CSE 532 – Theory of Database Systems

Lecture 24 (Chapter 17)
OLAP and Data Mining

Lecturer: Sael Lee

Slide adapted from the author’s and Dr. Ilchul Yoon’s slides.
OLTP Compared With OLAP

- **On Line Transaction Processing – OLTP**
  - Maintains a database that is an accurate model of some real-world enterprise. Supports day-to-day operations.
  - Characteristics:
    - Short simple transactions
    - Relatively frequent updates
    - Transactions access only a small fraction of the database

- **On Line Analytic Processing – OLAP**
  - Uses information in database to guide strategic decisions.
  - Characteristics:
    - Complex queries
    - Infrequent updates
    - Transactions access a large fraction of the database
    - Data need not be up-to-date
The Internet Grocer

- **OLTP-style transaction:**
  - John Smith, from Schenectady, N.Y., just bought a box of tomatoes; charge his account; deliver the tomatoes from our Schenectady warehouse; decrease our inventory of tomatoes from that warehouse

- **OLAP-style transaction:**
  - How many cases of tomatoes were sold in all northeast warehouses in the years 2000 and 2001?
# OLTP Systems vs. OLAP

<table>
<thead>
<tr>
<th>Criteria</th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many?</td>
<td>multiple in an organization</td>
<td>1</td>
</tr>
<tr>
<td>Purpose</td>
<td>Operational processing</td>
<td>Analytical processing</td>
</tr>
<tr>
<td>Data Age</td>
<td>Current</td>
<td>Historic</td>
</tr>
<tr>
<td>Latency</td>
<td>Real-time</td>
<td>Periodic</td>
</tr>
<tr>
<td>Data Granularity</td>
<td>Detailed</td>
<td>Detailed + Summarized</td>
</tr>
<tr>
<td>Reporting</td>
<td>Static fixed format</td>
<td>Dynamic, multi-dimensional</td>
</tr>
<tr>
<td>Users</td>
<td>Operators</td>
<td>Managerial users</td>
</tr>
</tbody>
</table>
OLAP: Traditional Compared with Newer Applications

● Traditional OLAP queries
  ● Uses data the enterprise gathers in its usual activities, perhaps in its OLTP system
  ● Queries are ad hoc, perhaps designed and carried out by non-professionals (managers)

● Newer Applications (e.g., Internet companies)
  ● Enterprise actively gathers data it wants, perhaps purchasing it
  ● Queries are sophisticated, designed by professionals, and used in more sophisticated ways

● The dynamic synthesis, analysis, and consolidation of large volumes of multi-dimensional data - Codd (1993).
OLAP Tools

- There are many varieties of OLAP tools available in the marketplace.
OLAP Tools

2006 Revenue Growth

29.4%
2006 revenue growth of 29.4% compares to a target of 22.7%.

2006 Profit Growth

28.3%
2006 profit growth of 28.3% compares to a target of 25%.

2006 Profit Margin

17.7%
2006 profit margin of 17.7% compares to a target of 16.1%.

Revenue | Profit | Profit Margin

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>$321,620</td>
<td>$384,427</td>
<td>$374,590</td>
<td>$378,383</td>
<td>$357,618</td>
<td>$424,680</td>
<td>$378,587</td>
<td>$376,008</td>
<td>$428,996</td>
<td>$438,890</td>
<td>$430,962</td>
<td>$453,811</td>
</tr>
<tr>
<td>Revenue Forecast</td>
<td>$328,052</td>
<td>$365,206</td>
<td>$374,590</td>
<td>$423,405</td>
<td>$272,874</td>
<td>$450,161</td>
<td>$315,746</td>
<td>$419,625</td>
<td>$468,915</td>
<td>$456,446</td>
<td>$426,905</td>
<td>$497,348</td>
</tr>
<tr>
<td>Variance</td>
<td>(6,432)</td>
<td>19,221</td>
<td>0</td>
<td>(45,022)</td>
<td>84,744</td>
<td>(25,481)</td>
<td>62,840</td>
<td>(43,617)</td>
<td>(39,919)</td>
<td>(17,556)</td>
<td>4,057</td>
<td>(43,537)</td>
</tr>
<tr>
<td>Revenue Growth %</td>
<td>30.9%</td>
<td>46.7%</td>
<td>21.6%</td>
<td>35.6%</td>
<td>47.1%</td>
<td>42.1%</td>
<td>32.2%</td>
<td>26.5%</td>
<td>20.8%</td>
<td>33.9%</td>
<td>18.6%</td>
<td>12.9%</td>
</tr>
</tbody>
</table>
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

  \[
  \text{Sales} \ (\text{Market}_\text{Id}, \text{Product}_\text{Id}, \text{Time}_\text{Id}, \text{Sales}_\text{Amt})
  \]

- The table can be viewed as **multidimensional**
  - \(\text{Market}_\text{Id}, \text{Product}_\text{Id}, \text{Time}_\text{Id}\) are the dimensions that represent specific supermarkets, products, and time intervals
  - \(\text{Sales}_\text{Amt}\) 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>500</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td>T1</td>
<td>800</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>5000</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>M1</td>
<td>P1</td>
<td>T2</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td>T2</td>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td>T2</td>
<td>1501</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td>T2</td>
<td>2501</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td>T2</td>
<td>501</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td>T2</td>
<td>801</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td>T2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>T2</td>
<td>3334</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td>T2</td>
<td>5001</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td>T2</td>
<td>8001</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td>T2</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td>T2</td>
<td>3301</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P1</td>
<td>T3</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td>T3</td>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td>T3</td>
<td>1502</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td>T3</td>
<td>2502</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td>T3</td>
<td>502</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td>T3</td>
<td>802</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td>T3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>T3</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td>T3</td>
<td>5002</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td>T3</td>
<td>8002</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td>T3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>M3</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:
  
  ```
  SELECT S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
  FROM Sales S
  GROUP BY S.Market_Id, 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</td>
</tr>
<tr>
<td>P1</td>
<td>3003</td>
</tr>
<tr>
<td>P2</td>
<td>6003</td>
</tr>
<tr>
<td>P3</td>
<td>4503</td>
</tr>
<tr>
<td>P4</td>
<td>7503</td>
</tr>
</tbody>
</table>
Drilling Down and Rolling Up

- Some dimension tables form an *aggregation hierarchy*

  Market_Id → City → State → 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 (Market_Id, Product_Id, Time_Id, Sales_Amt)
Market (Market_Id, City, State, Region)

1. SELECT S.Product_Id, M.Region, SUM (S.Sales_Amt)
   FROM Sales S, Market M
   WHERE M.Market_Id = S.Market_Id
   GROUP BY S.Product_Id, M.Region

2. SELECT S.Product_Id, M.State, SUM (S.Sales_Amt)
   FROM Sales S, Market M
   WHERE M.Market_Id = S.Market_Id
   GROUP BY S.Product_Id, M.State,
Rolling Up

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

1. SELECT S.Product_Id, M.State, SUM (S.Sales_Amt) FROM Sales S, Market M WHERE M.Market_Id = S.Market_Id GROUP BY S.Product_Id, M.State

then we can roll up from there to:

2. SELECT T.Product_Id, M.Region, SUM (T.Sales_Amt) FROM State_Sales T, Market M WHERE M.State = T.State GROUP BY T.Product_Id, M.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 $D_1,...,D_k$ in a data cube $D_1,...,D_k,D_{k+1},...,D_n$ means that we use \texttt{GROUP BY $A_1,...,A_k$} and aggregate over $A_{k+1},...A_n$, where $A_i$ is an attribute of the dimension $D_i$.

  - **Example**: Pivoting on Product and Time corresponds to grouping on \texttt{Product_id} and \texttt{Quarter} and aggregating \texttt{Sales_Amt} over \texttt{Market_id}:

    ```sql
    SELECT S.Product_Id, T.Quarter, SUM (S.Sales_Amt)
    FROM Sales S, Time T
    WHERE T.Time_Id = S.Time_Id
    GROUP BY S.Product_Id, T_QUARTER
    ```

![Pivot Example](chart.png)
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 S.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 S.Product_Id, T.Week
```

Slice

Pivot
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></td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>Product_Id</td>
<td></td>
<td></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)
  
  ```
  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)
  
  ```
  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)
  
  ```
  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)
  
  ```
  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 \((v_1, v_2, \ldots, v_n)\)
  - Equivalent to a collection of `GROUP BY`s, one for each of the \(2^n\) subsets of \(v_1, v_2, \ldots, v_n\)
Example of CUBE Operator

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

```sql
SELECT S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```
CSE 532 – Theory of Database Systems

Lecture 25 (Chapter 17)
OLAP and Data Mining

Lecturer: Sael Lee

Slide adapted from the author’s and Dr. Ilchul Yoon’s slides.
## 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** \((A_1, A_2, \ldots, A_n)\) is a series of these aggregations:
  - **GROUP BY** \(A_1, \ldots, A_{n-1}, A_n\)
  - **GROUP BY** \(A_1, \ldots, A_{n-1}\)
  - \(\ldots \ldots \ldots\)
  - **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>3003</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P2</td>
<td>6003</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P3</td>
<td>4503</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>P4</td>
<td>7503</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P1</td>
<td>1503</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P2</td>
<td>2402</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>7000</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P1</td>
<td>15003</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P2</td>
<td>24003</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P3</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td>9903</td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>NULL</td>
<td>21012</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>NULL</td>
<td>10908</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>NULL</td>
<td>48092</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>80862</td>
<td></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

Figure 33.3
Architecture for ROLAP tools.
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.
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

  Purchase_diapers => Purchase_beer
Confidence and Support

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

**Confidence** in A => B
- The percentage of transactions (recorded in the database) that contain B among those that contain A
  - Diapers => 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* (customers who bought both Diapers and Beer)/(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></td>
<td>001</td>
<td>diapers</td>
</tr>
<tr>
<td></td>
<td>001</td>
<td>beer</td>
</tr>
<tr>
<td></td>
<td>001</td>
<td>popcorn</td>
</tr>
<tr>
<td></td>
<td>001</td>
<td>bread</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>diapers</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>cheese</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>soda</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>beer</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>juice</td>
</tr>
<tr>
<td></td>
<td>003</td>
<td>diapers</td>
</tr>
<tr>
<td></td>
<td>003</td>
<td>cold cuts</td>
</tr>
<tr>
<td></td>
<td>003</td>
<td>cookies</td>
</tr>
<tr>
<td></td>
<td>003</td>
<td>napkins</td>
</tr>
<tr>
<td></td>
<td>004</td>
<td>cereal</td>
</tr>
<tr>
<td></td>
<td>004</td>
<td>beer</td>
</tr>
<tr>
<td></td>
<td>004</td>
<td>cold cuts</td>
</tr>
</tbody>
</table>

**FIGURE 17.15** PURCHASES table used for data mining.
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</th>
<th>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></td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>C2</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>100</td>
<td>no</td>
</tr>
<tr>
<td>C3</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>135</td>
<td>yes</td>
</tr>
<tr>
<td>C4</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>125</td>
<td>no</td>
</tr>
<tr>
<td>C5</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>C6</td>
<td>no</td>
<td>no</td>
<td></td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td>C7</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>10</td>
<td>no</td>
</tr>
<tr>
<td>C8</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>10</td>
<td>yes</td>
</tr>
<tr>
<td>C9</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>75</td>
<td>no</td>
</tr>
<tr>
<td>C10</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>45</td>
<td>no</td>
</tr>
<tr>
<td>C11</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td>C12</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>125</td>
<td>yes</td>
</tr>
<tr>
<td>C13</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>20</td>
<td>no</td>
</tr>
<tr>
<td>C14</td>
<td>no</td>
<td>no</td>
<td></td>
<td>15</td>
<td>no</td>
</tr>
<tr>
<td>C15</td>
<td>no</td>
<td>no</td>
<td></td>
<td>60</td>
<td>no</td>
</tr>
<tr>
<td>C16</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>15</td>
<td>yes</td>
</tr>
<tr>
<td>C17</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>35</td>
<td>no</td>
</tr>
<tr>
<td>C18</td>
<td>no</td>
<td>yes</td>
<td></td>
<td>160</td>
<td>yes</td>
</tr>
<tr>
<td>C19</td>
<td>yes</td>
<td>no</td>
<td></td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>C20</td>
<td>yes</td>
<td>no</td>
<td></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 for predicting defaults on a mortgage.
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
  
  \[\text{PreviousDefault} = \text{No} \quad \text{AND} \quad \text{Married} = \text{Yes} \quad \text{AND} \quad \text{Income} < 30\]
  
  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:
  
  \[
  \text{If } \  \text{PreviousDefault} = \text{No} \ \text{AND} \ \text{Married} = \text{Yes} \ \text{AND} \ \text{Income} \geq 30 \\
  \text{Then } \ \text{Default} = \text{No}
  \]
  
  does not correctly classify customer C11.

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

  - \textbf{Goal: Produce a small enough tree with a small enough number of errors}
Several algorithms have been developed for constructing a decision tree from a training set.

- We discuss the **ID3 algorithm** (top-down).

ID3 starts by selecting the attribute to be used at the top level of the tree to make the first decision.

- This decision yields the nodes at the second level of the tree. The procedure repeats for each of these nodes, and so on.
Picking the Top-Most Attribute

- Intuitively we want to pick the attribute that gives the “most information” about the final decision.

- The ID3 algorithm uses the entropy measure from Information Theory:

\[ \text{entropy}(\text{TrainingTable}) = - \sum_{i \in \text{outcomes}} p_i \log_2 p_i \]

\[ p_i = \text{probability of the outcome } i \text{ in TrainingTable} \]

- Practically: \( p_i \) is approximated as

\[ p_i = (\#\text{items in the table with outcome}=i) / (\#\text{ of all items in the table}) \]
Properties of the Entropy – $\Sigma p_i \log_2 p_i$

- Entropy determines the degree of randomness in the data:
  - $p_{\text{yes}} = p_{\text{no}} = \frac{1}{2}$ – *data is completely random*
    
    $$\text{entropy} = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1$$
  - $p_{\text{yes}} = 1, p_{\text{no}} = 0$ or $p_{\text{no}} = 1, p_{\text{yes}} = 0$ – *data is totally nonrandom*
    
    $$\text{entropy} = -1 \log_2 1 - 0 \log_2 0 = 0$$

- The lower the entropy – the less randomness exists in the data \(\equiv\) the more information exists in that data.
Information Gain

- For the entire table, 6 entries have the outcome “Yes” and 14 have the outcome “No”
  - So the entropy of the entire table is
    - \[-(6/20 \log_2 6/20 + 14/20 \log_2 14/20) = .881\]

- The ID3 algorithm selects as the top-most node the attribute that provides the largest \textit{information gain} (explained next)
Information Gain (cont’d)

- Select an attribute, $A$, and compute the entropies of the subtrees w.r.t. that attribute.

- **Information gain for $A$:**
  \[
  \text{entropy} - (\sum_{i=1}^{n} \text{entropy}_i \times \text{weight}_i)
  \]

  - $\text{weight}_i = \frac{\text{tuples}(T_i)}{\text{tuples}(T)}$
  - How much less random the data has become after splitting the training set into subtrees.
Information Gain (con’t)

If the top-most node in the tree were *Previous Default*, there would be two subtrees:
- A subtree with *Previous Default* = “Yes”
- A subtree with *Previous Default* = “No”

The entropies of these two subtrees would be
- **For *Previous Default* = “Yes”:**
  - 4 of the 6 entries have the outcome “Yes” and 2 have “No”
    - The entropy is $-\frac{4}{6} \log_2 \frac{4}{6} - \frac{2}{6} \log_2 \frac{2}{6} = .918$
- **For *Previous Default* = “No”:**
  - 2 of the 14 entries have the outcome “Yes” and 12 have “No”
    - The entropy is $-\frac{2}{14} \log_2 \frac{2}{14} - \frac{12}{14} \log_2 \frac{12}{14} = .592$

The average entropy of these subtrees is
\[
\frac{6}{20} \times 0.918 + \frac{14}{20} \times 0.592 = 0.690
\]

The **Information Gain** from using *Previous Default* as the top attribute is
\[
0.881 - 0.690 = 0.191
\]
Comparing Information Gains

- **Previous Default** as the top-most attribute
  - The information gain = .191
- **Married** as the top-most attribute
  - The information gain = .056
- **Income** as the top-most attribute
  - Must compute information gain for all possible ranges
  - For example for the range Income < 50 and Income >= 50 the Information Gain is .031 --- discretization

The maximum Information Gain turns out to be for the attribute **Previous Default**, so we select that as the top-most attribute in the decision tree
The Rest of the Tree

- Repeat the process on each of the subtrees
  - Different subtrees might have different top-most nodes and/or different ranges for Income
  - If all nodes in a subtree have the same outcome:
    - the subtree becomes a leaf node and the procedure stops for that subtree
  - If not all nodes in a subtree have the same outcome:
    - If there are no more attributes to use: That subtree becomes a leaf node and the procedure stops for that subtree
      - The classification rules that access such a subtree will incorrectly classify some data.
        E.g., the subtree PreviousDefault = No AND Married = Yes AND Income >= 30 incorrectly classifies C11.
    - If there are more attributes to use: Continue the process
Other Measures of Randomness

- A number of other measures can be used to produce a decision tree from a training set

- **Gain Ratio = (Information Gain)/SplitInfo**
  - where SplitInfo = $- \sum_i | t_i | / | t | \cdot \log_2 (| t_i | / | t |)$
  - $|t|$ is the number of entries in the table being decomposed
  - $|t_i|$ is the number of entries in the $i^{th}$ table produced

- **Gini Index = 1 - $\sum_i p_i^2$**
Clustering

- **Given:**
  - a set of *items*
  - a set of *characteristic attributes* for the items
  - a *similarity measure* based on those attributes

- **Clustering** involves placing those items into *clusters*, such that items in the same cluster are close according to the similarity measure.
  - Different from classification: there the categories are known in advance
  - For example, cancer patients might have the attribute *location*, and might be placed in clusters with similar locations.
Example: Clustering Students by Age

<table>
<thead>
<tr>
<th>Student</th>
<th>Id</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>17</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td>S_2</td>
<td>17</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>S_3</td>
<td>18</td>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td>S_4</td>
<td>20</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>S_5</td>
<td>23</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>S_6</td>
<td>26</td>
<td>3.6</td>
<td></td>
</tr>
</tbody>
</table>
K-Means Algorithm

- Center of a cluster is the mean of the items in the cluster

- To cluster a set of items into $k$ categories
  1. Pick $k$ items at random to be the (initial) centers of the clusters (so each selected item is in its own cluster)
  2. Place each item from the training set in the cluster to whose center it is the closest
  3. Recalculate the centers of each cluster
  4. Repeat the procedure starting at Step 2 until there is no change in the membership of any cluster
The Student Example (con’t)

- Suppose we want 2 clusters based on Age
  - Randomly pick S1 (age 17) and S4 (age 20) as the centers of the initial centers
  - The initial clusters are
    17  17  18         20  23  26
  - The centers of these clusters are
    17.333  and  23
  - Redistribute items among the clusters based on the new centers:
    17  17  18  20        23  26
  - If we repeat the procedure, the clusters remain the same

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>17</td>
<td>3.9</td>
</tr>
<tr>
<td>S_2</td>
<td>17</td>
<td>3.5</td>
</tr>
<tr>
<td>S_3</td>
<td>18</td>
<td>3.1</td>
</tr>
<tr>
<td>S_4</td>
<td>20</td>
<td>3.0</td>
</tr>
<tr>
<td>S_5</td>
<td>23</td>
<td>3.5</td>
</tr>
<tr>
<td>S_6</td>
<td>26</td>
<td>3.6</td>
</tr>
</tbody>
</table>
The Hierarchical or Agglomerative Algorithm

- Number of clusters is not fixed in advance
- Initially select each item in the training set as the center of its own cluster
- Select two clusters to merge into a single cluster
  - One approach is to pick the clusters whose centers are the closest according to some measure (e.g., Euclidean distance)
- Continue until some termination condition is reached (e.g., the number of clusters falls below some limit)
Student Example (con’t)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>26</td>
</tr>
</tbody>
</table>

space complexity and time complexity?

space: $O(n^2)$ – for all pair matrix

time: $O(n^3)$ – $n$ iterations with $n^2$ updates
Dendrogram

- One way to manually **analyze** the results of the hierarchical algorithm is with the use of a tree called a **dendrogram**

- The nodes are clusters in the intermediate stages of the hierarchical algorithm

- The tree is constructed in reverse order of the execution of the hierarchical algorithm, starting with the final (single) cluster
Dendrogram for the Student Example

FIGURE 17.29 A dendrogram corresponding to the example of the hierarchical clustering algorithm.

Possible set of clusters
Analysis of Dendrogram

- Any set of nodes whose children *partition* all the leaves is a possible clustering
- For example,

  17  17  18  20  23  26

  is an allowable set of clusters.

*Note*: these clusters were not seen at any of the intermediate steps in the hierarchical or K-means algorithms!