that need the entire tuple.
- Disadvantages
  - Not good for scanning large portions of the table and/or a subset of the attributes.

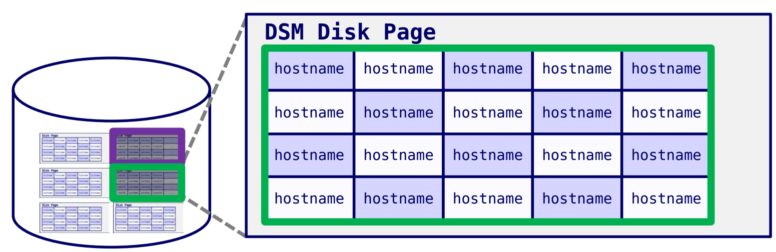
Decomposition Storage Model

- The DBMS stores a single attribute for all tuples contiguously in a block.



Decomposition Storage Model

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Decomposition Storage Model

- Advantages
  - Reduces the amount wasted I/O because the DBMS only reads the data that it needs.
  - Better compression (more on this later).
- Disadvantages
  - Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

History

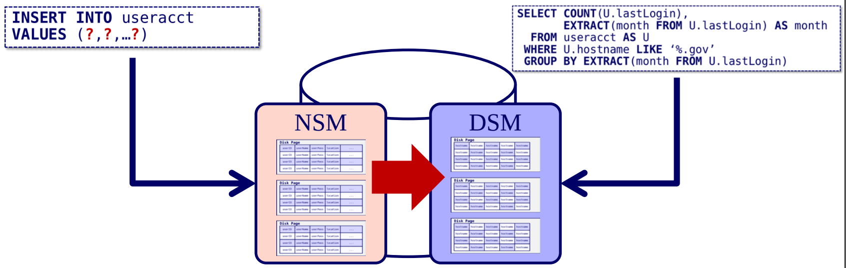
- **1970s:** Cantor DBMS
- **1980s:** DSM Proposal
- **1990s:** SybaseIQ (in-memory only)
- **2000s:** Vertica, Vectorwise, MonetDB
- **2010s:** Cloudera Impala, Amazon Redshift, "The Big Three"

System Architectures

- Fractured Mirrors
- Partition Attributes Across (PAX)
- Pure Columnar Storage

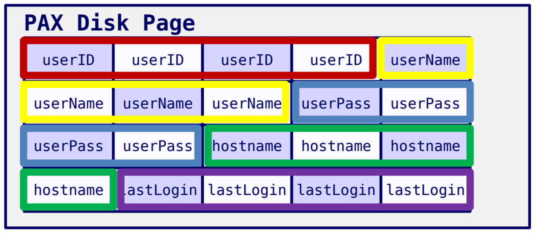
Fractured Mirrors

- Store a second copy of the database in a DSM layout that is automatically updated.
  - Examples: Oracle, IBM DB2 BLU



PAX

- Data is still stored in NSM blocks, but each block is organized as mini columns.

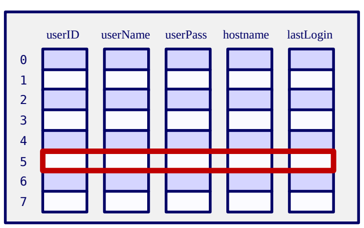


Column Stores

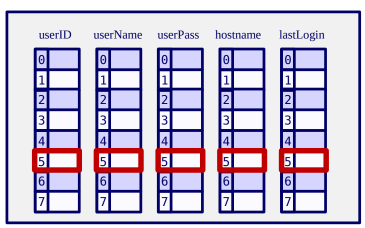
- Entire system is designed for columnar data.
  - Query Processing, Storage, Operator Algorithms, Indexing, etc.
  - Examples: Vertica, VectorWise, Paraccel, Cloudera Impala, Amazon Redshift

Virtual IDs vs. Offsets

- Need a way to stitch tuples back together.
- Two approaches:
  - Fixed length offsets
  - Virtual ids embedded in columns



Offsets



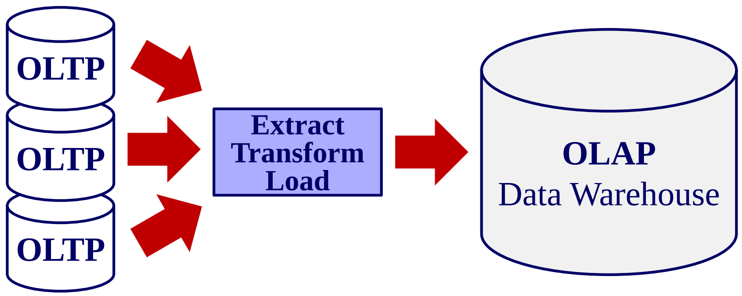
Virtual Ids

Modifying a Column Store

- **INSERT**:
  - Split tuple into attributes, append to columns.
- **DELETE**:
  - Mark the tuple as deleted in a separate bit-vector. Check visibility at runtime.
- **UPDATE**:
  - Implement as DELETE+INSERT

Bifurcated Architecture

- All txns are executed on OLTP database.
- Periodically migrate changes to OLAP database.



Today's Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Distributed Execution

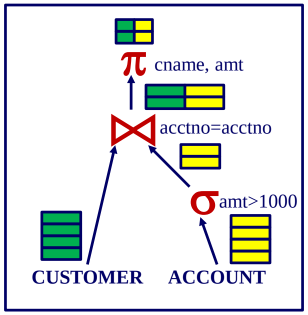
Query Processing Strategies

- The DBMS needs to process queries differently when using columnar data.
- We have already discussed the Iterator Model for processing tuples in the DBMS query operators.

Materialization Model

- Each operator consumes its entire input and generates the full output all at once.

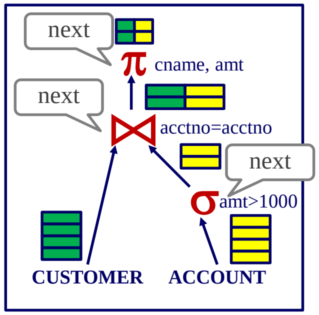
```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



Iterator Model

- Each operator calls **next()** on their child operator to process tuples one at a time.

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



Observations

- The Materialization Model is a bad because the intermediate results may be larger than the amount of memory in the system.
- The Iterator Model is bad with a DSM because it requires the DBMS to stitch tuples back together each time.

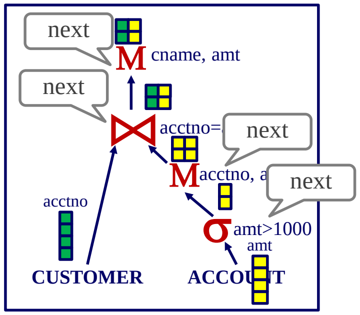
Vectorized Model

- Like the Iterator Model but each **next()** invocation returns a vector of tuples instead of a single tuple.
- This vector does not have to contain the entire tuple, just the attributes that are needed for query processing.

Vectorized Model

- Each operator calls **next()** on their child operator to process vectors.

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



Vectorized Model

- Reduced interpretation overhead.
- Better cache locality.
- Compiler optimization opportunities.

Today's Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Distributed Execution

Compression Overview

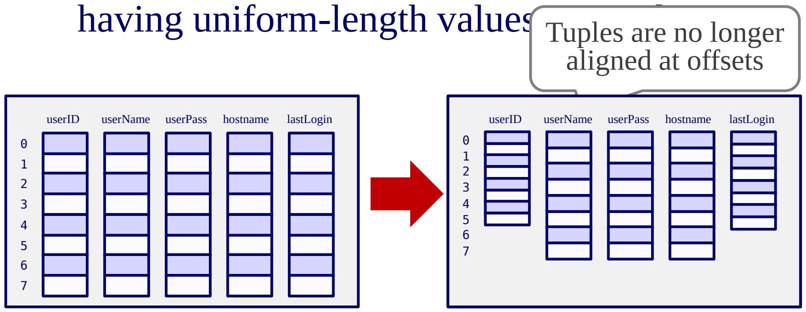
- Compress the database to reduce the amount of I/O needed to process queries.
- DSM databases compress much better than NSM databases.
  - Storing similar data together is ideal for compression algorithms.

Naïve Compression

- Use a general purpose algorithm to compress pages when they are stored on disk.
  - Example: 10KB page in memory, 4KB compressed page on disk.
- Do we have to decompress the page when it is brought into memory? Why or why not?

Fixed-width Compression

- Sacrifice some compression in exchange for having uniform-length values.

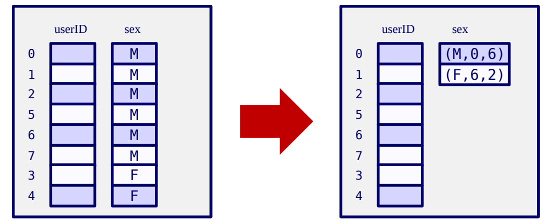


Original Data

Variable-Length / Fixed-Length Compression

Run-length Encoding

- Compress runs of the same value into a compact triplet:
  - `(value, startPosition, runLength)`

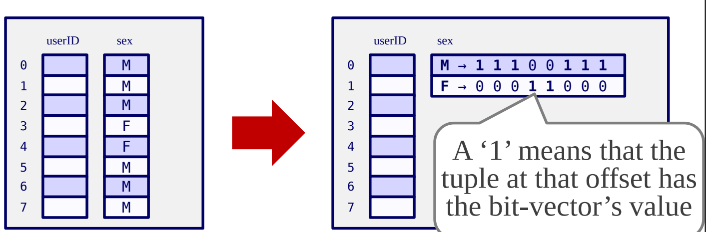


Original Data / Sorted Data

Unsorted RLE / Sorted RLE

Bit-Vector Encoding

- Store a separate bit-vector for each unique value for a particular attribute where an offset in the vector corresponds to a tuple.

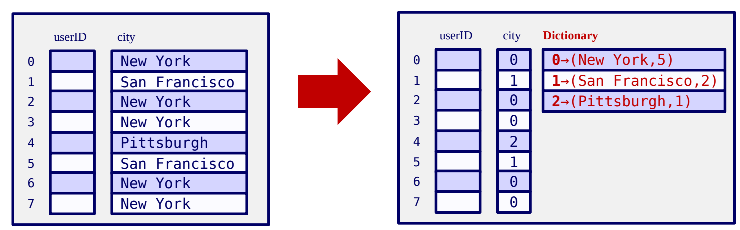


Original Data

Bit-Vector Compression

Dictionary Compression

- Construct a separate table of the unique values for an attribute sorted by frequency.
- For each tuple, store the position of its value in the dictionary instead of the real value.



Original Data

Dictionary Encoded

Processing Compressed Data

- Some operator algorithms can operate directly on compressed data
  - Saves I/O without having to decompress!
- Difficult to implement when the DBMS uses multiple compression schemes.
- It's generally good to wait as long as possible to materialize/decompress data when processing queries…

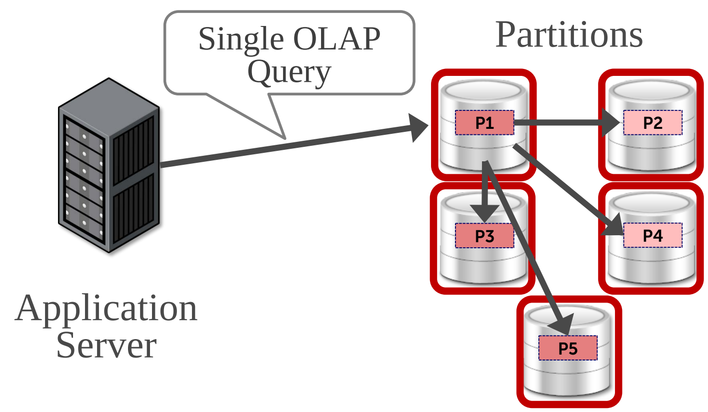
Today's Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Distributed Execution

Distributed OLAP

- Execute analytical queries that examine large portions of the database.
- Used for back-end data warehouses:
  - Example: Data mining
- Key Challenges:
  - Data movement.
  - Query planning.

Distributed OLAP



Distributed Joins Are Hard

```
SELECT * FROM table1, table2
 WHERE table1.val = table2.val
```

- Assume tables are horizontally partitioned:
  - Table1 Partition Key → table1.key
  - Table2 Partition Key → table2.key
- **Q:** How to execute?
- Naïve solution is to send all partitions to a single node and compute join.

Broadcast Join

- Main Idea: Send the smaller table to all nodes where the join is then computed in parallel.
  - Only works if the table is small.

Semi Join

- Main Idea: First distribute the join attributes between nodes and then recreate the full tuples in the final output.
  - Send just enough data from each table to compute which rows to include in output.
- Lots of choices make this problem hard:
  - What to materialize?
  - Which table to send?

Rest of the Semester

- **Wed Dec 2nd –** Data Warehousing + Mining
- **Mon Dec 7th –** Guest Speaker from MemSQL
- **Wed Dec 9th –** Final Review + Systems

http://cmudb.io/f15-systems