Data Warehousing & Mining

CPS 116
Introduction to Database Systems

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: ACID, transaction throughput

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

Implications on database design and optimization?
- "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
- 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
- Slower
- Interferes with local processing

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 and 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
    - Compare 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

**Product**
- Small
- Updated infrequently

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

**Dimension table**
- Small
- Updated infrequently

**Sale**
- Detailed record
- Latest transactions

**Customer**
- Large, detailed record
- Includes everything

**Fact table**
- Product
- Store
- Date
- SKU
- Quantity
- Discount
- Price

**Dimension table**
- Product
- Store
- Date

**Dimension table**
- Product
- Store
- Date

**Product**
- SKU
- Description
- Department

**Store**
- Store ID
- Store Name
- Address
- City
- State
- Zip

**Date**
- Date
- Year
- Month
- Day

**Sale**
- Sale ID
- Product ID
- Store ID
- Date
- Quantity
- Discount
- Price

**Customer**
- Customer ID
- Name
- Address
- City
- State
- Zip
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;

Completing the cube—axis

Total quantity of sales for each product

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

Completing the cube—origin

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), (ALL, ALL, sum), (ALL, ALL, ALL, sum), etc.
- Can you write a CUBE query using only GROUP BY’s?

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

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

- Roll up
- Drill down

GROUP BY ⊘
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

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 $s\%$ 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 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 $X$
- 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>ID</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>a, b, e</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>b, c</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>a, b, d</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>a, b, e, f</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>a, b, c</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>b, c</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td>b, c, e, f</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>a, b, c, d</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

Transactions
$\min = 20\%$

Frequent 1-itemsets
- a 6
- b 7
- c 6
- d 2
- e 2
- f

Candidate 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} 4
- {c, e} 2
- {d, e} 2

Example: pass 2

Generate candidates
Scan and count
Check min. support
Frequent 1-itemsets
- a 6
- b 7
- c 6
- d 2
- e 2

Candidate 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} 4
- {c, e} 2
- {d, e} 2

Example: pass 3

Generate candidates
Scan and count
Check min. support
Frequent 1-itemsets
- a 6
- b 7
- c 6
- d 2
- e 2

Example: pass 4

Generate candidates
Scan and count
Check min. support
Frequent 4-itemsets
- {a, b, c, e} 2

Example: final answer

<table>
<thead>
<tr>
<th>ID</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>a, b, e</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>b, c</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>a, b, d</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>a, b, e, f</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>a, c</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>a, b, c</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>b, c</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td>b, c, e, f</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>a, b, c, d</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

Transactions
$\min = 20\%$

Frequent 1-itemsets
- a 6
- b 7
- c 6
- e 2
- f

Frequent 2-itemsets
- {a, b} 4
- {a, c} 4
- {a, e} 2
- {b, c} 4
- {b, d} 2
- {b, e} 2
- {c, d} 4
- {c, e} 2
- {d, e} 2