Data Warehousing and Decision Support

Chapter 25
What is Data Warehouse?

- Defined in many different ways, but not rigorously.
  - A decision support database that is maintained separately from the organization’s operational database
  - Supports information processing by providing a solid platform of consolidated, historical data for analysis.
  - “A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision-making process.”—W. H. Inmon
- Data warehousing:
  - The process of constructing and using data warehouses
Data Warehouse—Subject-Oriented

- Organized around major subjects, such as customer, product, sales.
- Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing.
- Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.
Data Warehouse—Integrated

- Constructed by integrating multiple, heterogeneous data sources
  - relational databases, flat files, on-line transaction records
- Data cleaning and data integration techniques are applied.
  - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources
    - E.g., Hotel price: currency, tax, breakfast covered, etc.
  - When data is moved to the warehouse, it is converted.
Data Warehouse—Time Variant

- The time horizon for the data warehouse is significantly longer than that of operational systems.
  - Operational database: current value data.
  - Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)
- Every key structure in the data warehouse
  - Contains an element of time, explicitly or implicitly
  - But the key of operational data may or may not contain “time element”.
Data Warehouse—Non-Volatile

- A **physically separate store** of data transformed from the operational environment.
- Operational **update of data does not occur** in the data warehouse environment.
  - Does not require transaction processing, recovery, and concurrency control mechanisms
  - Requires only two operations in data accessing:
    - *initial loading of data* and *access of data*.
Data Warehouse vs. Heterogeneous DBMS

- Traditional heterogeneous DB integration:
  - Build wrappers/mediators on top of heterogeneous databases
  - Query driven approach
    - When a query is posed to a client site, a meta-dictionary is used to translate the query into queries appropriate for individual heterogeneous sites involved, and the results are integrated into a global answer set

- Data warehouse: update-driven, high performance
  - Information from heterogeneous sources is integrated in advance and stored in warehouses for direct query and analysis
Data Warehouse vs. Operational DBMS

- OLTP (on-line transaction processing)
  - Major task of traditional relational DBMS
  - Day-to-day operations: purchasing, inventory, banking, manufacturing, payroll, registration, accounting, etc.

- OLAP (on-line analytical processing)
  - Major task of data warehouse system
  - Data analysis and decision making

- Distinct features (OLTP vs. OLAP):
  - User and system orientation: customer vs. market
  - Data contents: current, detailed vs. historical, consolidated
  - Database design: ER + application vs. star + subject
  - View: current, local vs. evolutionary, integrated
  - Access patterns: update vs. read-only but complex queries
## OLTP vs. OLAP

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>users</strong></td>
<td>clerk, IT professional</td>
<td>knowledge worker</td>
</tr>
<tr>
<td><strong>function</strong></td>
<td>day to day operations</td>
<td>decision support</td>
</tr>
<tr>
<td><strong>DB design</strong></td>
<td>application-oriented</td>
<td>subject-oriented</td>
</tr>
<tr>
<td><strong>data</strong></td>
<td>current, up-to-date detailed, flat relational isolated</td>
<td>historical, summarized, multidimensional integrated, consolidated</td>
</tr>
<tr>
<td><strong>usage</strong></td>
<td>repetitive</td>
<td>ad-hoc</td>
</tr>
<tr>
<td><strong>access</strong></td>
<td>read/write index/hash on prim. key</td>
<td>lots of scans</td>
</tr>
<tr>
<td><strong>unit of work</strong></td>
<td>short, simple transaction</td>
<td>complex query</td>
</tr>
<tr>
<td><strong># records accessed</strong></td>
<td>tens</td>
<td>millions</td>
</tr>
<tr>
<td><strong>#users</strong></td>
<td>thousands</td>
<td>hundreds</td>
</tr>
<tr>
<td><strong>DB size</strong></td>
<td>100MB-GB</td>
<td>100GB-TB</td>
</tr>
<tr>
<td><strong>metric</strong></td>
<td>transaction throughput</td>
<td>query throughput, response</td>
</tr>
</tbody>
</table>
Why Separate Data Warehouse?

- High performance for both systems
  - DBMS—tuned for OLTP: access methods, indexing, concurrency control, recovery
  - Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation.

- Different functions and different data:
  - **missing data**: Decision support requires historical data which operational DBs do not typically maintain
  - **data consolidation**: DW requires consolidation (aggregation, summarization) of data from heterogeneous sources
  - **data quality**: different sources typically use inconsistent data representations, codes and formats which have to be reconciled
Conceptual Modeling of Data Warehouses

- Modeling data warehouses: dimensions & measures
  - **Star schema**: A fact table in the middle connected to a set of dimension tables
  - **Snowflake schema**: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
  - **Fact constellations**: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called **galaxy schema** or fact constellation
Example of Star Schema

**Sales** Fact Table:
- **time_key**
- **item_key**
- **branch_key**
- **location_key**
- **units_sold**
- **dollars_sold**
- **avg_sales**

**Dimensions**:
- **time**:
  - time_key
  - day
  - day_of_the_week
  - month
  - quarter
  - year
- **branch**:
  - branch_key
  - branch_name
  - branch_type
- **item**:
  - item_key
  - item_name
  - brand
  - type
  - supplier_type
- **location**:
  - location_key
  - street
  - city
  - province_or_street
  - country
Example of Snowflake Schema

**Time**
- time_key
- day
- day_of_the_week
- month
- quarter
- year

**Location**
- location_key
- street
- city_key

**Branch**
- branch_key
- branch_name
- branch_type

**Item**
- item_key
- item_name
- brand
- type
- supplier_key

**Supplier**
- supplier_key
- supplier_type

**Sales Fact Table**
- time_key
- item_key
- branch_key
- location_key
- units_sold
- dollars_sold
- avg_sales

**Measures**
- city
  - city_key
  - province_or_street
  - country
Example of Fact Constellation

**Sales Fact Table**
- time_key
- item_key
- branch_key
- location_key
- units_sold
- dollars_sold
- avg_sales

**Measures**
- branch_key
- branch_name
- branch_type

**Shipping Fact Table**
- time_key
- item_key
- shipper_key
- from_location
- to_location
- dollars_cost
- units_shipped

**Item**
- item_key
- item_name
- brand
- type
- supplier_type

**Location**
- location_key
- street
- city
- province_or_street
- country

**Shipper**
- shipper_key
- shipper_name
- location_key
- shipper_type
A Concept Hierarchy: Dimension (location)

- All
- Region
- Country
- City
- Office

Hierarchy:
- All
  - Europe
    - Germany
      - Frankfurt
    - Spain
    - ... (其他国家)
  - North America
    - Canada
      - Vancouver
      - Toronto
      - L. Chan
      - M. Wind
    - ... (其他城市)

From Tables and Spreadsheets to Data Cubes

- A data warehouse is based on a multidimensional data model which views data in the form of a data cube.
- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions.
  - Dimension tables, such as item (item_name, brand, type), or time(day, week, month, quarter, year).
  - Fact table contains measures (such as dollars_sold) and keys to each of the related dimension tables.
- In data warehousing literature, an n-D base cube is called a base cuboid. The top most 0-D cuboid, which holds the highest-level of summarization, is called the apex cuboid. The lattice of cuboids forms a data cube.
Multidimensional Data

- Sales volume as a function of product, month, and region

Dimensions: Product, Location, Time
Hierarchical summarization paths
A Sample Data Cube

Total annual sales of TV in U.S.A.
Cuboids Corresponding to the Cube

- 0−D (apex) cuboid
- 1−D cuboids
- 2−D cuboids
- 3−D (base) cuboid
Typical OLAP Operations

- **Roll up (drill-up):** summarize data
  - *by climbing up hierarchy or by dimension reduction*
- **Drill down (roll down):** reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- **Slice and dice:**
  - *project and select*
- **Pivot (rotate):**
  - *aggregation on selected dimensions*
- **Other operations**
  - *drill across: involving (across) more than one fact table*
  - *drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*
Multi-Tiered Architecture

Data Sources

Operational DBs

other sources

Metadata

Monitor & Integrator

Extract Transform Load Refresh

Data Warehouse

Data Marts

OLAP Server

Serve

Analysis Query Reports Data mining

OLAP Engine

Front-End Tools

Data Storage

November 30, 2016
Three Data Warehouse Models

- **Enterprise warehouse**
  - collects all of the information about subjects spanning the entire organization

- **Data Mart**
  - a subset of corporate-wide data that is of value to a specific groups of users. Its scope is confined to specific, selected groups, such as marketing data mart
    - Independent vs. dependent (directly from warehouse) data mart

- **Virtual warehouse**
  - A set of views over operational databases
  - Only some of the possible summary views may be materialized
OLAP Server Architectures

- **Relational OLAP (ROLAP)**
  - Use relational or extended-relational DBMS to store and manage warehouse data and OLAP middleware to support missing pieces
  - Include optimization of DBMS backend, implementation of aggregation navigation logic, and additional tools and services
  - Greater scalability

- **Multidimensional OLAP (MOLAP)**
  - Array-based multidimensional storage engine (sparse matrix techniques)
  - Fast indexing to pre-computed summarized data

- **Hybrid OLAP (HOLAP)**
  - User flexibility, e.g., low level: relational, high-level: array
  - Specialized SQL servers
    - Specialized support for SQL queries over star/snowflake schemas
Efficient Data Cube Computation

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube?

\[ 2^n \]
Problem: How to Implement Data Cube Efficiently?

- Physically materialize the whole data cube
  - Space consuming in storage and time consuming in construction
  - Indexing overhead

- Materialize nothing
  - No extra space needed but unacceptable response time

- Materialize only part of the data cube
  - Intuition: precompute frequently-asked queries?
  - However: each cell of data cube is an aggregation, the value of many cells are dependent on the values of other cells in the data cube
  - A better approach: materialize queries which can help answer many other queries quickly
Motivating example

- Assume the data cube:
  - Stored in a relational DB (MDDB is not very scalable)
  - Different cuboids are assigned to different tables
  - The cost of answering a query is proportional to the number of rows examined
- Use TPC-D decision-support benchmark
  - Attributes: part, supplier, and customer
  - Measure: total sales
  - 3-D data cube: cell \((p, s ,c)\)
Motivating example (cont.)

- **Hypercube lattice**: the eight views (cuboids) constructed by grouping on some of *part, supplier, and customer*

Finding total *sales* grouped by *part*

- Processing 6 million rows if cuboid *pc* is materialized
- Processing 0.2 million rows if cuboid *p* is materialized
- Processing 0.8 million rows if cuboid *ps* is materialized
Motivating example (cont.)

How to find a good set of queries?

- How many views must be materialized to get reasonable performance?
- Given space $S$, what views should be materialized to get the minimal average query cost?
- If we are willing to tolerate an X% degradation in average query cost from a fully materialized data cube, how much space can we save over the fully materialized data cube?
The dependence relation on queries:

- \( Q_1 \preceq Q_2 \) iff \( Q_1 \) can be answered using only the results of query \( Q_2 \) (\( Q_1 \) is dependent on \( Q_2 \)).

In which

- \( \preceq \) is a partial order, and
- There is a top element, a view upon which is dependent (base cuboid)

Example:

- \((part) \preceq (part, customer)\)
- \((part) \not\preceq (customer)\) and \((customer) \not\preceq (part)\)
The linear cost model

- For \( <L, \leq> \), \( Q \leq Q_A \), \( C(Q) \) is the number of rows in the table for that query \( Q_A \) used to compute \( Q \)
  - This linear relationship can be expressed as:
    \[
    T = m \times S + c
    \]
    (\( m: \) time/size ratio; \( c: \) query overhead; \( S: \) size of the view)
- Validation of the model using TPC-D data:

<table>
<thead>
<tr>
<th>Source</th>
<th>Size</th>
<th>Time (sec.)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>From cell itself</td>
<td>1</td>
<td>2.07</td>
<td>not applicable</td>
</tr>
<tr>
<td>From view (supplier)</td>
<td>10,000</td>
<td>2.38</td>
<td>.000031</td>
</tr>
<tr>
<td>From view (part, supplier)</td>
<td>800,000</td>
<td>20.77</td>
<td>.000023</td>
</tr>
<tr>
<td>From view (part, supplier, customer)</td>
<td>6,000,000</td>
<td>226.23</td>
<td>.000037</td>
</tr>
</tbody>
</table>

Growth of query response time with size of view
The benefit of a materialized view

- Denote the benefit of a materialized view \( v \), relative to some set of views \( S \), as \( B(v, S) \).
- For each \( w \prec v \), define \( B_w \) by:
  - Let \( C(v) \) be the cost of view \( v \).
  - Let \( u \) be the view of least cost in \( S \) such that \( w \prec u \) (such \( u \) must exist).
  - \( B_w = C(u) - C(v) \) if \( C(v) < C(u) \)
    = 0 \quad \text{if} \quad C(v) \geq C(u) \)
  - \( B_w \) is the benefit that it can obtain from \( v \).
- Define \( B(v, S) = \sum_{w \prec v} B_w \) which means how \( v \) can improve the cost of evaluating views, including itself.
The greedy algorithm

- **Objective**
  - Assume materializing a fixed number of views, regardless of the space they use
  - How to minimize the average time taken to evaluate a view?
  - The greedy algorithm for materializing a set of $k$ views

$$S = \{ \text{top view} \};$$

```plaintext
for i=1 to k do begin
    select that view $v$ not in $S$ such that $B(v,S)$ is maximized;
    $S = S \cup \{ v \};$
end;
resulting $S$ is the greedy selection;
```

- **Performance:** Greedy/Optimal $\geq 1 - (1 - 1/k)^k \geq (e - 1) / e$
Greedy algorithm: example 1

Suppose we want to choose three views (k = 3)

Example lattice with space costs

The selection is optimal (reduce cost from 800 to 420)
Suppose $k = 2$

- Greedy algorithm picks $c$ and $b$: benefit = $101 \times 41 + 100 \times 21 = 6241$
- Optimal selection is $b$ and $d$: benefit = $100 \times 41 + 100 \times 41 = 8200$
- However, greedy/optimal = $6241/8200 > 3/4$

A lattice where the greedy algorithm does poorly.
An experiment: how many views should be materialized?

Time and space for the greedy selection for the TPC-D-based example (full materialization is not efficient)

<table>
<thead>
<tr>
<th>Number</th>
<th>Selection</th>
<th>Benefit</th>
<th>Total Time</th>
<th>Total Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$cp$</td>
<td>infinite</td>
<td>72M</td>
<td>6M</td>
</tr>
<tr>
<td>2.</td>
<td>$ns$</td>
<td>24M</td>
<td>48M</td>
<td>6M</td>
</tr>
<tr>
<td>3.</td>
<td>$nt$</td>
<td>12M</td>
<td>36M</td>
<td>6M</td>
</tr>
<tr>
<td>4.</td>
<td>$c$</td>
<td>5.9M</td>
<td>30.1M</td>
<td>6.1M</td>
</tr>
<tr>
<td>5.</td>
<td>$p$</td>
<td>5.8M</td>
<td>24.3M</td>
<td>6.3M</td>
</tr>
<tr>
<td>6.</td>
<td>$cs$</td>
<td>1M</td>
<td>23.3M</td>
<td>11.3M</td>
</tr>
<tr>
<td>7.</td>
<td>$np$</td>
<td>1M</td>
<td>22.3M</td>
<td>16.3M</td>
</tr>
<tr>
<td>8.</td>
<td>$ct$</td>
<td>0.01M</td>
<td>22.3M</td>
<td>22.3M</td>
</tr>
<tr>
<td>9.</td>
<td>$t$</td>
<td>small</td>
<td>22.3M</td>
<td>22.3M</td>
</tr>
<tr>
<td>10.</td>
<td>$n$</td>
<td>small</td>
<td>22.3M</td>
<td>22.3M</td>
</tr>
<tr>
<td>11.</td>
<td>$s$</td>
<td>small</td>
<td>22.3M</td>
<td>22.3M</td>
</tr>
<tr>
<td>12.</td>
<td>none</td>
<td>small</td>
<td>22.3M</td>
<td>22.3M</td>
</tr>
</tbody>
</table>

Greedy order of view selection for TPC-D-based example
Indexing OLAP Data: Bitmap Index

- Index on a particular column
- Each value in the column has a bit vector: bit-op is fast
- The length of the bit vector: # of records in the base table
- The $i$-th bit is set if the $i$-th row of the base table has the value for the indexed column
- not suitable for high cardinality domains

<table>
<thead>
<tr>
<th>Cust</th>
<th>Region</th>
<th>Type</th>
<th>RecID</th>
<th>Asia</th>
<th>Europe</th>
<th>America</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Asia</td>
<td>Retail</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C2</td>
<td>Europe</td>
<td>Dealer</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C3</td>
<td>Asia</td>
<td>Dealer</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C4</td>
<td>America</td>
<td>Retail</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C5</td>
<td>Europe</td>
<td>Dealer</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Index on Region

Index on Type

<table>
<thead>
<tr>
<th>Cust</th>
<th>Region</th>
<th>Type</th>
<th>RecID</th>
<th>Retail</th>
<th>Dealer</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Asia</td>
<td>Retail</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C2</td>
<td>Europe</td>
<td>Dealer</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>Asia</td>
<td>Dealer</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C4</td>
<td>America</td>
<td>Retail</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C5</td>
<td>Europe</td>
<td>Dealer</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Indexing OLAP Data: Join Indices

- Join index: $JI(R\text{-id}, S\text{-id})$ where $R\ (R\text{-id}, \ldots) \bowtie S\ (S\text{-id}, \ldots)$
- Traditional indices map the values to a list of record ids
  - It materializes relational join in JI file and speeds up relational join — a rather costly operation
- In data warehouses, join index relates the values of the dimensions of a start schema to rows in the fact table.
  - E.g. fact table: Sales and two dimensions city and product
    - A join index on city maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
  - Join indices can span multiple dimensions
Summary

- **Data warehouse**
  - A subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision-making process

- A multi-dimensional model of a data warehouse
  - Star schema, snowflake schema, fact constellations
  - A data cube consists of dimensions & measures

- **OLAP** operations: drilling, rolling, slicing, dicing and pivoting

- **OLAP** servers: ROLAP, MOLAP, HOLAP

- Efficient computation of data cubes
  - Partial vs. full vs. no materialization
  - Multiway array aggregation
  - Bitmap index and join index implementations

- Further development of data cube technology
  - Discovery-drive and multi-feature cubes
  - From OLAP to OLAM (on-line analytical mining)