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 Thu. 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

<table>
<thead>
<tr>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly updates</td>
<td>Mostly reads</td>
</tr>
<tr>
<td>Short, simple transactions</td>
<td>Long, complex queries</td>
</tr>
<tr>
<td>Clerical users</td>
<td>Analysts, decision makers</td>
</tr>
<tr>
<td>Goal: transaction throughput</td>
<td>Goal: fast queries</td>
</tr>
</tbody>
</table>

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

- **Fact table**
  - Big
  - Constantly growing
  - Stores measures (often aggregated in queries)
  - Stores small, updated infrequently

- **Dimension table**
  - Small

- **Data cube**
  - Simplified schema: Sale (CID, PID, SID, qty)
  - Total quantity of sales for each product in each store

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

- Completing the cube—plane
  - 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

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

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 is small, but not useful to most queries
  - GROUP BY 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

```
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);
```

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

```
sid | pid | sum | rank
-----------------------
Durham | beer | 14 | 1
Durham | diaper | 5 | 2
Durham | chips | 3 | 3
Raleigh | diaper | 100 | 1
Raleigh | beer | 1 | 2
```
Multiple windows

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

So rank1 is the “global” rank:

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