Data Warehousing

Introduction to Databases
CompSci 316 Fall 2016
Announcements (Thu., Dec. 8)

- **Homework #4** last Gradiance problem due today
  - Sample solution to be posted by this weekend
- **Project demos** to start tomorrow
  - Check your email for schedule
  - **Submit report/code before demo** (you have until next Thursday to update it)
- **Final exam** **Thur. Dec. 15 7-10pm**
  - **Different room:** LSRC B101
  - Open-book, open-notes
  - Comprehensive, but with strong emphasis on the second half of the course
  - **Sample final + solution** posted on Sakai
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
- 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

<table>
<thead>
<tr>
<th>Eager (warehousing)</th>
<th>Lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td>• In advance: before queries</td>
<td>• On demand: at query time</td>
</tr>
<tr>
<td>• Copy data from sources</td>
<td>• Leave data at sources</td>
</tr>
<tr>
<td>📬 Answer could be stale</td>
<td>📬 Answer is more up-to-date</td>
</tr>
<tr>
<td>🚫 Need to maintain consistency</td>
<td>🚫 No need to maintain consistency</td>
</tr>
<tr>
<td>🚫 Query processing is local to the warehouse</td>
<td>🚫 Sources participate in query processing</td>
</tr>
<tr>
<td>• Faster</td>
<td>• Slower</td>
</tr>
<tr>
<td>• Can operate when sources are unavailable</td>
<td>• Interferes with local processing</td>
</tr>
<tr>
<td></td>
<td>• Still has consistency issues</td>
</tr>
</tbody>
</table>
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>Product</th>
<th>Dimension table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>name</td>
</tr>
<tr>
<td>p1</td>
<td>beer</td>
</tr>
<tr>
<td>p2</td>
<td>diaper</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sale</th>
<th>Fact table</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>Date</td>
</tr>
<tr>
<td>100</td>
<td>08/23/2015</td>
</tr>
<tr>
<td>102</td>
<td>09/12/2015</td>
</tr>
<tr>
<td>105</td>
<td>09/24/2015</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Dimension table</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>name</td>
</tr>
<tr>
<td>c3</td>
<td>Amy</td>
</tr>
<tr>
<td>c4</td>
<td>Ben</td>
</tr>
<tr>
<td>c5</td>
<td>Coy</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

- **Small**
- Updated infrequently
Data cube

Simplified schema: Sale (CID, PID, SID, qty)

(c3, p2, s1) = 2
(c5, p1, s3) = 5
(c3, p1, s1) = 1
(c5, p1, s1) = 3
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

```sql
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, p1, s1) = 4
(ALL, p2, s1) = 2
(ALL, p2, s1) = 2
(ALL, p1, s3) = 5
(c3, p1, s1) = 1
(c3, p1, s1) = 2
(c5, p1, s1) = 3
(c5, p1, s3) = 5

(ALL, p1, ALL) = 9
(ALL, p2, ALL) = 2
(ALL, ALL, ALL) = 11

Customer

Product

Store

“ALL”
CUBE operator

• Sale (CID, PID, SID, qty)
• Proposed SQL extension:
  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), etc.
  • Groups with one or more ALL’s
    • (ALL, p1, s1, sum), (c2, ALL, ALL, sum), (ALL, ALL, ALL, sum), etc.
• Can you write a CUBE query using only GROUP BY’s?

Aggregation lattice

GROUP BY $\emptyset$

GROUP BY CID

GROUP BY PID

GROUP BY SID

GROUP BY CID, PID

GROUP BY CID, SID

GROUP BY PID, SID

GROUP BY CID, PID, SID

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
  • 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?

• 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
Other OLAP extensions

• Besides extended grouping capabilities (e.g., CUBE), window operations have also been added to SQL

• A “window” specifies an ordered list of rows related to the “current row”

• A window function computes a value from this list and the “current row”
  • Standard aggregates: COUNT, SUM, AVG, MIN, MAX
  • New functions: RANK, PERCENT_RANK, LAG, LEAD, …
RANK window function example

```sql
SELECT SID, PID, SUM(qty), RANK() OVER w FROM Sale GROUP BY SID, PID
WINDOW w AS (PARTITION BY SID ORDER BY SUM(qty) DESC);
```

Applying the `RANK()` function after processing `FROM`, `WHERE`, `GROUP BY`, `HAVING`

- PARTITION defines the related set and ORDER BY orders it

### Example

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>Bob</td>
<td>100</td>
</tr>
</tbody>
</table>

E.g., for the following “row,”

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Bob</td>
<td>2</td>
</tr>
</tbody>
</table>

the related list is:

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>
RANK example (cont’d)

```sql
SELECT SID, PID, SUM(qty),
       RANK() OVER w
FROM Sale GROUP BY SID, PID
WINDOW w AS
  (PARTITION BY SID
   ORDER BY SUM(qty) DESC);
```

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>Bob</td>
<td>100</td>
</tr>
</tbody>
</table>

E.g., for the following “row,”

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Bob</td>
<td>2</td>
</tr>
</tbody>
</table>

Then, for each “row” and its related list, evaluate SELECT and return:

```sql
SELECT SID, PID, SUM(qty),
       RANK() OVER w
FROM Sale GROUP BY SID, PID
WINDOW w AS
  (PARTITION BY SID
   ORDER BY SUM(qty) DESC);
```

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>sum</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
### Multiple windows

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>cid</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>Alice</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>Bob</td>
<td>100</td>
</tr>
</tbody>
</table>

```sql
SELECT SID, PID, SUM(qty),
    RANK() OVER w,
    RANK() OVER w1 AS rank1
FROM Sale GROUP BY SID, PID
WINDOW w AS
    (PARTITION BY SID
        ORDER BY SUM(qty) DESC),
w1 AS
    (ORDER BY SUM(qty) DESC)
ORDER BY SID, rank;
```

No PARTITION means all “rows” are related to the current one

So rank1 is the “global” rank:

<table>
<thead>
<tr>
<th>sid</th>
<th>pid</th>
<th>sum</th>
<th>rank</th>
<th>rank1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durham</td>
<td>beer</td>
<td>12</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Durham</td>
<td>chips</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Summary

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

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

• SQL extensions: grouping and windowing