Today’s Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Data Modification

Wikipedia Example

```sql
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),
  content TEXT,
  updated DATETIME
);
```

OLTP

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

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

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

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

```sql
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 (?, ?, ?, ?)

B+Tree

NSM Disk Page

userID userName userPass lastLogin hostname

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

```sql
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)
```

**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**

```sql
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)
```

- **Advantages**
  - Reduces the amount wasted I/O because the DBMS only reads the data that it needs.
  - Better query processing and data 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

Today’s Class

• Storage Models
• System Architectures
• Vectorization
• Compression
• Data Modification
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.

Iterator Model

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

Materialization Model

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

Observations

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

VIRTUAL IDS vs. Offsets

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

- Reduced interpretation overhead.
- Better cache locality.
- Compiler optimization opportunities.
- AFAIK, VectorWise is still the only system that uses this model. Other systems use query compilation instead.

Today’s Class

- Storage Models
- System Architectures
- Vectorization
- Compression
- Data Modification

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 per attribute.

![Fixed-width Compression Diagram]

### Run-length Encoding

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

![Run-length Encoding Diagram]
**Delta Encoding**

- Record the difference between successive values in the same column.

**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.

**Dictionary Compression**

- Replace frequent patterns with smaller integer codes.
  - Need to support fast encoding and decoding.
  - Need to also support range queries.
Dictionary Compression

• Construct a separate table of the unique values for an attribute sorted by value.

```
SELECT * FROM users
WHERE name LIKE 'Tru%'
```

Original Data

```
<table>
<thead>
<tr>
<th>userId</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Truman</td>
</tr>
<tr>
<td>102</td>
<td>Obama</td>
</tr>
<tr>
<td>103</td>
<td>Bush</td>
</tr>
<tr>
<td>104</td>
<td>Reagan</td>
</tr>
<tr>
<td>105</td>
<td>Trump</td>
</tr>
<tr>
<td>106</td>
<td>Nixon</td>
</tr>
<tr>
<td>107</td>
<td>Carter</td>
</tr>
<tr>
<td>108</td>
<td>Ford</td>
</tr>
</tbody>
</table>
```

Compressed Data

```
<table>
<thead>
<tr>
<th>userIdx</th>
<th>name</th>
<th>value</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101</td>
<td>70</td>
<td>Bush</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
<td>50</td>
<td>Carter</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>10</td>
<td>Ford</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>60</td>
<td>Nixon</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>80</td>
<td>Obama</td>
</tr>
<tr>
<td>5</td>
<td>106</td>
<td>60</td>
<td>Reagan</td>
</tr>
<tr>
<td>6</td>
<td>107</td>
<td>20</td>
<td>Truman</td>
</tr>
<tr>
<td>7</td>
<td>108</td>
<td>30</td>
<td>Trump</td>
</tr>
</tbody>
</table>
```

Dictionary Compression

• A dictionary needs to support two operations:
  – `Encode`: For a given uncompressed value, convert it into its compressed form.
  – `Decode`: For a given compressed value, convert it back into its original form.

• We need two data structures to support operations in both directions.

Summary

• 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
• Data Modification
**Bifurcated Architecture**

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

**Delta Store**

- Stage updates in delta store and periodically apply them in batches to the main storage.
  - Examples: Vertica, SAP HANA

**Modifying a Column Store**

- Updating compressed data is expensive.
- Updating sorted data is expensive.

- The DBMS will store updates in an staging area and then apply them in batches.
  - Have to make sure that we execute queries on both the staging and main storage.

**HTAP**

- **Hybrid Transaction-Analytical Processing**
- Single database instance that can handle both OLTP workloads and OLAP queries.
  - Row-store for OLTP
  - Column-store for OLAP
  - Examples: SAP HANA, MemSQL, HyPer, SpliceMachine, Peloton, Cloudera Kudu (???)