CSE 532 – Theory of Database Systems

Lecture 25 (Chapter 17)
OLAP and Data Mining

Ilchul Yoon
Assistant Professor
State University of New York, Korea

Adapted from book authors’ slides

The Internet Grocer

- Traditional
  - How many cases of tomatoes were sold in all northeast warehouses in the years 2000 and 2001?

- Newer
  - Prepare a profile of the grocery purchases of John Smith for the years 2000 and 2001 (so that we can customize our marketing to him and get more of his business)
Data Mining

- *Data Mining* is an attempt at knowledge discovery – to extract knowledge from a database

- Comparison with OLAP
  - *OLAP:*
    - What percentage of people who make over $50,000 defaulted on their mortgage in the year 2000?
  - *Data Mining:*
    - How can information about salary, net worth, and other historical data be used to predict who will default on their mortgage?

Data Warehouses

- OLAP and data mining databases are frequently stored on special servers called data warehouses:
  - Can accommodate the huge amount of data generated by OLTP systems
  - Allow OLAP queries and data mining to be run off-line so as not to impact the performance of OLTP
OLAP, Data Mining, and Analysis

- The “A” in OLAP stands for “Analytical”
- Many OLAP and Data Mining applications involve sophisticated analysis methods from the fields of mathematics, statistical analysis, and artificial intelligence
- Our main interest is in the database aspects of these fields, not the sophisticated analysis techniques

Fact Tables

- Many OLAP applications are based on a fact table
- For example, a supermarket application might be based on a table
  \[ \text{Sales} \left( \text{Market\_Id, Product\_Id, Time\_Id, Sales\_Amt} \right) \]
- The table can be viewed as multidimensional
  - \( \text{Market\_Id, Product\_Id, Time\_Id} \) are the dimensions that represent specific supermarkets, products, and time intervals
  - \( \text{Sales\_Amt} \) is a function of the other three
### Example Fact Table

<table>
<thead>
<tr>
<th>Sales</th>
<th>Market_Id</th>
<th>Product_Id</th>
<th>Time_Id</th>
<th>Sales_Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>P1</td>
<td>T1</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td>T1</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td>T1</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td>T1</td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td>T1</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td>T1</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td>T1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>T1</td>
<td>3333</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td>T1</td>
<td>8000</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td>T1</td>
<td>8000</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td>T1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td>T1</td>
<td>3300</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P1</td>
<td>T2</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P2</td>
<td>T2</td>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P3</td>
<td>T2</td>
<td>1501</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P4</td>
<td>T2</td>
<td>2501</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>P1</td>
<td>T2</td>
<td>601</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>P2</td>
<td>T2</td>
<td>901</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>P3</td>
<td>T2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>P4</td>
<td>T2</td>
<td>3334</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>P1</td>
<td>T2</td>
<td>6001</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>P2</td>
<td>T2</td>
<td>8001</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>P3</td>
<td>T2</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>P4</td>
<td>T2</td>
<td>3301</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P1</td>
<td>T3</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P2</td>
<td>T3</td>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P3</td>
<td>T3</td>
<td>1503</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P4</td>
<td>T3</td>
<td>2502</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P1</td>
<td>T3</td>
<td>602</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P2</td>
<td>T3</td>
<td>802</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P3</td>
<td>T3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>P4</td>
<td>T3</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>M8</td>
<td>P1</td>
<td>T3</td>
<td>5002</td>
<td></td>
</tr>
<tr>
<td>M8</td>
<td>P2</td>
<td>T3</td>
<td>8002</td>
<td></td>
</tr>
<tr>
<td>M8</td>
<td>P3</td>
<td>T3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>M8</td>
<td>P4</td>
<td>T3</td>
<td>3302</td>
<td></td>
</tr>
</tbody>
</table>

### Dimension Tables

- The dimensions of the fact table are further described with *dimension tables*.

- **Fact table:**

  Sales *(Market_id, Product_Id, Time_Id, Sales_Amt)*

- **Dimension Tables:**

  Market *(Market_Id, City, State, Region)*  
  Product *(Product_Id, Name, Category, Price)*  
  Time *(Time_Id, Week, Month, Quarter)*
A Data Cube

- Fact tables can be viewed as an N-dimensional data cube (3-dimensional in our example)
  - The entries in the cube are the values for Sales_Amts

Star Schema

- The fact and dimension relations can be displayed in an E-R diagram, which looks like a star and is called a star schema
  - Frequently, use a constellation schema
    - Multiple fact tables sharing same dimensions
Aggregation

- Many OLAP queries involve **aggregation** of the data in the fact table.

- For example, to find the total sales (over time) of each product in each market, we might use:

  ```sql
  SELECT S.Product_Id, SUM(S.Sales_Amt) 
  FROM Sales S 
  GROUP BY S.Product_Id 
  ``

- The aggregation is over the entire time dimension and thus produces a two-dimensional view of the data.

- Note: aggregation here is over time, not supermarkets or products.

Aggregation over Time

- The output of the previous query:

<table>
<thead>
<tr>
<th>SUM(Sales_Amt)</th>
<th>Market_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1  M2  M3</td>
</tr>
<tr>
<td>P1</td>
<td>3003 1503 15003</td>
</tr>
<tr>
<td>P2</td>
<td>6003 2402 24003</td>
</tr>
<tr>
<td>P3</td>
<td>4503 3 33</td>
</tr>
<tr>
<td>P4</td>
<td>7503 7000 9903</td>
</tr>
</tbody>
</table>
Drilling Down and Rolling Up

- Some dimension tables form an *aggregation hierarchy*
  
  \[ \text{Market}_\text{Id} \rightarrow \text{City} \rightarrow \text{State} \rightarrow \text{Region} \]

- Executing a series of queries that moves down a hierarchy (e.g., from aggregation over regions to that over states) is called **drilling down**
  - Requires the use of the fact table or information more specific than the requested aggregation (e.g., cities)

- Executing a series of queries that moves up the hierarchy (e.g., from states to regions) is called **rolling up**
  - Note: In a rollup, coarser aggregations can be computed using prior queries for finer aggregations

---

Drilling Down

- **Drilling down on market: from Region to State**

  Sales (\(\text{Market}_\text{Id}, \text{Product}_\text{Id}, \text{Time}_\text{Id}, \text{Sales}_\text{Amt}\))
  Market (\(\text{Market}_\text{Id}, \text{City}, \text{State}, \text{Region}\))

  1. \[
  \text{SELECT} \quad S.\text{Product}_\text{Id}, M.\text{Region}, \text{SUM}(S.\text{Sales}_\text{Amt}) \\
  \text{FROM} \quad \text{Sales} \ S, \text{Market} \ M \\
  \text{WHERE} \quad M.\text{Market}_\text{Id} = S.\text{Market}_\text{Id} \\
  \text{GROUP BY} \quad S.\text{Product}_\text{Id}, M.\text{Region}
  \]

  2. \[
  \text{SELECT} \quad S.\text{Product}_\text{Id}, M.\text{State}, \text{SUM}(S.\text{Sales}_\text{Amt}) \\
  \text{FROM} \quad \text{Sales} \ S, \text{Market} \ M \\
  \text{WHERE} \quad M.\text{Market}_\text{Id} = S.\text{Market}_\text{Id} \\
  \text{GROUP BY} \quad S.\text{Product}_\text{Id}, M.\text{State}
  \]
Rolling Up

- Rolling up on market, from State to Region
  - If we have already created a table, State_Sales, using

1. \[
\text{SELECT } S.\text{Product}_\text{Id}, M.\text{State}, \text{SUM}(S.\text{Sales}_\text{Amt}) \\
\text{FROM } \text{Sales } S, \text{Market } M \\
\text{WHERE } M.\text{Market}_\text{Id} = S.\text{Market}_\text{Id} \\
\text{GROUP BY } S.\text{Product}_\text{Id}, M.\text{State}
\]

then we can roll up from there to:

2. \[
\text{SELECT } T.\text{Product}_\text{Id}, M.\text{Region}, \text{SUM}(T.\text{Sales}_\text{Amt}) \\
\text{FROM } \text{State}_\text{Sales } T, \text{Market } M \\
\text{WHERE } M.\text{State} = T.\text{State} \\
\text{GROUP BY } T.\text{Product}_\text{Id}, M.\text{Region}
\]

*Can reuse the results of query 1.*

Pivoting

- When we view the data as a multi-dimensional cube and group on a subset of the axes, we are said to be performing a *pivot* on those axes
- Pivoting on dimensions \(D_1, \ldots, D_k\) in a data cube \(D_1, \ldots, D_k, D_{k+1}, \ldots, D_n\) means that we use \(\text{GROUP BY } A_1, \ldots, A_k\) and aggregate over \(A_{k+1}, \ldots, A_n\), where \(A_i\) is an attribute of the dimension \(D_i\)
- *Example:* Pivoting on Product and Time corresponds to grouping on \(\text{Product}_\text{Id}\) and \(\text{Quarter}\) and aggregating \(\text{Sales}_\text{Amt}\) over \(\text{Market}_\text{Id}\):

\[
\text{SELECT } S.\text{Product}_\text{Id}, T.\text{Quarter}, \text{SUM}(S.\text{Sales}_\text{Amt}) \\
\text{FROM } \text{Sales } S, \text{Time } T \\
\text{WHERE } T.\text{Time}_\text{Id} = S.\text{Time}_\text{Id} \\
\text{GROUP BY } S.\text{Product}_\text{Id}, T.\text{Quarter}
\]
Time Hierarchy as a Lattice

- Not all aggregation hierarchies are linear
  - The time hierarchy is a lattice
    - Weeks are not contained in months
    - We can roll up days into weeks or months, but we can only roll up weeks into quarters

Slicing-and-Dicing

- When we use WHERE to specify a particular value for an axis (or several axes), we are performing a slice
- Slicing the data cube in the Time dimension (choosing sales only in week 12) then pivoting to Product_id (aggregating over Market_id)

```sql
SELECT S.Product_Id, SUM(Sales_Amt)
FROM Sales S, Time T
WHERE T.Time_Id = S.Time_Id AND T.Week = 'Wk-12'
GROUP BY S.Product_Id
```
Slicing-and-Dicing

- Typically slicing and dicing involves several queries to find the “right slice.”
  For instance, change the slice & the axes (from the prev. example):
  - Slicing on Time and Market dimensions then pivoting to Product_id and Week (in the time dimension)

```sql
SELECT T.Product_Id, T.Week, SUM(Sales_Amt)
FROM Sales S, Time T
WHERE T.Time_Id = S.Time_Id
  AND T.Quarter = 4
  AND S.Market_id = 12345
GROUP BY T.Product_Id, T.Week
```

The CUBE Operator

- To construct the following table, would take 4 queries (next slide)

<table>
<thead>
<tr>
<th>SUM(Sales_Amt)</th>
<th>Market_Id</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product_Id</td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>P1</td>
<td>3003</td>
<td>1503</td>
</tr>
<tr>
<td>P2</td>
<td>6003</td>
<td>2402</td>
</tr>
<tr>
<td>P3</td>
<td>4503</td>
<td>3</td>
</tr>
<tr>
<td>P4</td>
<td>7503</td>
<td>7000</td>
</tr>
<tr>
<td>Total</td>
<td>21012</td>
<td>10908</td>
</tr>
</tbody>
</table>
The Four Queries

- For the table entries, without the totals (aggregation on time)
  ```sql
  SELECT S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
  FROM Sales S
  GROUP BY S.Market_Id, S.Product_Id
  ```

- For the row totals (aggregation on time and markets)
  ```sql
  SELECT S.Product_Id, SUM(S.Sales_Amt)
  FROM Sales S
  GROUP BY S.Product_Id
  ```

- For the column totals (aggregation on time and products)
  ```sql
  SELECT S.Market_Id, SUM(S.Sales)
  FROM Sales S
  GROUP BY S.Market_Id
  ```

- For the grand total (aggregation on time, markets, and products)
  ```sql
  SELECT SUM(S.Sales)
  FROM Sales S
  ```

Definition of the CUBE Operator

- Doing these four queries is wasteful
  - The first does much of the work of the other two: if we could save that result and aggregate over Market_Id and Product_Id, we could compute the other queries more efficiently

- The CUBE clause is part of SQL:1999
  - GROUP BY CUBE (v1, v2, ..., vn)
  - Equivalent to a collection of GROUP BYs, one for each of the \(2^n\) subsets of v1, v2, ..., vn
Example of CUBE Operator

- The following query returns all the information needed to make the previous products/markets table:

```
SELECT S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```

CUBE Output

<table>
<thead>
<tr>
<th>RESULT SET</th>
<th>Market_Id</th>
<th>Product_Id</th>
<th>Sales_Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>P1</td>
<td></td>
<td>3003</td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td></td>
<td>6003</td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td></td>
<td>4503</td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td></td>
<td>7503</td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td></td>
<td>1503</td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td></td>
<td>2402</td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td></td>
<td>7000</td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td></td>
<td>15003</td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td></td>
<td>24003</td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td></td>
<td>9903</td>
</tr>
<tr>
<td>M1</td>
<td>NULL</td>
<td></td>
<td>21012</td>
</tr>
<tr>
<td>M2</td>
<td>NULL</td>
<td></td>
<td>10908</td>
</tr>
<tr>
<td>M3</td>
<td>NULL</td>
<td></td>
<td>48942</td>
</tr>
<tr>
<td>NULL</td>
<td>P1</td>
<td></td>
<td>19509</td>
</tr>
<tr>
<td>NULL</td>
<td>P2</td>
<td></td>
<td>32408</td>
</tr>
<tr>
<td>NULL</td>
<td>P3</td>
<td></td>
<td>4539</td>
</tr>
<tr>
<td>NULL</td>
<td>P4</td>
<td></td>
<td>24406</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td></td>
<td>80862</td>
</tr>
</tbody>
</table>

FIGURE 17.11 Result set returned with the CUBE operator.
ROLLUP

- ROLLUP is similar to CUBE except that instead of aggregating over all subsets of the arguments, it creates subsets moving from right to left.

- GROUP BY ROLLUP \((A_1, A_2, ..., A_n)\) is a series of these aggregations:
  - GROUP BY \(A_1, ..., A_{n-1}, A_n\)
  - GROUP BY \(A_1, ..., A_{n-1}\)
  - ... ... ...
  - GROUP BY \(A_1, A_2\)
  - GROUP BY \(A_1\)
  - No GROUP BY

- ROLLUP is also in SQL:1999

Example of ROLLUP Operator

```
SELECT S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
FROM Sales S
GROUP BY ROLLUP (S.Market_Id, S.Product_Id)
```

- first aggregates with the finest granularity:
  - GROUP BY S.Market_Id, S.Product_Id
- then with the next level of granularity:
  - GROUP BY S.Market_Id
- then the grand total is computed with no GROUP BY clause
ROLLUP Output

<table>
<thead>
<tr>
<th>RESULT SET</th>
<th>Market_ Id</th>
<th>Product_Id</th>
<th>Sales_Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>P1</td>
<td></td>
<td>3003</td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td></td>
<td>6003</td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td></td>
<td>4503</td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td></td>
<td>7503</td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td></td>
<td>1503</td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td></td>
<td>2402</td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td></td>
<td>7000</td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td></td>
<td>15003</td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td></td>
<td>24003</td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td></td>
<td>9903</td>
</tr>
<tr>
<td>M1</td>
<td>NULL</td>
<td></td>
<td>21012</td>
</tr>
<tr>
<td>M2</td>
<td>NULL</td>
<td></td>
<td>10908</td>
</tr>
<tr>
<td>M3</td>
<td>NULL</td>
<td></td>
<td>48092</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td></td>
<td>80862</td>
</tr>
</tbody>
</table>

FIGURE 17.12 Result set returned with the ROLLUP operator.

ROLLUP vs. CUBE

- The same query with CUBE:
  - first aggregates with the finest granularity:
    GROUP BY S.Market_Id, S.Product_Id
  - then with the next level of granularity:
    GROUP BY S.Market_Id
    and
    GROUP BY S.Product_Id
  - then the grand total with no GROUP BY
Materialized Views

- The CUBE operator is often used to precompute aggregations on all dimensions of a fact table and then save them as a materialized views to speed up future queries.

Categories of OLAP Tools

- Categorized according to the architecture used to store and process multi-dimensional data.
  - Relational OLAP (ROLAP)
  - Multi-dimensional OLAP (MOLAP)
  - Desktop OLAP (DOLAP)
Relational OLAP (ROLAP)

- OLAP data is stored in a relational database. Data cube is a conceptual view – way to think about a fact table
- Fastest-growing style of OLAP technology
  - Need to analyze ever-increasing amounts of data
  - Realize that we cannot store all the data in MOLAP databases
- Supports RDBMS products using a metadata layer
  - Avoids creating a static multi-dimensional data structure
  - Focus on making query processing faster

Multi-dimensional OLAP (MOLAP)

- Vendor provides an OLAP server that implements a fact table as a data cube using a special multi-dimensional (non-relational) data structure
  - No standard query language.
  - Vendors provide proprietary visual languages that allow casual users to make queries that involve pivots, drilling down, or rolling up
- Data is typically aggregated and stored according to predicted usage to enhance query performance.
Desktop OLAP (DOLAP)

- Store the OLAP data in client-based files and support multi-dimensional processing using a client multi-dimensional engine.
  - Move a part of computation to client-side.

- Requires that relatively small extracts of data are held on client machines. They may be distributed in advance, or created on demand (possibly through the Web).

Implementation Issues

- OLAP applications are characterized by a very large amount of data that is relatively static, with infrequent updates
  - Thus, various aggregations can be precomputed and stored in the database
  - Star joins, join indices, and bitmap indices can be used to improve efficiency (recall the methods to compute star joins in Chapter 14)
  - Since updates are infrequent, the inefficiencies associated with updates are minimized
Data Warehouse

- Data (often derived from OLTP) for both OLAP and data mining applications is usually stored in a special database called a **data warehouse**

- Data warehouses are generally large and contain data that has been gathered at different times from DBMSs provided by different vendors and with different schemas

- Populating such a data warehouse is not trivial

Issues Involved in Populating a Data Warehouse

- **Transformations**
  - *Syntactic*: syntax used in different DMBSs for the same data might be different
    - Attribute names: SSN vs. Ssnum
    - Attribute domains: Integer vs. String
  - *Semantic*: semantics might be different
    - Summarizing sales on a daily basis vs. summarizing sales on a monthly basis

- **Data Cleaning**
  - Removing errors and inconsistencies in data
Metadata
- As with other databases, a warehouse must include a metadata repository
  - Information about physical and logical organization of data
  - Information about the source of each data item and the dates on which it was loaded and refreshed

Incremental Updates
- The large volume of data in a data warehouse makes loading and updating a significant task
- For efficiency, updating is usually incremental
  - Different parts are updated at different times
- Incremental updates might result in the database being in an inconsistent state
  - Usually not important because queries involve only statistical summaries of data, which are not greatly affected by such inconsistencies
**Loading Data into A Data Warehouse**

**FIGURE 17.14** Loading data into an OLAP database.

---

**Data Mining**

- An attempt at knowledge discovery
- Searching for patterns and structure in a sea of data
- Uses techniques from many disciplines, such as statistical analysis and machine learning
  - These techniques are not our main interest
Goals of Data Mining

- **Association**
  - Finding patterns in data that associate instances of that data to related instances
  - Example: people who bought book X are also likely to buy book Y.

- **Classification**
  - Finding patterns in data that can be used to classify that data (and possibly the people it describes)
  - Example “high-end buyers” and “low-end buyers”
  - This classification might then be used for prediction
    - Which bank customers will default on their mortgages?
  - Categories for classification are known in advance

- **Clustering**
  - Finding patterns in data that can be used to classify that data (and possibly the people it describes) into categories determined by a similarity measure
  - Example: Are cancer patients clustered in any geographic area (possibly around certain power plants)?
  - Categories are not known in advance, unlike in the classification problem

Associations

- An *association* is a correlation between certain values in a database (in the same or different columns)
  - In a convenience store in the early evening, a large percentage of customers who bought diapers also bought beer

- This association can be described using the notation
  
  \[
  \text{Purchase}_{-}\text{diapers} \Rightarrow \text{Purchase}_{-}\text{beer}
  \]
Confidence and Support

- To determine whether an association exists, the system computes the **confidence** and **support** for that association.

- **Confidence** in $A \Rightarrow B$
  - The percentage of transactions (recorded in the database) that contain $B$ among those that contain $A$
  - Diapers $\Rightarrow$ Beer:
    - The percentage of customers who bought beer among those who bought diapers.

- **Support**
  - The percentage of transactions that contain both items among all transactions
  - $100 \times (\text{customers who bought both Diapers and Beer})/(\text{all customers})$.

Confidence and Support

- Confidence: 2/3
- Support: 2/4

<table>
<thead>
<tr>
<th>PURCHASES</th>
<th>Transaction_Id</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td></td>
<td>diapers</td>
</tr>
<tr>
<td>001</td>
<td></td>
<td>beer</td>
</tr>
<tr>
<td>001</td>
<td></td>
<td>popcorn</td>
</tr>
<tr>
<td>001</td>
<td></td>
<td>bread</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>diapers</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>cheese</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>soda</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>beer</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>juice</td>
</tr>
<tr>
<td>003</td>
<td></td>
<td>diapers</td>
</tr>
<tr>
<td>003</td>
<td></td>
<td>cold cuts</td>
</tr>
<tr>
<td>003</td>
<td></td>
<td>cookies</td>
</tr>
<tr>
<td>003</td>
<td></td>
<td>napkins</td>
</tr>
<tr>
<td>004</td>
<td></td>
<td>cereal</td>
</tr>
<tr>
<td>004</td>
<td></td>
<td>beer</td>
</tr>
<tr>
<td>004</td>
<td></td>
<td>cold cuts</td>
</tr>
</tbody>
</table>

**FIGURE 17.15** PURCHASES table used for data mining.
Ascertain an Association

- To ascertain that an association exists, both the confidence and the support must be above a certain threshold
  - **Confidence** states that there is a high probability, given the data, that someone who purchased diapers also bought beer
  - **Support** states that the data shows a large percentage of people who purchased both diapers and beer (so that the confidence measure is not an accident)

A Priori Algorithm for Computing Associations

- Based on this observation:
  - If the support for $A \Rightarrow B$ is larger than $T$, then the support for $A$ and $B$ must separately be larger than $T$

- Find all items whose support is larger than $T$
  - Requires checking $n$ items
  - If there are $m$ items with support $> T$ (presumably, $m<<n$), find all pairs of such items whose support is larger than $T$
  - Requires checking $m(m-1)$ pairs

- If there are $p$ pairs with support $> T$, compute the confidence for each pair
  - Requires checking $p$ pairs
Classification

- *Classification* involves finding patterns in data items that can be used to place those items in certain categories. That classification can then be used to predict future outcomes.
  - A bank might gather data from the application forms of past customers who applied for a mortgage and classify them as *defaulters* or *non-defaulters*.
  - Then when new customers apply, they might use the information on their application forms to predict whether or not they would default.

Example: Loan Risk Evaluation

- Suppose the bank used only three types of information to do the classification
  - Whether or not the applicant was married
  - Whether or not the applicant had previously defaulted
  - The applicant’s current income
- The data about previous applicants might be stored in a table called the *training table*
Training Table

<table>
<thead>
<tr>
<th>CUSTOMER Id</th>
<th>Married</th>
<th>PrevDefault</th>
<th>Income</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>yes</td>
<td>no</td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>C2</td>
<td>yes</td>
<td>no</td>
<td>100</td>
<td>no</td>
</tr>
<tr>
<td>C3</td>
<td>no</td>
<td>yes</td>
<td>135</td>
<td>yes</td>
</tr>
<tr>
<td>C4</td>
<td>yes</td>
<td>no</td>
<td>125</td>
<td>no</td>
</tr>
<tr>
<td>C5</td>
<td>yes</td>
<td>no</td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>C6</td>
<td>no</td>
<td>no</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td>C7</td>
<td>yes</td>
<td>yes</td>
<td>10</td>
<td>no</td>
</tr>
<tr>
<td>C8</td>
<td>yes</td>
<td>no</td>
<td>10</td>
<td>yes</td>
</tr>
<tr>
<td>C9</td>
<td>yes</td>
<td>no</td>
<td>75</td>
<td>no</td>
</tr>
<tr>
<td>C10</td>
<td>yes</td>
<td>yes</td>
<td>45</td>
<td>no</td>
</tr>
<tr>
<td>C11</td>
<td>yes</td>
<td>no</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td>C12</td>
<td>no</td>
<td>yes</td>
<td>125</td>
<td>yes</td>
</tr>
<tr>
<td>C13</td>
<td>yes</td>
<td>yes</td>
<td>20</td>
<td>no</td>
</tr>
<tr>
<td>C14</td>
<td>no</td>
<td>no</td>
<td>15</td>
<td>no</td>
</tr>
<tr>
<td>C15</td>
<td>no</td>
<td>no</td>
<td>60</td>
<td>no</td>
</tr>
<tr>
<td>C16</td>
<td>yes</td>
<td>no</td>
<td>15</td>
<td>yes</td>
</tr>
<tr>
<td>C17</td>
<td>yes</td>
<td>no</td>
<td>35</td>
<td>no</td>
</tr>
<tr>
<td>C18</td>
<td>no</td>
<td>yes</td>
<td>160</td>
<td>yes</td>
</tr>
<tr>
<td>C19</td>
<td>yes</td>
<td>no</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>C20</td>
<td>yes</td>
<td>no</td>
<td>30</td>
<td>no</td>
</tr>
</tbody>
</table>

Classification Using Decision Trees

- The goal is to use the information in this table to classify new applicants into defaulters or non defaulters.
- One approach is to use the training table to make a decision tree.
A Decision Tree

Decision Trees Imply Classification Rules

- Each classification rule implied by the tree corresponds to a path from the root to a leaf.
- For example, one such rule is
  
  If
  
  \(\text{PreviousDefault} = \text{No} \ AND \ \text{Married} = \text{Yes} \ AND \ \text{Income} < 30\)
  
  Then
  
  \(\text{Default} = \text{Yes}\)
Decision Trees Might Make Mistakes

- Some of the classification rules developed from a decision tree might incorrectly classify some data; for example

  If  PreviousDefault = No  AND  Married = Yes  AND  Income  >=  30
  Then  Default = No
  does not correctly classify customer C11

- It is unreasonable to expect that a small number of classification rules can always correctly classify a large amount of data

- Goal: Produce a small enough tree with a small enough number of errors