## Data Warehousing and Data Mining

**CPS 116**
Introduction to Database Systems

### Announcements (December 1)
- Homework #4 due today
  - Sample solution available Thursday
- Course project demo period has begun!
  - Check email for your scheduled slot
  - Check course website for what to submit
- Final exam next Tuesday, Dec. 8, 9am-12pm
  - Again, open book, open notes
  - Focus on the second half of the course
  - Sample final (from last year) available today
  - Sample final solution available Thursday
- Course evaluations

### 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?
- 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

### 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
    - 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>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>Beer</td>
</tr>
<tr>
<td>2</td>
<td>Diaper</td>
</tr>
</tbody>
</table>

**Fact table**

<table>
<thead>
<tr>
<th>Product</th>
<th>Store</th>
<th>Sale</th>
<th>CID</th>
<th>PID</th>
<th>SID</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beer</td>
<td>Store</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Diaper</td>
<td>Store</td>
<td>12</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

**Dimension table**

<table>
<thead>
<tr>
<th>Customer</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>Amy</td>
</tr>
<tr>
<td>2</td>
<td>Ben</td>
</tr>
<tr>
<td>3</td>
<td>Coy</td>
</tr>
</tbody>
</table>

**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
      ```
      (6, p2, s1) = 2
      ```
    - Groups with one or more ALL's
      ```
      (6, ALL, ALL) = 11
      ```
    - 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
  - Called automatic summary tables in DB2

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 $\min\%$ 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 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: 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>
<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 1-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 2-itemsets

Summary

- Data warehousing
  - Eagerly integrate data from operational sources and store a redundant copy to support OLAP
  - OLAP vs. OLTP: different workload → 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