What if indices are not enough for decent online performance?

- Buy RAM
- Use a column database
  In analytics queries can give a 10x easily
- Scalable, parallel processing
  Mostly via no SQL
- **Precompute**
  Fast answers!
  Penalty: Cost of maintaining precomputed results
  Applicability depends on schema and queries
  Star schemas and summation are a good (but not the only) target of precomputation

Precomputation problems

- Choose what data to precompute
- Use the precomputed data smartly in your queries
- Update smartly the precomputed data as the database changes

The problem of smart update of the precomputed tables

Given
- database tables R1, R2, ..., Rn
- precomputed table V, computed from R1, R2, ..., Rn
- a list of insertions DR1+, DR2+, ... that are inserted in the tables R1, R2, ...
  General case may also have deletes and updates

Update V incrementally, so that it reflects the state of the tables after the insertions

Complete precomputation Vs Incremental maintenance

<table>
<thead>
<tr>
<th>Snapshot 0</th>
<th>Problem: Application efficiently updates V so that its new state is $V^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precomputed table state before insertions</td>
<td>$V^0 = f(R_1^0, ..., R_n^0)$</td>
</tr>
<tr>
<td>Database tables state before insertions</td>
<td>$R_1^0, ..., R_n^0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table Insertions</th>
<th>Precomputed table state after insertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta R_1^+, ..., \Delta R_n^+$</td>
<td>$V^1 = f(R_1^1, ..., R_n^1)$</td>
</tr>
<tr>
<td>$R_1^1, ..., R_n^1$</td>
<td>Database tables state after insertions</td>
</tr>
</tbody>
</table>

Example

Database has huge table `Sales(product, store, date, amt)`

Application issues often this slow query and displays the results

```sql
SELECT product, SUM(amt) AS sumamt
FROM Sales
GROUP BY product
```

To improve performance we precompute table

```sql
ProductSales(product, sumamt)
```

and insert in it the precomputed data by

```sql
INSERT INTO ProductSales (SELECT product, SUM(amt) AS sumamt FROM Sales GROUP BY product )
```

Now the application issues instead this fast query below

```sql
SELECT * FROM ProductSales
```

Example (cont’d)

Now we have to keep up to date the

```sql
ProductSales(product, sumamt)
```
as new sales happen. E.g., if another $10 of product 23 were just sold

```sql
UPDATE ProductSales
SET sumamt = sumamt + 10
WHERE product = 23
```

(in actual code it will use prepared queries)
Star Schemas

Terms

- Fact table
- Dimension tables
- Measures

Cube

3-D Cube
Aggregates

- Add up amounts for day 1
  SELECT sum(amt) FROM SALE
  WHERE date = 1

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Add up amounts by day

- Add up amounts by day
  SELECT date, sum(amt) FROM SALE
  GROUP BY date

<table>
<thead>
<tr>
<th>date</th>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td>81</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td>48</td>
</tr>
</tbody>
</table>

Another Example

- Add up amounts by day, product
  SELECT date, sum(amt) FROM SALE
  GROUP BY date, prodId

<table>
<thead>
<tr>
<th>date</th>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Cube Aggregation

Example: computing sums

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>12</td>
<td>50</td>
<td>129</td>
</tr>
</tbody>
</table>

Cube Operators

- Calculating sums
  sale(c1, *)
  sale(c2, *)
  sale(c3, *)

- Rollup and drill-down operations

- “Having” clause
- Using dimension hierarchy
  average by region (within store)
  maximum by month (within date)
Extended Cube

Aggregation Using Hierarchies

Pivoting

Materialized Views

What to Materialize?

Cube Aggregates Lattice
Cube Aggregates Lattice
Example assumes fact table is sales(city, product, amt) and cities classify into regions

Should one precompute joins?
• Notice that we have featured foreign keys, not printable values. Why?
• Why (city product) and not (city region product)?
• Minor penalty to find the cities of a particular region
• Probably larger penalty by having a larger table
  Think space in storage and time to scan it