Announcements (December 4)

- Homework #4 will be graded over this weekend
  - Sample solution available on Thursday
- Remember your project demo slot!
- Final exam next Saturday, Dec. 15, 7-10pm
  - Again, open book, open notes
  - Focus on the second half of the course
  - Sample final available now
  - Sample final solution available Thursday
- Wrap-up and review on Thursday

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: ACID, transaction throughput</td>
<td>Goal: fast queries</td>
</tr>
</tbody>
</table>

Implications on database design and optimization?

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>
</tbody>
</table>

Maintaining a data warehouse

- The “ETL” process
  - Extraction: extract relevant data and/or changes from sources
  - Transformation: transform data to match the warehouse schema
  - Loading: integrate data/changes into the warehouse

- Approaches
  - Recomputation
    - Easy to implement; just take periodic dumps of the sources, say, every night
  - Incremental maintenance
    - Compute and apply only incremental changes
"Star" schema of a data warehouse

<table>
<thead>
<tr>
<th>Product</th>
<th>Store</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>OID</td>
<td>CID</td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>s1</td>
</tr>
<tr>
<td>p2</td>
<td>c3</td>
<td>s2</td>
</tr>
<tr>
<td>p3</td>
<td>c4</td>
<td>s3</td>
</tr>
</tbody>
</table>

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

Dimension tables
- Small
- Updated infrequently

Data cube

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

<table>
<thead>
<tr>
<th>Sale</th>
<th>Product</th>
<th>Store</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>SID</td>
<td>CID</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c4</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>c5</td>
<td>s1</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Sale</th>
<th>Product</th>
<th>Store</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>SID</td>
<td>CID</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c3</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>c4</td>
<td>s1</td>
<td></td>
</tr>
</tbody>
</table>

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

(ALL, p1, s3) = 5
(ALL, c3, p1, s3) = 2
(ALL, c4, p1, s3) = 4
(ALL, c5, p1, s3) = 3
(6, p2, s1) = 2
(6, p3, s1) = 1
(6, p1, s3) = 5
(6, p1, s1) = 3
(6, p1, s3) = 5
Completing the cube—axis

Total quantity of sales for each product

\[
\text{SELECT PID, SUM(qty) FROM Sale GROUP BY PID;}
\]

Completion points onto the Product axis

\[
(\text{ALL, p1, s3}) = 5 \quad (\text{c5, p1, s1}) = 3
\]

\[
(\text{ALL, p2, s1}) = 2 \quad (\text{c3, p1, s1}) = 1
\]

\[
(\text{ALL, p1, ALL}) = 9 \quad (\text{c5, p1, s1}) = 3
\]

Completing the cube—origin

Total quantity of sales

\[
\text{SELECT SUM(qty) FROM Sale;}
\]

Completion points onto the origin

\[
(\text{ALL, p1, s3}) = 5 \quad (\text{c5, p1, s1}) = 3
\]

\[
(\text{ALL, p2, s1}) = 2 \quad (\text{c3, p1, s1}) = 1
\]

\[
(\text{ALL, p1, ALL}) = 9 \quad (\text{c5, p1, s1}) = 3
\]

CUBE operator

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

Gray et al., "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total." ICDE 1996
Automatic summary tables

- Computing GROUP BY and CUBE aggregates is expensive
- OLAP queries perform these operations over and over again

- Idea: precompute and store the aggregates as automatic summary tables (a DB2 term)
  - Maintained automatically as base data changes
  - Same as materialized views

Aggregation view lattice

- Roll up
- Drill down

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

Harinarayan et al., "Implementing Data Cubes Efficiently." SIGMOD 1996
Data mining
- Data → knowledge
- DBMS meets AI and statistics
- Clustering, prediction (classification and regression), association analysis, outlier analysis, evolution analysis, etc.
  - Usually complex statistical “queries” that are difficult to answer → often specialized algorithms outside DBMS
- We will focus on frequent itemset mining

Mining frequent itemsets
- Given: a large database of transactions, each containing a set of items
  - Example: market baskets
- Find all frequent itemsets
  - A set of items $X$ is frequent if no less than $s_{\text{min}}$% of all transactions contain $X$
  - Examples: \{diaper, beer\}, \{scanner, color printer\}

First try
- A naïve algorithm
  - Keep a running count for each possible itemset
  - For each transaction $T$, and for each itemset $X$, if $T$ contains $X$ then increment the count for $X$
  - Return itemsets with large enough counts
- Problem:
The Apriori property

- All subsets of a frequent itemset must also be frequent
  - Because any transaction that contains \( X \) must also contains subsets of \( X \)

- If we have already verified that \( X \) is infrequent, there is no need to count \( X \)'s supersets because they must be infrequent too

The Apriori algorithm

Multiple passes over the transactions

- Pass \( k \) finds all frequent \( k \)-itemsets (itemset of size \( k \))
- Use the set of frequent \( k \)-itemsets found in pass \( k \) to construct candidate \((k+1)\)-itemsets to be counted in pass \((k+1)\)
  - A \((k+1)\)-itemset is a candidate only if all its subsets of size \( k \) are frequent

Example: pass 1

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>6</td>
</tr>
<tr>
<td>{B}</td>
<td>7</td>
</tr>
<tr>
<td>{C}</td>
<td>6</td>
</tr>
<tr>
<td>{D}</td>
<td>2</td>
</tr>
<tr>
<td>{E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Transactions

\( \text{min} \% = 20\% \)

Frequent 1-itemsets

(Itemset \{F\} is infrequent)
Example: pass 2

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, E</td>
</tr>
<tr>
<td>T002</td>
<td>A, D</td>
</tr>
<tr>
<td>T003</td>
<td>B, C</td>
</tr>
<tr>
<td>T004</td>
<td>A, B, D</td>
</tr>
<tr>
<td>T005</td>
<td>A, C</td>
</tr>
<tr>
<td>T006</td>
<td>B, C</td>
</tr>
<tr>
<td>T007</td>
<td>A, C</td>
</tr>
<tr>
<td>T008</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>T009</td>
<td>A, B, C</td>
</tr>
<tr>
<td>T010</td>
<td>F</td>
</tr>
</tbody>
</table>

Transactions
$m_{\text{min}}\% = 20\%$

Generate candidates
Scan and count
Check min. support

Frequent 1-itemsets

- \{A\} 6
- \{B\} 7
- \{C\} 6
- \{D\} 2
- \{E\} 2

Frequent 2-itemsets

- \{A, B\} 4
- \{A, C\} 4
- \{A, D\} 1
- \{A, E\} 2
- \{B, C\} 4
- \{B, D\} 2
- \{B, E\} 2
- \{C, D\} 2
- \{C, E\} 1
- \{D, E\} 0

Candidate 2-itemsets

Example: pass 3

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, E</td>
</tr>
<tr>
<td>T002</td>
<td>A, D</td>
</tr>
<tr>
<td>T003</td>
<td>B, C</td>
</tr>
<tr>
<td>T004</td>
<td>A, B, D</td>
</tr>
<tr>
<td>T005</td>
<td>A, C</td>
</tr>
<tr>
<td>T006</td>
<td>B, C</td>
</tr>
<tr>
<td>T007</td>
<td>A, C</td>
</tr>
<tr>
<td>T008</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>T009</td>
<td>A, B, C</td>
</tr>
<tr>
<td>T010</td>
<td>F</td>
</tr>
</tbody>
</table>

Transactions
$m_{\text{min}}\% = 20\%$

Generate candidates
Scan and count
Check min. support

Frequent 1-itemsets

- \{A\} 4
- \{B\} 4
- \{C\} 4
- \{D\} 2
- \{E\} 2

Frequent 2-itemsets

- \{A, B\} 4
- \{A, C\} 4
- \{A, D\} 2
- \{A, E\} 2
- \{B, C\} 4
- \{B, D\} 2
- \{B, E\} 2
- \{C, D\} 2
- \{C, E\} 1
- \{D, E\} 0

Candidate 3-itemsets

Example: pass 4

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, E</td>
</tr>
<tr>
<td>T002</td>
<td>A, D</td>
</tr>
<tr>
<td>T003</td>
<td>B, C</td>
</tr>
<tr>
<td>T004</td>
<td>A, B, D</td>
</tr>
<tr>
<td>T005</td>
<td>A, C</td>
</tr>
<tr>
<td>T006</td>
<td>B, C</td>
</tr>
<tr>
<td>T007</td>
<td>A, C</td>
</tr>
<tr>
<td>T008</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>T009</td>
<td>A, B, C</td>
</tr>
<tr>
<td>T010</td>
<td>F</td>
</tr>
</tbody>
</table>

Transactions
$m_{\text{min}}\% = 20\%$

Generate candidates

Frequent 1-itemsets

- \{A, B, C\} 2

Frequent 3-itemsets

- \{A, B, E\} 2

Candidate 4-itemsets

- \{A, B, C, E\} 2

No more itemsets to count!
Example: final answer

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>6</td>
</tr>
<tr>
<td>{B}</td>
<td>7</td>
</tr>
<tr>
<td>{C}</td>
<td>6</td>
</tr>
<tr>
<td>{D}</td>
<td>2</td>
</tr>
<tr>
<td>{E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Frequent 1-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B}</td>
<td>4</td>
</tr>
<tr>
<td>{A,C}</td>
<td>4</td>
</tr>
<tr>
<td>{A,E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Frequent 2-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>2</td>
</tr>
<tr>
<td>{A,B,E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Frequent 3-itemsets

Summary

- **Data warehousing**
  - Eagerly integrate data from operational sources and store a redundant copy to support OLAP
  - OLAP vs. OLTP: different workload $\rightarrow$ different degree of redundancy

- **Data mining**
  - Only covered frequent itemset counting
  - Skipped many other techniques (clustering, classification, regression, etc.)
  - One key difference from statistics and machine learning: massive datasets and I/O-efficient algorithms