Data Warehousing and Data Mining

Introduction to Databases
CompSci 316 Fall 2017
Announcements (Tue., Dec. 5)

• **Homework #4** due today
  • Sample solution to be posted by this weekend

• **Project demos** to start Thursday
  • Check your email for schedule
  • Submit report/code before demo

• **Final exam** Sat. Dec. 16 2-5pm
  • **This room**
  • 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

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
- **Product**
  - | PID | name | cost |
  - | p1  | beer | 10   |
  - | p2  | diaper | 16   |
  - ...

- **Customer**
  - | CID | name | address       | city   |
  - | c3  | Amy  | 100 Main St. | Durham |
  - | c4  | Ben  | 102 Main St. | Durham |
  - | c5  | Coy  | 800 Eighth St. | Durham |
  - ...

- **Store**
  - | SID | city |
  - | s1  | Durham |
  - | s2  | Chapel Hill |
  - | s3  | RTP |
  - ...

- **Fact table**
  - | OID | Date       | CID | PID | SID | qty | price |
  - | 100 | 08/23/2017 | c3  | p1  | s1  | 1   | 12    |
  - | 102 | 09/12/2017 | c3  | p2  | s1  | 2   | 17    |
  - | 105 | 09/24/2017 | c5  | p1  | s3  | 5   | 13    |
  - ...

#### Notes
- **Dimension table**
  - Small
  - Updated infrequently

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

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

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

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

“ALL”

“ALL”

Product

Customer

Store

Total quantity of sales

SELECT SUM(qty) FROM Sale;
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

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 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);
```

E.g., for the following “row,” the related list is:

```
Durham | beer | Alice | 10
Durham | beer | Bob   | 2
```

Apply WINDOW after processing
FROM, WHERE, GROUP BY, HAVING

- PARTITION defines the related set and ORDER BY orders it
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></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>

E.g., for the following “row,”

```sql
Durham beer Alice 10
Durham beer Bob  2
Durham diaper Alice 3
```

the related list is:

```sql
Durham beer Alice 10
Durham beer Bob  2
Durham diaper Alice 5
```

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

```sql
<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>Durham</td>
<td>chips</td>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Durham</td>
<td>diaper</td>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>Raleigh</td>
<td>beer</td>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Raleigh</td>
<td>diaper</td>
<td>Bob</td>
<td>2</td>
</tr>
</tbody>
</table>

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:
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
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, as a sample problem in data 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 $s_{min}\%$ of all transactions contain $X$
  • Examples: \{diaper, beer\}, \{scanner, color printer\}

<table>
<thead>
<tr>
<th>TID</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>diaper, milk, candy</td>
</tr>
<tr>
<td>T002</td>
<td>milk, egg</td>
</tr>
<tr>
<td>T003</td>
<td>milk, beer</td>
</tr>
<tr>
<td>T004</td>
<td>diaper, milk, egg</td>
</tr>
<tr>
<td>T005</td>
<td>diaper, beer</td>
</tr>
<tr>
<td>T006</td>
<td>milk, beer</td>
</tr>
<tr>
<td>T007</td>
<td>diaper, beer</td>
</tr>
<tr>
<td>T008</td>
<td>diaper, milk, beer, candy</td>
</tr>
<tr>
<td>T009</td>
<td>diaper, milk, beer</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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 number of itemsets is huge!
  • $2^n$, where $n$ is the number of items

• Think: How do we prune the search space?
The Apriori property

- All subsets of a frequent itemset must also be frequent
  - Because any transaction that contains $X$ must also contain subsets of $X$

_popup
- 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 (i.e., itemsets 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>TID</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, E</td>
</tr>
<tr>
<td>T002</td>
<td>B, 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

\[ s_{min} \% = 20\% \]

Frequent 1-itemsets

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

Transactions

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

<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>B, 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>

Scan and count

Check min. support

<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,D}</td>
<td>1</td>
</tr>
<tr>
<td>{A,E}</td>
<td>2</td>
</tr>
<tr>
<td>{B,C}</td>
<td>4</td>
</tr>
<tr>
<td>{B,D}</td>
<td>2</td>
</tr>
<tr>
<td>{B,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}</td>
<td>4</td>
</tr>
<tr>
<td>{A,C}</td>
<td>4</td>
</tr>
<tr>
<td>{A,E}</td>
<td>2</td>
</tr>
<tr>
<td>{B,C}</td>
<td>4</td>
</tr>
<tr>
<td>{B,D}</td>
<td>2</td>
</tr>
<tr>
<td>{B,E}</td>
<td>2</td>
</tr>
</tbody>
</table>
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>B, 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

\[ s_{\text{min}}\% = 20\% \]

Generate candidates

Scan and count

Check min. support

Frequent 2-itemsets

Candidate 3-itemsets

Frequent 3-itemsets

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{count} \\
\hline
\{A,B\} & 4 \\
\{A,C\} & 4 \\
\{A,E\} & 2 \\
\{B,C\} & 4 \\
\{B,D\} & 2 \\
\{B,E\} & 2 \\
\hline
\end{array}
\]
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>B, 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

$s_{min}\% = 20\%$

Generate candidates

Candidate 4-itemsets

Frequent 3-itemsets

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>
<tr>
<td>{B,C}</td>
<td>4</td>
</tr>
<tr>
<td>{B,D}</td>
<td>2</td>
</tr>
<tr>
<td>{B,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

• Only covered frequent itemset counting
• Skipped many other techniques (clustering, classification, regression, etc.)
• Compared with statistics and machine learning: more focus on massive datasets and I/O-efficient algorithms