Lecture 2: Data Warehouse and OLAP
Outline

• DSS, Data Warehouse, and OLAP
• Models & operations
  – OLAP operations
  – SQL:1999 Supports
• Implementation Techniques
  – View Materialization
  – Indexing
• Future directions
The Need for Data Analysis

• Managers must be able to track daily transactions to evaluate how the business is performing

• By tapping into the operational database, management can develop strategies to meet organizational goals

• Data analysis can provide information about short-term tactical evaluations and strategies
Decision Support Systems

• Methodology (or series of methodologies) designed to extract information from data and to use such information as a basis for decision making

• Decision support system (DSS):
  – Arrangement of computerized tools used to assist managerial decision making within a business
  – Usually requires extensive data “massaging” to produce information
  – Used at all levels within an organization
  – Often tailored to focus on specific business areas
  – Provides ad hoc query tools to retrieve data and to display data in different formats
Decision Support Systems (continued)

• Composed of four main components:
  – Data store component
    • Basically a DSS database
  – Data extraction and filtering component
    • Used to extract and validate data taken from operational database and external data sources
  – End-user query tool
    • Used to create queries that access database
  – End-user presentation tool
    • Used to organize and present data
Main Components of a Decision Support System (DSS)

**FIGURE 12.1** Main Components of a Decision Support System (DSS)
Transforming Operational Data Into Decision Support Data

Operational data have a narrow time span, low granularity, and single focus. Such data are usually presented in tabular format, in which each row represents a single transaction. This format often makes it difficult to derive useful information.

Decision support system (DSS) data focus on a broader time span, tend to have high levels of granularity, and can be examined in multiple dimensions. For example, note these possible aggregations:
- Sales by product, region, agent, etc.
- Sales for all years or only a few selected years.
- Sales for all products or only a few selected products.
Contrasting Operational and DSS Data Characteristics

<table>
<thead>
<tr>
<th>CHARACTERISTIC</th>
<th>OPERATIONAL DATA</th>
<th>DSS DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data currency</td>
<td>Current operations</td>
<td>Historic data</td>
</tr>
<tr>
<td></td>
<td>Real-time data</td>
<td>Snapshot of company data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time component (week/month/year)</td>
</tr>
<tr>
<td>Granularity</td>
<td>Atomic-detailed data</td>
<td>Summarized data</td>
</tr>
<tr>
<td>Summarization level</td>
<td>Low; some aggregate yields</td>
<td>High; many aggregation levels</td>
</tr>
<tr>
<td>Data model</td>
<td>Highly normalized</td>
<td>Nonnormalized</td>
</tr>
<tr>
<td></td>
<td>Mostly relational DBMS</td>
<td>Complex structures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Some relational, but mostly multidimensional DBMS</td>
</tr>
<tr>
<td>Transaction type</td>
<td>Mostly updates</td>
<td>Mostly query</td>
</tr>
<tr>
<td>Transaction volumes</td>
<td>High update volumes</td>
<td>Periodic loads and summary calculations</td>
</tr>
<tr>
<td>Transaction speed</td>
<td>Updates are critical</td>
<td>Retrievals are critical</td>
</tr>
<tr>
<td>Query activity</td>
<td>Low to medium</td>
<td>High</td>
</tr>
<tr>
<td>Query scope</td>
<td>Narrow range</td>
<td>Broad range</td>
</tr>
<tr>
<td>Query complexity</td>
<td>Simple to medium</td>
<td>Very complex</td>
</tr>
<tr>
<td>Data volumes</td>
<td>Hundreds of megabytes and up to gigabytes</td>
<td>Hundreds of gigabytes to terabytes</td>
</tr>
</tbody>
</table>
Ten-Year Sales History for a Single Department, in Millions of Dollars

<table>
<thead>
<tr>
<th>YEAR</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>8,227</td>
</tr>
<tr>
<td>1995</td>
<td>9,109</td>
</tr>
<tr>
<td>1996</td>
<td>10,104</td>
</tr>
<tr>
<td>1997</td>
<td>11,553</td>
</tr>
<tr>
<td>1998</td>
<td>10,018</td>
</tr>
<tr>
<td>1999</td>
<td>11,875</td>
</tr>
<tr>
<td>2000</td>
<td>12,699</td>
</tr>
<tr>
<td>2001</td>
<td>14,875</td>
</tr>
<tr>
<td>2002</td>
<td>16,301</td>
</tr>
<tr>
<td>2003</td>
<td>19,986</td>
</tr>
</tbody>
</table>
Yearly Sales Summaries, Two Stores and Two Departments per Store, in Millions of Dollars

<table>
<thead>
<tr>
<th>YEAR</th>
<th>STORE</th>
<th>DEPARTMENT</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>A</td>
<td>1</td>
<td>1,985</td>
</tr>
<tr>
<td>1994</td>
<td>A</td>
<td>2</td>
<td>2,401</td>
</tr>
<tr>
<td>1994</td>
<td>B</td>
<td>1</td>
<td>1,879</td>
</tr>
<tr>
<td>1994</td>
<td>B</td>
<td>2</td>
<td>1,962</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1998</td>
<td>A</td>
<td>1</td>
<td>3,912</td>
</tr>
<tr>
<td>1998</td>
<td>A</td>
<td>2</td>
<td>4,158</td>
</tr>
<tr>
<td>1998</td>
<td>B</td>
<td>1</td>
<td>3,426</td>
</tr>
<tr>
<td>1998</td>
<td>B</td>
<td>2</td>
<td>1,203</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2003</td>
<td>A</td>
<td>1</td>
<td>7,683</td>
</tr>
<tr>
<td>2003</td>
<td>A</td>
<td>2</td>
<td>6,912</td>
</tr>
<tr>
<td>2003</td>
<td>B</td>
<td>1</td>
<td>3,768</td>
</tr>
<tr>
<td>2003</td>
<td>B</td>
<td>2</td>
<td>1,623</td>
</tr>
</tbody>
</table>
The Data Warehouse

• Integrated, subject-oriented, time-variant, nonvolatile database that provides support for decision making
• Growing industry: $8 billion in 1998
• Range from desktop to huge:
  – Walmart: 900-CPU, 2,700 disk, 23TB Teradata system
• Lots of buzzwords, hype
  – slice & dice, rollup, MOLAP, pivot, ...
What is a Warehouse?

• Collection of diverse data
  – subject oriented
  – aimed at executive, decision maker
  – often a copy of operational data
  – with value-added data (e.g., summaries, history)
  – integrated
  – time-varying
  – non-volatile
What is a Warehouse?

- Collection of tools
  - gathering data
  - cleansing, integrating, ...
  - querying, reporting, analysis
  - data mining
  - monitoring, administering warehouse
Warehouse Architecture

Client

Query & Analysis

Metadata

Warehouse

Integration

Source

Source

Source
Motivating Examples

• Forecasting
• Comparing performance of units
• Monitoring, detecting fraud
• Visualization
Why a Warehouse?

- Two Approaches:
  - Query-Driven (Lazy)
  - Warehouse (Eager)
Query-Driven Approach
Advantages of Warehousing

- High query performance
- Queries not visible outside warehouse
- Local processing at sources unaffected
- Can operate when sources unavailable
- Can query data not stored in a DBMS
- Extra information at warehouse
  - Modify, summarize (store aggregates)
  - Add historical information
Advantages of Query-Driven

• No need to copy data
  – less storage
  – no need to purchase data
• More up-to-date data
• Query needs can be unknown
• Only query interface needed at sources
Data Marts

- Smaller warehouses
- Spans part of organization
  - e.g., marketing (customers, products, sales)
- Do not require enterprise-wide consensus
  - but long term integration problems?
## A Comparison of Data Warehouse and Operational Database Characteristics

<table>
<thead>
<tr>
<th>CHARACTERISTIC</th>
<th>OPERATIONAL DATABASE DATA</th>
<th>DATA WAREHOUSE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated</td>
<td>Similar data can have different representations or meanings. For example, Social Security numbers may be stored as $###-###-#####, or as $###########$, and a given condition may be labeled as T/F or 0/1 or Y/N. A sales value may be shown in thousands or in millions.</td>
<td>Provide a unified view of all data elements with a common definition and representation for all business units.</td>
</tr>
<tr>
<td>Subject-oriented</td>
<td>Data are stored with a functional, or process, orientation. For example, data may be stored for invoices, payments, credit amounts, and so on.</td>
<td>Data are stored with a subject orientation that facilitates multiple views of the data and facilitates decision making. For example, sales may be recorded by product, by division, by manager, or by region.</td>
</tr>
<tr>
<td>Time-variant</td>
<td>Data are recorded as current transactions. For example, the sales data may be the sale of a product on a given date, such as $342.78 on 12-MAY-2004.</td>
<td>Data are recorded with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.</td>
</tr>
<tr>
<td>Nonvolatile</td>
<td>Data updates are frequent and common. For example, an inventory amount changes with each sale. Therefore, the data environment is fluid.</td>
<td>Data cannot be changed. Data are only added periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.</td>
</tr>
</tbody>
</table>
Creating a Data Warehouse

**Figure 12.3 Creating a Data Warehouse**

- Operational data
- Data extraction
  - Extract
  - Filter
  - Transform
  - Integrate
  - Classify
  - Aggregate
  - Summarize
- Data warehouse
  - Integrated
  - Subject-oriented
  - Time-variant
  - Nonvolatile
Online Analytical Processing

• Advanced data analysis environment that supports decision making, business modeling, and operations research

• OLAP systems share four main characteristics:
  – Use multidimensional data analysis techniques
  – Provide advanced database support
  – Provide easy-to-use end-user interfaces
  – Support client/server architecture
OLTP vs. OLAP

- **OLTP**: On Line Transaction Processing
  - Describes processing at operational sites
- **OLAP**: On Line Analytical Processing
  - Describes processing at warehouse
OLTP vs. OLAP

**OLTP**

- Mostly updates
- Many small transactions
- Mb-Tb of data
- Raw data
- Clerical users
- Up-to-date data
- Consistency, recoverability critical

**OLAP**

- Mostly reads
- Queries long, complex
- Gb-Tb of data
- Summarized, consolidated data
- Decision-makers, analysts as users
Operational vs. Multidimensional View of Sales

**Figure 12.4 Operational vs. Multidimensional View of Sales**

**Table name: DW_INVOICE**

<table>
<thead>
<tr>
<th>INV_NUM</th>
<th>INV_DATE</th>
<th>CUS_NAME</th>
<th>INV_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>2034</td>
<td>15-May-04</td>
<td>Dartonik</td>
</tr>
<tr>
<td>+</td>
<td>2036</td>
<td>15-May-04</td>
<td>Summer Lake</td>
</tr>
<tr>
<td>+</td>
<td>2036</td>
<td>16-May-04</td>
<td>Dartonik</td>
</tr>
<tr>
<td>+</td>
<td>2037</td>
<td>16-May-04</td>
<td>Summer Lake</td>
</tr>
<tr>
<td>+</td>
<td>2038</td>
<td>16-May-04</td>
<td>Trydon</td>
</tr>
</tbody>
</table>

**Table name: DW_LINE**

<table>
<thead>
<tr>
<th>INV_NUM</th>
<th>LINE_NUM</th>
<th>PROD_DESCRIPTION</th>
<th>LINE_PRICE</th>
<th>LINE_QUANTITY</th>
<th>LINE_AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2034</td>
<td>1</td>
<td>Optical Mouse</td>
<td>$45.00</td>
<td>20</td>
<td>$900.00</td>
</tr>
<tr>
<td>2034</td>
<td>2</td>
<td>Wireless RF remote and laser pointer</td>
<td>$50.00</td>
<td>10</td>
<td>$600.00</td>
</tr>
<tr>
<td>2035</td>
<td>1</td>
<td>Everlast Hard Drive, 60 GB</td>
<td>$200.00</td>
<td>6</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>2036</td>
<td>1</td>
<td>Optical Mouse</td>
<td>$45.00</td>
<td>30</td>
<td>$1,350.00</td>
</tr>
<tr>
<td>2037</td>
<td>1</td>
<td>Optical Mouse</td>
<td>$45.00</td>
<td>10</td>
<td>$450.00</td>
</tr>
<tr>
<td>2037</td>
<td>2</td>
<td>Roadster 50KB Ext. Modern</td>
<td>$120.00</td>
<td>5</td>
<td>$600.00</td>
</tr>
<tr>
<td>2037</td>
<td>3</td>
<td>Everlast Hard Drive, 60 GB</td>
<td>$205.00</td>
<td>10</td>
<td>$2,050.00</td>
</tr>
<tr>
<td>2038</td>
<td>1</td>
<td>NoTech Speaker Set</td>
<td>$50.00</td>
<td>8</td>
<td>$400.00</td>
</tr>
</tbody>
</table>

**Multidimensional View of Sales**

<table>
<thead>
<tr>
<th>Customer Dimension</th>
<th>15-May-04</th>
<th>16-May-04</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dartonik</td>
<td>$1,400.00</td>
<td>$1,350.00</td>
<td>$2,750.00</td>
</tr>
<tr>
<td>Summer Lake</td>
<td>$1,800.00</td>
<td>$3,100.00</td>
<td>$4,900.00</td>
</tr>
<tr>
<td>Trydon</td>
<td></td>
<td>$400.00</td>
<td>$400.00</td>
</tr>
<tr>
<td>Totals</td>
<td>$3,200.00</td>
<td>$4,850.00</td>
<td>$8,050.00</td>
</tr>
</tbody>
</table>

Sales are located in the intersection of a customer row and time column. Aggregations are provided for both dimensions.
Integration of OLAP with a Spreadsheet Program

**Figure 12.5** Integration of OLAP with a Spreadsheet Program
OLAP Client/Server Architecture

FIGURE 12.6 OLAP CLIENT SERVER ARCHITECTURE

OLAP SYSTEM

THE OLAP SYSTEM EXHIBITS ...

- Client/Server architecture
- Easy-to-use GUI
  - Dimensional presentation
  - Dimensional modeling
  - Dimensional analysis
- Multidimensional data
  - Analysis
  - Manipulation
  - Structure
- Database support
  - Data warehouse
  - Operational DB
  - Relational
  - Multidimensional

DATA WAREHOUSE

- Integrated
- Subject-oriented
- Time-variant
- Nonvolatile

- Dimensional
- Aggregated
- Very large DB

OPERATIONAL DATA

- Drill-down
- Roll-up
- Detailed
OLAP Server Arrangement

FIGURE 12.7  OLAP SERVER ARRANGEMENT

OLAP System

Data warehouse
- Integrated
- Subject-oriented
- Time-variant
- Nonvolatile

Operational data

Shared OLAP “engine”

Analytical processing logic

Data-processing logic

The OLAP “engine” provides a front end to the data warehouse

OLAP GUI
Excel plug-in

OLAP GUI
Lotus plug-in

OLAP GUI
Query tool plug-in

OLAP GUI
Multiple users access OLAP engine
OLAP Server with Multidimensional Data Store Arrangement

Figure 12.8 OLAP Server with Multidimensional Data Store Arrangement

OLAP System

Operational data

Operational data when drill-down, detailed data are needed

Shared OLAP “engine”

Analytical processing logic

Data-processing logic

Data warehouse

Multidimensional data

Multiple users access OLAP engine
OLAP Server With Local Mini Data Marts

![Diagram of OLAP Server With Local Mini Data Marts](image)

**Figure 12.9** OLAP Server With Local Mini Data Marts

- **Operational data**
- **Data warehouse**
- **Multidimensional data**
- **Shared OLAP “engine”**
- **Analytical processing logic**
- **Multiple users access OLAP engine**
- **Data extracted from the data warehouse provide faster processing**

Local data marts:
- Customers
- Marketing
- Production
- Vendors

OLAP System components:
- OLAP GUI
Typical ROLAP Client/Server Architecture

An RDBMS is used to manage both data warehouse and operational data. The RDBMS executes data requests and sends back the data sets.

ROLAP server interprets end-user requests and builds complex SQL queries required to access the data warehouse. If an end user requests a drill-down operation, the ROLAP server builds the required SQL code to access the operational database.

GUI front end runs on client computer and passes data-analysis requests to the ROLAP server. The GUI receives data replies from the ROLAP server and formats them according to the end user’s presentation needs.
MOLAP Client/Server Architecture

**Figure 12.11 MOLAP Client/Server Architecture**

MOLAP System

- Multidimensional database
  - MDBMS
  - Data cube

- MOLAP server
  - MOLAP analytical processing logic
  - MOLAP data-processing logic

- Data cube is created within predefined dimensions.
- MOLAP engine receives data requests from end users and translates them into data cube requests that are passed to the MDBMS.

- RDBMS
  - Operational data
  - Data warehouse data

- MOLAP GUI
  - MOLAP GUI
  - MOLAP GUI
  - MOLAP GUI

- MOLAP GUI allows end users to interact with the MOLAP server and request data for analysis.
# Relational vs. Multidimensional OLAP

## Table 12.8 Relational vs. Multidimensional OLAP

<table>
<thead>
<tr>
<th>CHARACTERISTIC</th>
<th>ROLAP</th>
<th>MOLAP</th>
</tr>
</thead>
</table>
| Schema         | Uses star schema  
                 Additional dimensions can be added dynamically | Uses data cubes  
                 Additional dimensions require re-creation of the data cube |
| Database size  | Medium to large | Small to medium |
| Architecture   | Client/server  
                 Standards-based  
                 Open | Client/server  
                 Proprietary |
| Access         | Supports ad hoc requests  
                 Unlimited dimensions | Limited to predefined dimensions |
| Resources      | High | Very high |
| Flexibility    | High | Low |
| Scalability    | High | Low |
| Speed          | Good with small data sets;  
                 average for medium to large data sets | Faster for small to medium data sets;  
                 average for large data sets |
Models & Operators

• Data Models
  – relations
  – stars & snowflakes
  – cubes

• Operators
  – slice & dice
  – roll-up, drill down
  – pivoting
  – other
### Star

<table>
<thead>
<tr>
<th>product</th>
<th>prodId</th>
<th>name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>nut</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>store</th>
<th>storeId</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>nyc</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>sfo</td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>la</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sale</th>
<th>orderId</th>
<th>date</th>
<th>custId</th>
<th>prodId</th>
<th>storeId</th>
<th>qty</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>o100</td>
<td>1/7/97</td>
<td>53</td>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>o102</td>
<td>2/7/97</td>
<td>53</td>
<td>p2</td>
<td>c1</td>
<td>2</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>3/8/97</td>
<td>111</td>
<td>p1</td>
<td>c3</td>
<td>5</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer</th>
<th>custId</th>
<th>name</th>
<th>address</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>joe</td>
<td>10 main</td>
<td>sfo</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>fred</td>
<td>12 main</td>
<td>sfo</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>sally</td>
<td>80 willow</td>
<td>la</td>
<td></td>
</tr>
</tbody>
</table>
Star Schema

product
  prodId
  name
  price

sale
  orderId
  date
  custId
  prodId
  storeId
  qty
  amt

store
  storeId
  city

customer
  custId
  name
  address
  city
Terms

- Fact table
- Dimension tables
- Measures
Star Schemas

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

1. **Fact table**: a very large accumulation of facts such as sales.
   - Often “insert-only.”

2. **Dimension tables**: smaller, generally static information about the entities involved in the facts.
Dimension Hierarchies

- store
- city
- region

\[\text{\textbf{sType}}\]

<table>
<thead>
<tr>
<th>store</th>
<th>storeId</th>
<th>cityId</th>
<th>tId</th>
<th>mgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5</td>
<td>sfo</td>
<td>t1</td>
<td>joe</td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td>sfo</td>
<td>t2</td>
<td>fred</td>
<td></td>
</tr>
<tr>
<td>s9</td>
<td>la</td>
<td>t1</td>
<td>nancy</td>
<td></td>
</tr>
</tbody>
</table>

\[\text{\textbf{sType}} \quad \text{\textbf{tld}} \quad \text{\textbf{size}} \quad \text{\textbf{location}}\]

- sType: t1 small downtown
- sType: t2 large suburbs

\[\text{\textbf{city}} \quad \text{\textbf{cityld}} \quad \text{\textbf{pop}} \quad \text{\textbf{regId}}\]

- city: sfo 1M north
- city: la 5M south

\[\text{\textbf{region}} \quad \text{\textbf{regId}} \quad \text{\textbf{name}}\]

- region: north cold region
- region: south warm region

\[\rightarrow \text{snowflake schema}\]

\[\rightarrow \text{constellations}\]
Review

• DSS, Data Warehouse, OLAP
• What is the architecture of Data Warehouse?
• What is the difference between operational data (OLTP) and Data Warehouse (OLAP) data
Review-2

• What is the database schema in data warehouse (OLAP)?
• What are fact tables, dimension tables?
• What are dimensional attributes and measure (dependent) attributes
**Cube**

**Fact table view:**

<table>
<thead>
<tr>
<th>sale</th>
<th>proddId</th>
<th>storeId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**Multi-dimensional cube:**

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

dimensions = 2
### 3-D Cube

**Fact table view:**

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Multi-dimensional cube:**

```plaintext
day 1

<table>
<thead>
<tr>
<th>prodId</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

day 2

<table>
<thead>
<tr>
<th>prodId</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

dimensions = 3
ROLAP vs. MOLAP

• ROLAP: Relational On-Line Analytical Processing
• MOLAP: Multi-Dimensional On-Line Analytical Processing
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.

• 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.
Aggregates

- Add up amounts for day 1
- In SQL: SELECT sum(amt) FROM SALE WHERE date = 1

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

81
Aggregates

- Add up amounts by day
- In SQL: SELECT date, sum(amt) FROM SALE GROUP BY date

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ans</th>
<th>date</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>48</td>
</tr>
</tbody>
</table>
Another Example

- Add up amounts by day, product
- In SQL: SELECT date, sum(amt) FROM SALE GROUP BY date, prodId

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>1</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>1</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>2</td>
<td>48</td>
<td></td>
</tr>
</tbody>
</table>

rollup ➔

drill-down ←
Aggregates

- Operators: sum, count, max, min, median, ave
- “Having” clause
- Using dimension hierarchy
  - average by region (within store)
  - maximum by month (within date)
# Cube Aggregation

## Example: computing sums

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td>8</td>
<td>50</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>67</td>
<td>12</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>110</td>
</tr>
<tr>
<td>p2</td>
<td>19</td>
</tr>
</tbody>
</table>

---

day 1

day 2

---

rollup

drill-down
Cube Operators

day 1

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

sale(c2,p2,*)

day 2

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

sum

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>67</td>
<td>12</td>
<td>50</td>
</tr>
</tbody>
</table>

129

sale(*,*,*)

sale(c1,*,*)

sale(129)
Extended Cube

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>56</td>
<td>4</td>
<td>50</td>
<td>110</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td>19</td>
<td>129</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td>19</td>
<td>129</td>
</tr>
</tbody>
</table>

sale(*,p2,*)

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td>50</td>
<td>62</td>
<td>48</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td>19</td>
<td>48</td>
</tr>
<tr>
<td>*</td>
<td>23</td>
<td>8</td>
<td>50</td>
<td>81</td>
</tr>
</tbody>
</table>
Aggregation Using Hierarchies

(customer c1 in Region A; customers c2, c3 in Region B)
## Pivoting

### Fact table view:

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

### Multi-dimensional cube:

**Day 1**

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

**Day 2**

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Query & Analysis Tools

- Query Building
- Report Writers (comparisons, growth, graphs,…)
- Spreadsheet Systems
- Web Interfaces
- Data Mining
Other Operations

• Time functions
  – e.g., time average

• Computed Attributes
  – e.g., commission = sales * rate

• Text Queries
  – e.g., find documents with words X AND B
  – e.g., rank documents by frequency of words X, Y, Z
Data Mining

• Decision Trees
• Clustering
• Association Rules
SQL:1999 additional supports

- Extended Aggregation
- Ranking
- Windowing
The **cube** operation computes union of **group by**’s on every subset of the specified attributes

E.g. consider the query

```sql
select item-name, color, size, sum(number)
from sales
group by cube(item-name, color, size)
```

This computes the union of eight different groupings of the `sales` relation:

{ (item-name, color, size), (item-name, color),
  (item-name, size),  (color, size),
  (item-name),        (color),
  (size),             ( ) }

where ( ) denotes an empty **group by** list.

For each grouping, the result contains the null value for attributes not present in the grouping.
Extended Aggregation (Cont.)

- Relational representation of cross-tab that we saw earlier, but with *null* in place of *all*, can be computed by
  
  ```sql
  select item-name, color, sum(number)
  from sales
  group by cube(item-name, color)
  ```

- The function `grouping()` can be applied on an attribute
  - Returns 1 if the value is a null value representing all, and returns 0 in all other cases.
  
  ```sql
  select item-name, color, size, sum(number),
  grouping(item-name) as item-name-flag,
  grouping(color) as color-flag,
  grouping(size) as size-flag,
  from sales
  group by cube(item-name, color, size)
  ```

- Can use the function `decode()` in the `select` clause to replace such nulls by a value such as *all*
  - E.g. replace *item-name* in first query by
    ```sql
    decode( grouping(item-name), 1, ‘all’, item-name)
    ```
Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes

- E.g.

  ```sql
  select item-name, color, size, sum(number)
  from sales
  group by rollup(item-name, color, size)
  ```

  Generates union of four groupings:

  ```sql
  { (item-name, color, size), (item-name, color), (item-name), ( ) } 
  ```

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.

- E.g., suppose table **itemcategory(item-name, category)** gives the category of each item. Then

  ```sql
  select category, item-name, sum(number)
  from sales, itemcategory
  where sales.item-name = itemcategory.item-name
  group by rollup(category, item-name)
  ```

  would give a hierarchical summary by *item-name* and by *category*. 
**Extended Aggregation (Cont.)**

- Multiple rollups and cubes can be used in a single group by clause
  - Each generates set of group by lists, cross product of sets gives overall set of group by lists
- E.g.,

  ```sql
  select item-name, color, size, sum(number)
  from sales
  group by rollup(item-name), rollup(color, size)
  ```

  generates the groupings

  \[
  \{ \text{item-name, ()} \} \times \{(\text{color, size}), (\text{color}), (\)} \]

  \[
  = \{ (\text{item-name, color, size}), (\text{item-name, color}), (\text{item-name}),
          (\text{color, size}), (\text{color}), (\) \}
  \]
Ranking

- Ranking is done in conjunction with an order by specification.
- Given a relation student-marks(student-id, marks) find the rank of each student.

\[
\text{select } \text{student-id, rank}( ) \text{ over (order by marks desc) as s-rank}
\text{from student-marks}
\]

- An extra \textbf{order by} clause is needed to get them in sorted order

\[
\text{select student-id, rank ( ) over (order by marks desc) as s-rank}
\text{from student-marks}
\text{order by s-rank}
\]

- Ranking may leave gaps: e.g. if 2 students have the same top mark, both have rank 1, and the next rank is 3
  
  ★ \textbf{dense_rank} does not leave gaps, so next dense rank would be 2
Ranking (Cont.)

- Ranking can be done within partition of the data.
- “Find the rank of students within each section.”

```sql
select student-id, section,
    rank () over (partition by section order by marks desc) as sec-rank
from student-marks, student-section
where student-marks.student-id = student-section.student-id
order by section, sec-rank
```

- Multiple `rank` clauses can occur in a single `select` clause.
- Ranking is done after applying `group by` clause/aggregation.
Other ranking functions:

- **percent_rank** (within partition, if partitioning is done)
- **cume_dist** (cumulative distribution)
  - fraction of tuples with preceding values
- **row_number** (non-deterministic in presence of duplicates)

SQL: 1999 permits the user to specify **nulls first** or **nulls last**

```sql
select student-id,
       rank () over (order by marks desc nulls last) as s-rank
from student-marks
```
For a given constant \( n \), the ranking the function \( ntile(n) \) takes the tuples in each partition in the specified order, and divides them into \( n \) buckets with equal numbers of tuples.

E.g.:

```sql
select threetile, sum(salary)
from (   select salary, ntile(3) over (order by salary) as threetile
         from employee) as s
group by threetile
```
Windowing

- Used to smooth out random variations.
- E.g.: moving average: “Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day”

**Window specification** in SQL:
- Given relation `sales(date, value)`
  ```sql
  select date, sum(value) over 
  (order by date between rows 1 preceding and 1 following) 
  from sales
  ```

Examples of other window specifications:
- between rows unbounded preceding and current
- rows unbounded preceding
- range between 10 preceding and current row
  - All rows with values between current row value –10 to current value
- range interval 10 day preceding
  - Not including current row
Windowing (Cont.)

- Can do windowing within partitions
- E.g. Given a relation `transaction (account-number, date-time, value)`, where value is positive for a deposit and negative for a withdrawal

  “Find total balance of each account after each transaction on the account”

  ```sql
  select account-number, date-time,
         sum (value) over
            (partition by account-number
             order by date-time
             rows unbounded preceding)
         as balance
  from transaction
  order by account-number, date-time
  ```
Review

• Data Models
  – Stars
  – Cubes
  – ROLAP/MOLAP

• Operators
  – slice & dice
  – roll-up, drill down
  – pivoting
  – data cube
Review SQL’99

• Advanced Aggregation
  – Data Cube
  – Roll up
  – Rank
  – Window
SQL Examples

1. ```sql
   select item-name, color, size, sum(number)
   from sales
   group by cube(item-name, color, size)
```

2. ```sql
   select item-name, color, size, sum(number)
   from sales
   group by rollup(item-name, color, size)
```

3. ```sql
   select item-name, color, size, sum(number)
   from sales
   group by rollup(item-name), rollup(color, size)
```
SQL Example

1. select student-id, rank () over (order by marks desc) as s-rank
   from student-marks
   order by s-rank

2. select student-id, section,
   rank () over (partition by section order by marks desc)
   as sec-rank
   from student-marks, student-section
   where student-marks.student-id = student-section.student-id
   order by section, sec-rank

3. select date, sum(value) over
   (order by date between rows 1 preceding and 1 following)
   from sales

4. select account-number, date-time,
   sum(value) over
   (partition by account-number
    order by date-time
    rows unbounded preceding)
   as balance
   from transaction
   order by account-number, date-time
OLAP Implementation Techniques

- View Materialization
- Indexing
  - Bitmap Indexing
  - Join Indexing
View Materialization

• What is View?
• Why do need to materialize views?
• What to materialize?
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 Attrs.

Fact Table - Sales

Dimension Attrs.

Dimension Table (Drinkers)

Dependent Attrs.

Dimension Table (Beers)

Dimension Table (etc.)
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;
```
Example: In SQL

```sql
SELECT bar, beer, SUM(price)
FROM Sales NATURAL JOIN Bars
    NATURAL JOIN Beers
WHERE addr = 'Palo Alto' 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 Palo-Alto 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:

  SELECT bar, beer, sales
  FROM BABMS
  WHERE addr = 'Palo Alto' AND
       manf = 'Anheuser-Busch';
What to Materialize?

• Store in warehouse results useful for common queries

• Example:

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>day 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>day 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

total sales

materialize
Materialization Factors

• Type/frequency of queries
• Query response time
• Storage cost
• Update cost
use greedy algorithm to decide what to materialize
Dimension Hierarchies

```
all
 state
 city
```

<table>
<thead>
<tr>
<th>cities</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td></td>
<td>CA</td>
</tr>
<tr>
<td>c2</td>
<td></td>
<td>NY</td>
</tr>
</tbody>
</table>
Dimension Hierarchies

not all arcs shown...
Interesting Hierarchy

1. all
2. years
   - weeks
   - quarters
   - months
3. days

<table>
<thead>
<tr>
<th>time</th>
<th>day</th>
<th>week</th>
<th>month</th>
<th>quarter</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

conceptual dimension table
Index Structures

- Traditional Access Methods
  - B-trees, hash tables, R-trees, grids, ...
- Popular in Warehouses
  - inverted lists
  - bit map indexes
  - join indexes
  - text indexes
Inverted Lists

<table>
<thead>
<tr>
<th>rld</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>r4</td>
<td>joe</td>
<td>20</td>
</tr>
<tr>
<td>r18</td>
<td>fred</td>
<td>20</td>
</tr>
<tr>
<td>r19</td>
<td>sally</td>
<td>21</td>
</tr>
<tr>
<td>r34</td>
<td>nancy</td>
<td>20</td>
</tr>
<tr>
<td>r35</td>
<td>tom</td>
<td>20</td>
</tr>
<tr>
<td>r36</td>
<td>pat</td>
<td>25</td>
</tr>
<tr>
<td>r5</td>
<td>dave</td>
<td>21</td>
</tr>
<tr>
<td>r41</td>
<td>jeff</td>
<td>26</td>
</tr>
</tbody>
</table>

age index

inverted lists

data records
Using Inverted Lists

- Query:
  - Get people with age = 20 and name = “fred”
- List for age = 20: r4, r18, r34, r35
- List for name = “fred”: r18, r52
- Answer is intersection: r18
Bit Maps

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>joe</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>fred</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sally</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>nancy</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>tom</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>pat</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>dave</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>jeff</td>
<td>26</td>
</tr>
</tbody>
</table>
Using Bit Maps

- Query:
  - Get people with age = 20 and name = “fred”
- List for age = 20: 1101100000
- List for name = “fred”: 0100000001
- Answer is intersection: 010000000000

- Good if domain cardinality small
- Bit vectors can be compressed
Join

• “Combine” SALE, PRODUCT relations
• In SQL: SELECT * FROM SALE, PRODUCT

<table>
<thead>
<tr>
<th>sale</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>id</th>
<th>name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>nut</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>joinTb</th>
<th>prodId</th>
<th>name</th>
<th>price</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>nut</td>
<td>5</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>nut</td>
<td>5</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Join Indexes

join index

<table>
<thead>
<tr>
<th>product</th>
<th>id</th>
<th>name</th>
<th>price</th>
<th>jIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p1</td>
<td>bolt</td>
<td>10</td>
<td>r1, r3, r5, r6</td>
</tr>
<tr>
<td>p2</td>
<td>p2</td>
<td>nut</td>
<td>5</td>
<td>r2, r4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sale</th>
<th>rId</th>
<th>prodId</th>
<th>storeId</th>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>p1</td>
<td>c1</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>p2</td>
<td>c1</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>p1</td>
<td>c3</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>p2</td>
<td>c2</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>p1</td>
<td>c1</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td>p1</td>
<td>c2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Current State of Industry

• Extraction and integration done off-line
  – Usually in large, time-consuming, batches

• Everything copied at warehouse
  – Not selective about what is stored
  – Query benefit vs storage & update cost

• Query optimization aimed at OLTP
  – High throughput instead of fast response
  – Process whole query before displaying anything
Future Directions

- Better performance
- Larger warehouses
- Easier to use
- What are companies & research labs working on?
Research (1)

- Incremental Maintenance
- Data Consistency
- Data Expiration
- Recovery
- Data Quality
- Error Handling (Back Flush)
Research (2)

- Rapid Monitor Construction
- Temporal Warehouses
- Materialization & Index Selection
- Data Fusion
- Data Mining
- Integration of Text & Relational Data
Conclusions

• Massive amounts of data and complexity of queries will push limits of current warehouses

• Need better systems:
  – easier to use
  – provide quality information