Data integration

• Data resides in many distributed, heterogeneous OLTP (On-Line Transaction Processing) sources
  • Sales, inventory, customer, ...
  • NC branch, NY branch, CA branch, ...

• Need to support OLAP (On-Line Analytical Processing) over an integrated view of the data

• Possible approaches to integration
  • Eager: integrate in advance and store the integrated data at a central repository called the data warehouse
  • Lazy: integrate on demand; process queries over distributed sources—mediated or federated systems
OLTP versus OLAP

OLTP
• Mostly updates
• Short, simple transactions
• Clerical users, end users
• Goal: transaction throughput

OLAP
• Mostly reads
• Long, complex queries
• Analysts, decision makers
• Goal: fast queries

Implications on database design and optimization?
OLAP databases do not care much about redundancy
  ▪ “Denormalize” tables
  ▪ Many, many indexes
  ▪ Precomputed query results
Eager versus lazy integration

Eager (warehousing)
• In advance: before queries
• Copy data from sources
  👎 Answer could be stale
  👎 Need to maintain consistency
  👎 Query processing is local to the warehouse
    • Faster
    • Can operate when sources are unavailable

Lazy
• On demand: at query time
• Leave data at sources
  👎 Answer is more up-to-date
  👎 No need to maintain consistency
  👎 Sources participate in query processing
    • Slower
    • Interferes with local processing
    • Still has consistency issues
Maintaining a data warehouse

• The “ETL” process
  • Extract relevant data and/or changes from sources
  • Transform data to match the warehouse schema
  • Load/integrate data/changes into the warehouse

• Approaches
  • Recomputation
    • Easy to implement; just take periodic dumps of the sources, say, every night
    • What if there is no “night,” e.g., a global organization?
    • What if recomputation takes more than a day?
  • Incremental maintenance
    • Compute and apply only incremental changes
    • Fast if changes are small
    • Not easy to do for complicated transformations
    • Need to detect incremental changes at the sources
“Star” schema of a data warehouse

**Dimension table**

<table>
<thead>
<tr>
<th>PID</th>
<th>name</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>beer</td>
<td>10</td>
</tr>
<tr>
<td>p2</td>
<td>diaper</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Store**

<table>
<thead>
<tr>
<th>SID</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Durham</td>
</tr>
<tr>
<td>s2</td>
<td>Chapel Hill</td>
</tr>
<tr>
<td>s3</td>
<td>RTP</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Sale**

<table>
<thead>
<tr>
<th>OID</th>
<th>Date</th>
<th>CID</th>
<th>PID</th>
<th>SID</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>08/23/2015</td>
<td>c3</td>
<td>p1</td>
<td>s1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>102</td>
<td>09/12/2015</td>
<td>c3</td>
<td>p2</td>
<td>s1</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>105</td>
<td>09/24/2015</td>
<td>c5</td>
<td>p1</td>
<td>s3</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Customer**

<table>
<thead>
<tr>
<th>CID</th>
<th>name</th>
<th>address</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3</td>
<td>Amy</td>
<td>100 Main St.</td>
<td>Durham</td>
</tr>
<tr>
<td>c4</td>
<td>Ben</td>
<td>102 Main St.</td>
<td>Durham</td>
</tr>
<tr>
<td>c5</td>
<td>Coy</td>
<td>800 Eighth St.</td>
<td>Durham</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Fact table**

- Big
- Constantly growing
- Stores measures (often aggregated in queries)

**Dimension table**

- Small
- Updated infrequently
Data cube

Simplified schema: Sale (CID, PID, SID, qty)
Completing the cube—plane

Total quantity of sales for each product in each store

SELECT PID, SID, SUM(qty) FROM Sale
GROUP BY PID, SID;

Project all points onto Product-Store plane
Completing the cube—axis

Total quantity of sales for each product

SELECT PID, SUM(qty) FROM Sale GROUP BY PID;

Further project points onto Product axis
Completing the cube—origin

Total quantity of sales

```
SELECT SUM(qty) FROM Sale;
```

Further project points onto the origin

(ALL, p2, ALL) = 2

(ALL, p1, ALL) = 9

(ALL, p1, s1) = 4

(ALL, p2, s1) = 2

(c5, p1, s3) = 5

(c3, p1, s1) = 1

(c5, p1, s1) = 3

(Customer, Store, Product)

(ALL, ALL, ALL) = 11
CUBE operator

- **Sale** \((CID, PID, SID, qty)\)
- Proposed SQL extension:
  
  ```sql
  SELECT SUM(qty) FROM Sale
  GROUP BY CUBE CID, PID, SID;
  ```
- Output contains:
  - Normal groups produced by GROUP BY
    - \((c1, p1, s1, sum), (c1, p2, s3, sum), \ldots\)
  - Groups with one or more ALL’s
    - \((\text{ALL}, p1, s1, \text{sum}), (\text{c2}, \text{ALL}, \text{ALL}, \text{sum}), (\text{ALL}, \text{ALL}, \text{ALL, sum}), \ldots\)
- Can you write a CUBE query using only GROUP BY’s?

Aggregation lattice

A parent can be computed from any child
Materialized views

• Computing GROUP BY and CUBE aggregates is expensive

• OLAP queries perform these operations over and over again

 Idea: precompute and store the aggregates as materialized views

• Maintained automatically as base data changes—how?
• No. 1 user-requested feature in PostgreSQL!
Selecting views to materialize

• Factors in deciding what to materialize
  • What is its storage cost?
  • What is its update cost?
  • Which queries can benefit from it?
  • How much can a query benefit from it?

☞ Analogous to selecting indexes

• Example
  • GROUP BY $\emptyset$ is small, but not useful to most queries
  • GROUP BY $\text{CID, PID, SID}$ is useful to any query, but too large to be beneficial
Summary

• Eagerly integrate data from operational sources and store a redundant copy to support OLAP

• OLAP vs. OLTP: different workload → different degree of redundancy

• CUBE extension to SQL
  • Not covered: other OLAP extensions such as windowing