CS 4604: Introduction to Database Management Systems

Data Mining and Warehousing

Virginia Tech CS 4604 Sprint 2021
Instructor: Yinlin Chen
Today’s Topics

• OLAP: Online Analytical Processing
• Data Mining
• Cloud Database
Introduction

Traditional database systems are tuned to many, small, simple queries. New applications use fewer, more time-consuming, *analytic* queries. New architectures have been developed to handle analytic queries efficiently.
Problem

Given: multiple data sources
Find: patterns (classifiers, rules, clusters, outliers...)

- sales(p-id, c-id, date, $price)
- customers(c-id, age, income, ...)

NY

SF

BBURG
Data Ware-housing

**Step 1:** collect the data, in a single place (= Data Warehouse)

How?

A: Triggers/Materialized views

How often?

A: Depends

How about discrepancies / non-homegeneities?

A: Wrappers/Mediators

**Step 2:** collect counts. (DataCubes/OLAP)
The Data Warehouse

The most common form of data integration.

– Copy sources into a single DB (*warehouse*) and try to keep it up-to-date.

– Usual method: periodic reconstruction of the warehouse, perhaps overnight.

– Frequently essential for analytic queries.
OLTP

Most database operations involve *On-Line Transaction Processing* (OTLP).

– Short, simple, frequent queries and/or modifications, each involving a small number of tuples.

– Examples: Answering queries from a Web interface, sales at cash registers, selling airline tickets.
OLAP

*On-Line Application Processing* (OLAP, or “analytic”) queries are, typically:

– Few, but complex queries --- may run for hours.
– Queries do not depend on having an absolutely up-to-date database.
OLAP Examples

1. Amazon analyzes purchases by its customers to come up with an individual screen with products of likely interest to the customer.

2. Analysts at Wal-Mart look for items with increasing sales in some region.
   - Use empty trucks to move merchandise between stores.
Common Architecture

Databases at store branches handle OLTP.
Local store databases copied to a central warehouse overnight.
Analysts use the warehouse for OLAP.
Star Schemas

A *star schema* is a common organization for data at a warehouse. It consists of:

- *Fact table*: a very large accumulation of facts such as sales.
  - Often “insert-only.”
- *Dimension tables*: smaller, generally static information about the entities involved in the facts.
Example: Star Schema

Suppose we want to record in a warehouse information about every beer sale: the bar, the brand of beer, the drinker who bought the beer, the day, the time, and the price charged. The fact table is a relation:
Sales(bar, beer, drinker, day, time, price)
Example -- Continued

The dimension tables include information about the bar, beer, and drinker “dimensions”:

Bars(bar, addr, license)
Beers(beer, manf)
Drinkers(drinker, addr, phone)
Visualization – Star Schema

- Dimension Table (Bars)
- Dimension Table (Drinkers)
- Dimension Table (Beers)
- Dimension Table (etc.)
- Fact Table - Sales
- Dimension Attrs.
- Dependent Attrs.
Dimensions and Dependent Attributes

Two classes of fact-table attributes:

1. *Dimension attributes*: the key of a dimension table.
2. *Dependent attributes*: a value determined by the dimension attributes of the tuple.
Example: Dependent Attribute

price is the dependent attribute of our example Sales relation. It is determined by the combination of dimension attributes: bar, beer, drinker, and the time (combination of day and time-of-day attributes).
Approaches to Building Warehouses

1. **ROLAP** = “relational OLAP”: Tune a relational DBMS to support star schemas.

2. **MOLAP** = “multidimensional OLAP”: Use a specialized DBMS with a model such as the “data cube.”
ROLAP Techniques

1. **Bitmap indexes**: For each key value of a dimension table (e.g., each beer for relation Beers) create a bit-vector telling which tuples of the fact table have that value.

2. **Materialized views**: Store the answers to several useful queries (views) in the warehouse itself.
Typical OLAP Queries

Often, OLAP queries begin with a “star join”: the natural join of the fact table with all or most of the dimension tables.

Example:

```sql
SELECT * 
FROM Sales, Bars, Beers, Drinkers 
WHERE Sales.bar = Bars.bar AND 
  Sales.beer = Beers.beer AND 
  Sales.drinker = Drinkers.drinker;
```
Typical OLAP Queries --- (2)

The typical OLAP query will:
1. Start with a star join.
2. Select for interesting tuples, based on dimension data.
3. Group by one or more dimensions.
4. Aggregate certain attributes of the result.
Example: OLAP Query

For each bar in Blacksburg, find the total sale of each beer manufactured by Anheuser-Busch.
Filter: addr = “Blacksburg” and manf = “Anheuser-Busch”.
Grouping: by bar and beer.
Aggregation: Sum of price.
Example: In SQL

SELECT bar, beer, SUM(price)
FROM Sales NATURAL JOIN Bars
   NATURAL JOIN Beers
WHERE addr = 'Blacksburg' AND
   manf = 'Anheuser-Busch'
GROUP BY bar, beer;
Using Materialized Views

A direct execution of this query from Sales and the dimension tables could take too long.
If we create a materialized view that contains enough information, we may be able to answer our query much faster.
Example: Materialized View

Which views could help with our query?

Key issues:
1. It must join Sales, Bars, and Beers, at least.
2. It must group by at least bar and beer.
3. It must not select out Blacksburg bars or Anheuser-Busch beers.
4. It must not project out addr or manf.
Example --- Continued

Here is a materialized view that could help:

```
CREATE VIEW BABMS(bar, addr, beer, manf, sales) AS
SELECT bar, addr, beer, manf,
       SUM(price) sales
FROM Sales NATURAL JOIN Bars
    NATURAL JOIN Beers
GROUP BY bar, addr, beer, manf;
```

Since bar -> addr and beer -> manf, there is no real grouping. We need addr and manf in the SELECT.
Example --- Concluded

Here’s our query using the materialized view BABMS:

```sql
SELECT bar, beer, sales 
FROM BABMS 
WHERE addr = 'Blacksburg' AND 
manf = 'Anheuser-Busch';
```
MOLAP and Data Cubes

Keys of dimension tables are the dimensions of a hypercube. Example:
Sales(bar, beer, drinker, time, price)
– for the Sales data, the four dimensions are bar, beer, drinker, and time. Dependent attributes (e.g., price) appear at the points of the cube.
Visualization -- Data Cubes

beer

price

bar

drinker
The data cube also includes aggregation (typically SUM) along the margins of the cube. The *marginals* include aggregations over one dimension, two dimensions,...
Visualization --- Data Cube w/Aggregation

SUM over all Drinkers
Example: Marginals

Our 4-dimensional Sales cube includes the sum of price over each bar, each beer, each drinker, and each time unit (perhaps days).

It would also have the sum of price over all bar-beer pairs, all bar-drinker-day triples,...
Marginals

Think of each dimension as having an additional value *. A point with one or more *’s in its coordinates aggregates over the dimensions with the *’s.
Example: ("Joe’s Bar", "Bud", *, *) holds the sum, over all drinkers and all time, of the Bud consumed at Joe’s.
Drill-Down

*Drill-down = “de-aggregate” = break an aggregate into its constituents.*

Example: having determined that Joe’s Bar sells very few Anheuser-Busch beers, break down his sales by particular A.-B. beer.
Roll-Up

Roll-up = aggregate along one or more dimensions.
Example: given a table of how much Bud each drinker consumes at each bar, roll it up into a table giving total amount of Bud consumed by each drinker.
**Example: Roll Up and Drill Down**

**$ of Anheuser-Busch by drinker/bar**

<table>
<thead>
<tr>
<th></th>
<th>Jim</th>
<th>Bob</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's Bar</td>
<td>45</td>
<td>33</td>
<td>30</td>
</tr>
<tr>
<td>Bull &amp; Bones</td>
<td>50</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>Blue Chalk</td>
<td>38</td>
<td>31</td>
<td>40</td>
</tr>
</tbody>
</table>

**$ of A-B / drinker**

<table>
<thead>
<tr>
<th></th>
<th>Jim</th>
<th>Bob</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim</td>
<td>133</td>
<td>100</td>
<td>112</td>
</tr>
</tbody>
</table>

Roll up by Bar

**$ of A-B Beers / drinker**

<table>
<thead>
<tr>
<th></th>
<th>Jim</th>
<th>Bob</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>40</td>
<td>29</td>
<td>40</td>
</tr>
<tr>
<td>M'lob</td>
<td>45</td>
<td>31</td>
<td>37</td>
</tr>
<tr>
<td>Bud Light</td>
<td>48</td>
<td>40</td>
<td>35</td>
</tr>
</tbody>
</table>

Drill down by Beer
Structure of the Data Cube

CUBE(F) of fact table F is *roughly* === the Fact table (F) +
aggregations across all dimensions (i.e. marginals)
– Note CUBE(F) is a relation itself!
CUBE in SQL: Example

For our Sales example:

Sales(bar, beer, drinker, time, price)

CREATE MATERIALIZED VIEW SalesCube AS
SELECT bar, beer, drinker, time, SUM(price)
FROM Sales
GROUP BY bar, beer, drinker, time WITH CUBE;
Tuples in SalesCube

Tuples implied by the standard GROUP-BY:
(Joes, Bud, John, 4/19/13, 3.00)

And those tuples of that are constructed by rolling-up the dimensions in GROUP-BY (== marginals, NULL == *). E.g:
(Joes, NULL, John, 4/19/13, 10.00)
(Joes, NULL, John, NULL, 200.00)
(Joes, NULL, NULL, NULL, 200000.00)
(NULL, NULL, NULL, NULL, 2000000.00)
Tuples in SalesCube

Tuples implied by the standard GROUP-BY:
(Joes, Bud, John, 4/19/13, 3.00)

And those tuples of that are constructed by rolling-up the dimensions in GROUP-BY (= marginals, NULL == *). E.g:
(Joes, NULL, John, 4/19/13, 10.00)
(Joes, NULL, John, NULL, 200.00)
(Joes, NULL, NULL, NULL, 200000.00)
(NULL, NULL, NULL, NULL, 2000000.00)

Total spent by John at Joes on Apr 19.
Tuples in SalesCube

Tuples implied by the standard GROUP-BY:
(Joes, Bud, John, 4/19/13, 3.00)

And those tuples of that are constructed by rolling-up the dimensions in GROUP-BY (== marginals, NULL == *). E.g:
(Joes, NULL, John, 4/19/13, 10.00)
(Joes, NULL, John, NULL, 200.00)
(Joes, NULL, NULL, NULL, 200000.00)
(NULL, NULL, NULL, NULL, 2000000.00)

Total spent by John at Joes ever.
Tuples implied by the standard GROUP-BY:
(Joes, Bud, John, 4/19/13, 3.00)

And those tuples of that are constructed by rolling-up the dimensions in GROUP-BY (== marginals, NULL == *). E.g:
(Joes, NULL, John, 4/19/13, 10.00)
(Joes, NULL, John, NULL, 200.00)
(Joes, NULL, NULL, NULL, 200000.00)
(NULL, NULL, NULL, NULL, 2000000.00)

Total spent by everyone at Joes ever.
**Tuples in SalesCube**

Tuples implied by the standard GROUP-BY:
(Joes, Bud, John, 4/19/13, 3.00)

*And* those tuples of that are constructed by rolling-up the dimensions in GROUP-BY (== marginals, NULL == *). E.g:
(Joes, NULL, John, 4/19/13, 10.00) Total spent by everyone at every bar ever.
(Joes, NULL, John, NULL, 200.00)
(Joes, NULL, NULL, NULL, 200000.00)
(NULL, NULL, NULL, NULL, 2000000.00)
## Compare ROLAP vs MOLAP

### ROLAP Solution

```sql
CREATE VIEW BABMS(bar, addr,
    beer, manf, sales) AS
SELECT bar, addr, beer, manf,
    SUM(price) sales
FROM Sales NATURAL JOIN Bars
    NATURAL JOIN Beers
GROUP BY bar, addr, beer, manf;
```

A specific view for a specific type of query (note the join)

### MOLAP (Data Cube) Solution

```sql
CREATE MATERIALIZED VIEW SalesCube
AS
SELECT bar, beer, drinker, time,
    SUM(price)
FROM Sales
GROUP BY bar, beer, drinker, time WITH CUBE;
```

A generalized view which stores marginals as well (no join)
Data Mining

*Data mining* is a popular term for techniques to summarize big data sets in useful ways.

Examples:

1. Clustering all Web pages by topic.
2. Finding characteristics of fraudulent credit-card use.
### Supervised Learning: Decision Trees: Problem

<table>
<thead>
<tr>
<th>Age</th>
<th>Chol-level</th>
<th>Gender</th>
<th>CLASS-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>150</td>
<td>M</td>
<td>+</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Has heart disease
Supervised Learning: Decision Trees: Problem

<table>
<thead>
<tr>
<th>Age</th>
<th>Chol-level</th>
<th>Gender</th>
<th>…</th>
<th>CLASS-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>150</td>
<td>M</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

What is the label for this new patient?
## Supervised Learning: Decision Trees: Problem

What is the label for this new patient?

<table>
<thead>
<tr>
<th>Age</th>
<th>Chol-level</th>
<th>Gender</th>
<th>CLASS-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>150</td>
<td>M</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>90</td>
<td>F</td>
<td>??</td>
</tr>
</tbody>
</table>

Training Data Set
Decision trees

Pictorially, we have

num. attr#1 (e.g., ‘age’)

num. attr#2 (e.g., chol-level)
Decision trees

and we want to label ‘?’

num. attr#2
(eg., chol-level)

num. attr#1 (eg., ‘age’)

+ - - +
+ - - -
+ - + -
+ - + -
+ - - +
Decision trees

so we build a decision tree:

num. attr#1 (eg., ‘age’)  
um. attr#2 (eg., chol-level)

40

50

num. attr#1 (eg., ‘age’)

Decision trees

so we build a decision tree:

```
  age<50
     /   \
   Y     N
+     -
  
chol. <40
    /   \
   Y     N
     -
     ...
```
Conclusions

Data Mining: of high commercial interest (think BIG data)
DM = DB + Machine Learning + Stats.

Data Warehousing/OLAP: to get the data
Tree Classifiers
Association Rules
..... (like clustering etc.)
Summary

- For OLAP, column-oriented storage trumps row-oriented storage
  - Many tricks beyond splitting columns up
  - compression, late materialization, redundant layouts, efficient write processing
- For OLTP, the costs of many random accesses for updating columns makes a columnar layout not worth it
  - Hence OLTP systems look more traditional, and typically opt for row-oriented storage
- Many systems are now opting for hybrid layouts to try to support both OLAP and OLTP in the same system
Cloud Database

[Diagram showing asynchronous replication from Amazon Aurora Global Database (primary) to Amazon Aurora Global Database (secondary) in AWS Cloud.]

[Diagram showing backup snapshots and cross-region snapshot copies for AWS Cloud, with options for restore during disaster.]
AWS Athena

<table>
<thead>
<tr>
<th>Results</th>
<th>bucket</th>
<th>key</th>
<th>Query editor</th>
<th>Saved queries</th>
<th>History</th>
<th>Data sources</th>
<th>Workgroup</th>
<th>primary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Athena</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>elapsed</th>
<th>stage</th>
<th>files</th>
<th>state</th>
<th>status</th>
<th>restatename</th>
<th>restatereason</th>
<th>algorithm</th>
<th>chunksize</th>
<th>bytestread</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32.1</td>
<td>778</td>
<td></td>
<td>421772</td>
<td></td>
<td>421772</td>
<td></td>
<td></td>
<td>0.7615</td>
<td>0.649</td>
<td>0.582</td>
<td>0.649</td>
</tr>
<tr>
<td>26.9</td>
<td>882</td>
<td></td>
<td>405756</td>
<td></td>
<td>405756</td>
<td></td>
<td></td>
<td>0.3815</td>
<td>0.689</td>
<td>0.603</td>
<td>0.774</td>
</tr>
<tr>
<td>174.4</td>
<td>859</td>
<td></td>
<td>4295900</td>
<td></td>
<td>4295900</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>78.6</td>
<td>235</td>
<td></td>
<td>441776</td>
<td></td>
<td>441776</td>
<td></td>
<td></td>
<td>0.62</td>
<td>0.0</td>
<td>0.56</td>
<td>0.56</td>
</tr>
<tr>
<td>128.5</td>
<td>678</td>
<td></td>
<td>4481240</td>
<td></td>
<td>4481240</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>40.5</td>
<td>705</td>
<td></td>
<td>4148532</td>
<td></td>
<td>4148532</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>152.6</td>
<td>634</td>
<td></td>
<td>1734380</td>
<td></td>
<td>1734380</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.1</td>
<td>605</td>
<td></td>
<td>1939696</td>
<td></td>
<td>1939696</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>84.9</td>
<td>506</td>
<td></td>
<td>4104852</td>
<td></td>
<td>4104852</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>474</td>
<td></td>
<td>3389120</td>
<td></td>
<td>3389120</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.3</td>
<td>809</td>
<td></td>
<td>2134552</td>
<td></td>
<td>2134552</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>139.8</td>
<td>774</td>
<td></td>
<td>483780</td>
<td></td>
<td>483780</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>215.5</td>
<td>1180</td>
<td></td>
<td>4140794</td>
<td></td>
<td>4140794</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>27.7</td>
<td>744</td>
<td></td>
<td>4025452</td>
<td></td>
<td>4025452</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>9.4</td>
<td>1036</td>
<td></td>
<td>4256620</td>
<td></td>
<td>4256620</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>11.1</td>
<td>733</td>
<td></td>
<td>446588</td>
<td></td>
<td>446588</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>320.5</td>
<td>829</td>
<td></td>
<td>4198464</td>
<td></td>
<td>4198464</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>10.0</td>
<td>2390</td>
<td></td>
<td>4944104</td>
<td></td>
<td>4944104</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>39.6</td>
<td>876</td>
<td></td>
<td>4229288</td>
<td></td>
<td>4229288</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>135.5</td>
<td>764</td>
<td></td>
<td>5930804</td>
<td></td>
<td>5930804</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>365.5</td>
<td>693</td>
<td></td>
<td>4348368</td>
<td></td>
<td>4348368</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>219.7</td>
<td>646</td>
<td></td>
<td>4305280</td>
<td></td>
<td>4305280</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
AWS CloudWatch Insight
AWS Glue

1. Discover your data and store associated metadata in the Glue Data Catalog

2. The Glue Data Catalog serves as a central metadata repository

3. Access the Glue Data Catalog for ETL and analytics

4. Read and write metadata to the Glue Data Catalog

5. Run reports

Create and run ELT jobs in AWS Glue

Amazon Redshift

Amazon S3

Amazon RDS

Databases running on Amazon EC2

AWS Glue Data Catalog

Amazon Athena

Amazon Redshift

Amazon EMR

Amazon QuickSight & Other BI Tools
Reading and Next Class

• Data Mining and Warehousing
• Cloud Database
• Next: Final Review