Many aggregations are done over continuous ranges. For example, you don’t aggregate sales randomly, but over date ranges (say last 2 years).

Hence, it is worth it to optimize range queries.

It is also neat mathematically speaking.
References for this lecture
Ho et al., Range Queries in Data Cubes

Geffner et al., Relative Prefix Sums

Poon, Optimal Range Max Datacube for Fixed Dimensions

Lemire, Wavelet-Based Relative Prefix Sum
Kaser-Lemire, Ola

Schmidt, ProPolyne
What is OLAP after all?

- OLAP mission: pretty pictures, pivot tables...
- OLAP strength: FAST queries
- Fast for all queries?
- No. There will always be expensive queries.
- Fast for TYPICAL queries.
Typical Queries?

- What are some **typical** queries?

- Generate a Pivot Table given Measures

- Measures are Max, Min, Sum, Variance, Median, Standard Deviation and more
Sources of confusion

- Queries can be over the measures or over the attribute values
- This is very confusing!
Example: milk in my fridge

<table>
<thead>
<tr>
<th></th>
<th>age = 1 day</th>
<th>age = 2 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=plain</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>type=chocolate</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Sources of confusion

- The max. is what? 3 or (type=plain, age=1 day) or even age = 2 days

- The mean is what? Mean of measures is \( \frac{7}{4} \), mean of time is \( \frac{5 \times 1 + 2 \times 2}{7} = \frac{9}{7} \).

- Lesson: in a multidimensional space, queries can be confusing.
Classification of Queries

- Range Queries: a single, expensive aggregate (group of cells → value).

- Other queries: single cell, simple views (slice query), multiple aggregates (rollup)
Examples of Range Queries

- Average sales per day for month of July.
- Number of salesmen in New Jersey, 55 years old or more.
- Number of students with unpaid fees.
- Total amount of unpaid fees across all faculties.
Range Queries versus ROLAP/MOLAP

- If you want to buffer range queries, it is easier if your buffer is the size of the dense cube or more.

- Hence, fast range queries often implies MOLAP.

- This may become more practical as storage becomes infinite (in 5 years?).
Categories of Queries

- DISTRIBUTIVE. Very easy to buffer.

- ALGEBRAIC. Somewhat easy to buffer.

- HOLISTIC. Hard problems. (Ph.D. topic?)

Beyer and Ramakrishnan, Bottom-Up Computation of Sparse and Iceberg CUBEs.
Distributive Queries in pictures
Distributive Queries

- Distributive Queries include RANGE COUNT, RANGE SUM, RANGE MAX, RANGE MIN.

- Further aggregation (bottom-up) is always doable using a unique aggregation operation.

- For a distributive aggregate function $F$, given $F(X_1, \ldots, X_j)$ and $F(X_j + 1, \ldots, X_N)$, then $F(X_1, \ldots, X_n) = F(F(X_1, \ldots, X_j), F(X_j + 1, \ldots, X_N))$. 
Distributive Queries (Example)

▷ Example: \( \max\{1, 2, -1\} = 2 \) and \( \max\{3, 2, 1\} = 3 \)

▷ so \( \max\{1, 2, -1, 3, 2, 1\} = \max\{2, 3\} = 3. \)
Algebraic Queries in pictures

Data Cube

Final Aggregate

\[(\text{sum, count})\]

- 4,2
- 3,2
- 11,2
- 3,2
- 14,4
- 21,4

Lemire - CS6905ateb
Algebraic Queries

- Algebraic Queries include RANGE AVERAGE, RANGE VARIANCE, RANGE STANDARD DEVIATION.

- Further aggregation (bottom-up) is always possible, but you must aggregate tuples.

- Distributive $\Rightarrow$ Algebraic
Algebraic Queries (Formal)

- For a distributive aggregate function $F$

- given $H(X_1, \ldots, X_j)$ and $H(X_j + 1, \ldots, X_N)$

- $(H$ might be vector-valued$)$

- then

$$F(X_1, \ldots, X_n) = G(H(X_1, \ldots, X_j), H(X_j + 1, \ldots, X_N)).$$
Algebraic Queries (Examples)

- $\text{average}\{1, 2, -1\} = \frac{2}{3}$

- and $\text{average}\{3, 2, 1\} = 2$,

- so $\text{average}\{1, 2, -1, 3, 2, 1\} = \frac{2\times3 + 2\times3}{6}$. 
Holistic Queries

- Whatever is not Algebraic is Holistic

- Holistic Queries include RANGE MEDIAN, RANGE PERCENTILE.

- Further aggregation (bottom-up) is not possible.

- These queries are HARD to buffer (no bottom-up possible)
Other classifications

Poon differentiate between orthogonal and non-orthogonal range queries.

For orthogonal: range must be sub-cube (DICE).

Not clear in Poon’s paper whether this is the only constraint.
Algebraic Orthogonal Range Queries

- Range Sum Queries (simplest): Geffner, Lemire and others

- Polynomial Range Sum Queries (average, variance, stddev): Schmidt.

- Range Maximum Queries: Poon and others.
ProPolyne

- Sum, Average, Variance, Standard Deviation are Polynomial Range Queries.

- Can be written in terms of

\[
\sum_{k_1,k_2,\ldots,k_N} k_1^{p_1} \times \cdots \times k_N^{p_N} C_{k_1,k_2,\ldots,k_N}
\]

(These can be viewed as inner products of \(k_N^{p_N}\) against \(C_{k_1,k_2,\ldots,k_N}\) in \(L_2\).)
Question

You guys know what $L_2$, Hilbert Space and all these things are, right?
## Polynomial Range Queries: Example

<table>
<thead>
<tr>
<th>day=1</th>
<th>day=2</th>
<th>day=3</th>
<th>day=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Given 1D cube above, sum is \( \sum_{day=1}^{4} C_{day} = 1 + 3 + 2. \)
Polynomial Range Queries: Example (part 2)

Average is

\[
\frac{\sum_{day=1}^{4} d \cdot c_{day}}{\sum_{day=1}^{4} c_{day}} = \frac{1 + 3 \times 2^1 + 2 \times 3^1}{1 + 3 + 2}.
\]
Polynomial Range Queries: Proof by 2 examples

Average and sum can be written in terms of

\[ \sum_{k_1, k_2, \ldots, k_N} k_1^{p_1} \times \cdots \times k_N^{p_N} C_{k_1, k_2, \ldots, k_N} \]
ProPolyne: What is that?

- Key Idea: Buffer Cube using *Wavelets!*

- Wavelets have null moments ($\sum_k k^p \text{wavelet}_k = 0$)

- Orthogonal Wavelet Transform $T$ preserve inner product have null moments ($\sum_k k^p C_k = \sum T(k^p) T(C_k)$)

- Similar Idea: Fast Operations on Large Numbers using FFT
ProPolyne with a Picture

You solve the queries against a “transformed cube”.

[Diagram of transformed cubes]
Simplest Example of Wavelet Transform

- We don’t have time for full picture, just taste
- Haar (1910), 1 null moment
- Given data $a, b, c, d$
- Do $\left\{ \frac{a+b}{\sqrt{2}}, \frac{c+d}{\sqrt{2}} \right\}$ and $\left\{ \frac{a-b}{\sqrt{2}}, \frac{c-d}{\sqrt{2}} \right\}$.
- Iterate $\left\{ \frac{a+b+c+d}{2}, \frac{a+b-c-d}{2} \right\}$ and $\left\{ \frac{a-b}{\sqrt{2}}, \frac{c-d}{\sqrt{2}} \right\}$. 
Haar: Null Moments and Inner Products

- Check that it preserves inner product

\[ \langle (1, 0, 1, 0), (0, 0, 1, 1) \rangle = \langle (1, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), (1, -1, 0, 0) \rangle \]

- 1 Null Moment: \( T(1, 1, 1, 1) = (2, 0, 0, 0) \) (mostly zeroes)
Wavelet Transform with a Picture
ProPolyne: What is that again?

- Polynomial Range Sums = ⟨polynomial, cube⟩

- Wavelet Transform $T$ preserve inner product

- Polynomial Range Sums = ⟨$T$(polynomial), $T$(cube)⟩

- With Wavelets, $T$(polynomial) is sparse (mostly zero).

- If $T$(cube) is precomputed, $T$(polynomial) is sparse ⇒ efficient computation!
Did you get that the first time?

- inner product (polynomial, cube)

- $T$ preserve inner product

- inner product (polynomial, cube) = inner product $(T(\text{polynomial}), T(\text{cube}))$

- If $T(\text{polynomial})$ mostly zero, fast computation.

- Fundamental principle of algorithms!
ProPolyne Summary

- Fast queries (log) for all poly. range sums with $O(n^d)$ buffer
- Fast updates (log)
- Wavelet-Based
Rushed Intro to PyRPS

- Technique for $O(1)$ Range Sums with $O(n^d)$ buffer
- Fast updates (log)
- Wavelet-Based (or wavelet inspired)
A Tiny Cube

How many between indices 2 to 5?

✔ 5+6+7+2 = 20!

✔ Time: $O(n)$ (bad)
The Prefix Sum Method!

Can do MUCH better!

| 1 | 5 | 6 | 7 | 2 | 1 | 0 | 3 | 1 | 6 | 12 | 19 | 21 | 22 | 22 | 25 |

✔ 21 - 1 = 20!

✔ Much more scalable! $O(1)$

✔ Updates are painful! $O(n)$
Our strategy

✔ Computing global prefix sums is enough

✔ Can we do it in a smarter way?
A Pyramidal Alternative

Original data

One-step transform

Two-step transform
Base $b$ - Transform

1. Choose a basis ($b=2$)

2. Initial data

| 1 | 5 | 6 | 7 | 2 | 1 | 0 | 3 |

3. Prefix Sums over $b$ items

| 1 | 6 | 6 | 13 | 2 | 3 | 0 | 3 |

4. So far, just “relative prefix sums”
Two-step Base $b$ - Transform

1. Result from previous

   
   | 1 | 6 | 6 | 13 | 2 | 3 | 0 | 3 |

2. Fix first $b - 1$ items of each series

   
   | 1 | 6 | 6 | 13 | 2 | 3 | 0 | 3 |

3. Repeat transform

   
   | 1 | 6 | 6 | 13+6 | 2 | 3 | 0 | 3+3 |
Three-step Base $b$ - Transform

1. Result from previous

| 1 | 6 | 6 | 13+6 | 2 | 3 | 0 | 3+3 |

2. Fix first $b-1$ items

| 1 | 6 | 6 | 19 | 2 | 3 | 0 | 6 |

3. Repeat transform

| 1 | 6 | 6 | 19 | 2 | 3 | 0 | 6+19 |

4. The end!
Going from One Dimension to Several

Generalize to $d$ dimension? direct product!

Practical application: apply on each dimension (rows, columns)
1D - Computing Range Sums

Compute first 7 terms:

1. \(0 \rightarrow 1\)

2. \(0 + 3 \rightarrow 1\)

3. \(0 + 3 + 19 \rightarrow [1\]
A First Theorem

Let $\beta = \log_b n$.

**THEOREM 1 (1D):** Compute Prefix Sums in time $O(\beta)$.

**THEOREM 1 (multiD):** Compute Prefix Sums in time $O(\beta^d)$.

**Proof:** formula for queries involves multiple sums $\sum_{r_1=0}^{\beta-1} \cdots \sum_{r_d=0}^{\beta-1}$. 
A problem?

Queries take a time $O(\log_b n)$? Isn’t that logarithmic time?
Fix $\beta$ and choose $b = n^{1/\beta}$. 
1D - Updating The Transform

1. Update local:  

2. Update higher (green):  

3. Update top-level (red):
A Second Theorem

Note: \( \beta = \log_b n, \ b = n^{1/\beta} \)

**THEOREM 2 (1D):** Update transform in time \( O((b - 1)\beta) = O(\beta n^{1/\beta}) \)

**THEOREM 2 (multiD):** Update transform in time \( O(\beta^d (b - 1)^d) = O(\beta^d n^{d/\beta}) \)
Multidimensional Transformation

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>5</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

\[ \rightarrow \]

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>8</th>
<th>9</th>
<th>2</th>
<th>4</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>18</td>
<td>21</td>
<td>8</td>
<td>18</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>29</td>
<td>11</td>
<td>24</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>14</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>40</td>
<td>50</td>
<td>25</td>
<td>45</td>
<td>126</td>
<td></td>
</tr>
</tbody>
</table>
## Multidimensional Updates

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3+0.1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.1</th>
<th>8.1</th>
<th>9.1</th>
<th>2</th>
<th>4</th>
<th>17.1</th>
<th>6</th>
<th>9</th>
<th>30.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>18.1</td>
<td>21.1</td>
<td>8</td>
<td>18</td>
<td>50.1</td>
<td>7</td>
<td>12</td>
<td>67.1</td>
</tr>
<tr>
<td>12.1</td>
<td>24.1</td>
<td>29.1</td>
<td>11</td>
<td>24</td>
<td>67.1</td>
<td>11</td>
<td>21</td>
<td>97.1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>19</td>
<td>2</td>
<td>10</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>14</td>
<td>36</td>
<td>9</td>
<td>18</td>
<td>57</td>
</tr>
<tr>
<td>21.1</td>
<td>40.1</td>
<td>50.1</td>
<td>25</td>
<td>45</td>
<td>126.1</td>
<td>25</td>
<td>45</td>
<td>185.1</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>28</td>
<td>3</td>
<td>6</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>19</td>
<td>9</td>
<td>13</td>
<td>42</td>
<td>12</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>28.1</td>
<td>59.1</td>
<td>74.1</td>
<td>35</td>
<td>74</td>
<td>178.1</td>
<td>45</td>
<td>71</td>
<td>275.1</td>
</tr>
</tbody>
</table>
## How does our method compare?

<table>
<thead>
<tr>
<th>method</th>
<th>Query cost</th>
<th>Update cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Cube</td>
<td>$O(n^d)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Prefix Sum</td>
<td>$O(1)$</td>
<td>$O(n^d)$</td>
</tr>
<tr>
<td>RPS</td>
<td>$O(1)$</td>
<td>$O(n^{d/2})$</td>
</tr>
<tr>
<td>PyRPS</td>
<td>$O(1)$</td>
<td>$O(n^{d/\beta})$, $\beta = 1, 2, \ldots$</td>
</tr>
</tbody>
</table>
Do you have some real benchmarks?

Pentium 4 laptop running Java 1.4 (Sun)

<table>
<thead>
<tr>
<th>$b$</th>
<th>$\beta$</th>
<th>transform (min)</th>
<th>prefix sum ($\mu s$)</th>
<th>update ($\mu s$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2 (RPS)</td>
<td>43.9</td>
<td>1.6</td>
<td>3105</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>44.0</td>
<td>3.9</td>
<td>274</td>
</tr>
<tr>
<td>2</td>
<td>8 (Best!)</td>
<td>43.9</td>
<td>7.9</td>
<td>98</td>
</tr>
</tbody>
</table>

$n = 256, d = 3 (64\, Megs)$
Range Maximum Queries

- Constant time queries $O(1)$
- Storage amount to original cube $O(n^d)$ being buffered
- Fast updates
Sketch of Poon’s algo (1D)

- Divide in large chunks (size $m$), for each chunk precompute the max, do prefix max ($log$-$star$ macro-structure)

- Inside the chunks, buffer using words (each subinterval $[i,j]$ has $\log(m)$ bit pointing to the position of the max). Total $m \log(m)$ storage. For $m = 16$, that’s $16 \times 4 = 64$.

- KEY INSIGHT: Cheaper to store index than value!
Example of Poon’s algo (1D)

Max over $[8, 32]$ with $m = 10$ is max over $[8, 10]$ (micro), $[10, 20]$ (macro), $[20, 30]$ (macro), $[30, 32]$ (micro)