Data Warehouses

Yanlei Diao

Slides Courtesy of R. Ramakrishnan and J. Gehrke
Introduction

- In the late 80s and early 90s, companies began to use their DBMSs for complex, interactive, exploratory analysis of historical data.

Operational Data
(purchase transactions: store, customer, products, sales, etc.)

Decision Making:
- how much of which products to order for which store?
- when to deliver the products?
- benefits of promotional offers?
Data Warehousing

- **Data:**
  - Integrated data spanning long time periods, often augmented with summary information.
  - Large volumes: several terabytes to petabytes common.

- **Queries:**
  - Interactive response time expected for complex queries.
  - Ad-hoc updates uncommon.

- **Data mining**
  - Automated procedure for insight discovery

![Diagram of Data Warehousing Process](image)
Data Warehouses [Chaudhuri & Dayal 97]

- A **data warehouse** (DW) is an organization-wide data repository, used for decision making
  - An integrated enterprise warehouse collects info about all subjects, e.g. customers, products, sales, assets, personnel.
  - The data is used to assist in decision making
    - e.g., how much of which products to order for which stores, when to deliver the products, the benefits of various promotional offers, etc.
  - Analytics is called **On-Line Analytic Processing** (OLAP).
  - OLAP tasks slowed down the *normal operation* of the company, called **On-Line Transaction Processing** (OLTP), leading to separation of DWs from operational DBs.
# OLTP vs OLAP Databases

<table>
<thead>
<tr>
<th>OLTP / Operational / Production</th>
<th>OLAP / Data Warehouse / DSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operate the business / Clerks</td>
<td>Diagnose the business / Managers</td>
</tr>
<tr>
<td>Short queries, small amts of data</td>
<td>Large queries, large amts of data</td>
</tr>
<tr>
<td>Current data</td>
<td>Current and historical data</td>
</tr>
<tr>
<td>Queries change data</td>
<td>Queries are mostly read-only</td>
</tr>
<tr>
<td>• Examples: customer inquiry, order entry</td>
<td>• OLAP, data mining, statistics, visualization, etc.</td>
</tr>
<tr>
<td>Legacy applications, heterogeneous databases</td>
<td>Opposite</td>
</tr>
<tr>
<td>Often distributed</td>
<td>Often integrated and centralized (Warehouse)</td>
</tr>
</tbody>
</table>
Overview of Topics

1. Introduction
   - Operational vs. Warehouse

2. Multidimensional Data
   - Data model & schema

3. Queries
   - OLAP Queries
   - CUBE Operator
   - Window Operator

4. Implementation Algorithms
   - Bitmap Index
   - MOLAP vs ROLAP

5. Materialized Views
   - View definition
   - Query answering using views
   - View selection

6. Constructing a Data Warehouse (ETL)
To support OLAP, warehouse data is often structured multidimensionally, as *measures* and *dimensions*.

- **Measure**: numeric attribute, e.g. sales amount, ROI.
- **Dimension**: attribute categorizing the measure, e.g. product, store, date of sale.

**Star schema:**

- The central *fact table* contains a foreign key for each dimension, plus an attribute for each measure.
- There will also be a *dimension table* for each dimension.
Example Multidimensional Design

This kind of schema, called a **star schema**, is very common in OLAP applications.

Is this a good design of the schema? It avoids redundant storage, but requires joins.
Dimension Hierarchies

- For each dimension, some of the attributes may be organized in a hierarchy:

  **PRODUCT**
  - category
  - pname
  - PID

  **TIME**
  - year
  - quarter
  - week
  - date

  **LOCATION**
  - state
  - city
  - ZIP
Star/Snowflake Schemas

**Star schema**

**Order**
- OrderNo
- OrderDate

**Customer**
- CustomerNo
- CustomerName
- CustomerAddress
- City

**Salesperson**
- SalespersonID
- SalespersonName
- City
- Quota

**Product**
- ProdNo
- ProductName
- ProductDescr
- Category
- CategoryDescr
- UnitPrice

**Fact Table**
- OrderNo
- CustomerNo
- SalespersonID
- ProdNo
- DateKey
- CityName
- Quantity
- TotalPrice

**Date**
- DateKey
- Date
- Month
- Year

**City**
- CityName
- State
- Country

**Snowflake schema**

**Order**
- OrderNo
- OrderDate

**Customer**
- CustomerNo
- CustomerName
- CustomerAddress
- City

**Salesperson**
- SalespersonID
- SalespersonName
- City
- Quota

**Product**
- ProdNo
- ProductName
- ProductDescr
- Category
- CategoryDescr
- UnitPrice

**Category**
- CategoryName
- CategoryDescr

**Fact Table**
- OrderNo
- CustomerNo
- SalespersonID
- ProdNo
- DateKey
- CityName
- Category
- Quantity
- TotalPrice

**Date**
- DateKey
- Date
- Month
- Year

**City**
- CityName
- State
- Country

**Month**
- Month
- Year

**State**
- State
- Country

**Country**
- Country
Star/Snowflake Schemas

- Why normalize?
  - Save space
  - Remove store redundancy and related anomalies
  - If fully normalized, it is a snowflake schema

- Why denormalize?
  - Performance benefits, e.g., avoiding joins

- Which is more important in Data Warehouses?
Examples of Multi-Dimensional Data

- **Purchase** (ProductID, StoreID, DateID, Amt)
  - Product(ID, SKU, size, brand)
  - Store(ID, Address, Sales District, Region, Manager)
  - Date (ID, Week, Month, Holiday, Promotion)

- **Claims** (ProvID, MemblID, ProcedureID, DateID, Cost)
  - Providers(ID, Practice, Address, ZIP, City, State)
  - Members(ID, Contract, Name, Address)
  - Procedure (ID, Name, Type)

- **Telecomm** (CustID, SalesRepID, ServiceID, DateID)
  - SalesRep(ID, Address, Sales District, Region, Manager)
  - Service(ID, Name, Category)
  - ...
MOLAP vs ROLAP

- **ROLAP**: Multidimensional data can be stored as a relation, called ROLAP systems.
  - The main relation, which relates dimensions to a measure (e.g., sales), is the fact table.
  - Each dimension has additional attributes in a dimension table.
  - E.g., Products(pid, locid, timeid, amt)
  - Fact tables are much larger than dimensional tables.

- **MOLAP**: Multidimensional data can be stored physically in a (disk-resident, persistent) array, called MOLAP systems.
**Multidimensional Data Model**

- Collection of numeric *measures*, which depend on a set of *dimensions*.
  - E.g., measure **Amt**, dimensions **Product** (key: pid), **Time** (timeid), and **Location** (locid).

```
<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
</tbody>
</table>
```

Slice locid=1 ➞

![Diagram of multidimensional data model](chart.png)
Overview of Topics

1. Introduction
   - Operational vs. Warehouse

2. Multidimensional Data
   - Data model & schema

3. Queries
   - OLAP Queries
   - CUBE Operator
   - Window Operator

4. Implementation Algorithms
   - Bitmap Index
   - MOLAP vs ROLAP

5. Materialized Views
   - View definition
   - Query answering using views
   - View selection

6. Constructing a Data Warehouse (ETL)
3. OLAP Queries (Ch 25.3)

- Influenced both by SQL and by spreadsheets.
- A common operation is to **aggregate** a measure over one or more dimensions.
  - Find total sales.
  - Find total sales for each city, or for each state.
  - Find top five products ranked by total sales.
OLAP Queries

- **Roll-up:** aggregates at increasingly coarser levels of a dimension hierarchy.
  - E.g., Given total sales by ZIP, we can roll-up to get sales by city, and then by state.
OLAP Queries

- **Drill-down**: aggregates at increasingly finer levels of dimensions.
  - E.g., Given total sales by state, can drill-down to compute total sales by city.
  - E.g., Can also drill-down on a different dimension to compute total sales of each state by product.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>category</td>
</tr>
<tr>
<td></td>
<td>city</td>
</tr>
<tr>
<td></td>
<td>ZIP</td>
</tr>
</tbody>
</table>
**OLAP Queries**

- **Pivoting:** aggregates on selected dimensions
  - E.g., Pivoting on State and Year yields the **cross-tabulation** as shown below

<table>
<thead>
<tr>
<th></th>
<th>2007</th>
<th>2008</th>
<th>2009</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>63</td>
<td>38</td>
<td>75</td>
<td>176</td>
</tr>
<tr>
<td>CA</td>
<td>81</td>
<td>107</td>
<td>35</td>
<td>223</td>
</tr>
<tr>
<td>Total</td>
<td>144</td>
<td>145</td>
<td>110</td>
<td>339</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
</tbody>
</table>
**OLAP Queries**

- **Slicing and Dicing:** equality and range selections on one or more dimensions.

Slice pid=12 ➝

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
</tbody>
</table>
Tableau Demo

- https://www.youtube.com/watch?v=pXYgsd9xOZI

- Note the many measures.
- Pivot sales on category and region.
- Clear date, pivot on product and drill down on subcategory.
- Add profit as another measure
- Change bars to circles
- Pivot on dates (columns)
Comparison with SQL Queries

- The cross-tabulation obtained by pivoting:

<table>
<thead>
<tr>
<th></th>
<th>OR</th>
<th>CA</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>63</td>
<td>81</td>
<td>144</td>
</tr>
<tr>
<td>2008</td>
<td>38</td>
<td>107</td>
<td>145</td>
</tr>
<tr>
<td>2009</td>
<td>75</td>
<td>35</td>
<td>110</td>
</tr>
<tr>
<td>Total</td>
<td>176</td>
<td>223</td>
<td>339</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
</tbody>
</table>
The cross-tabulation obtained by pivoting can also be computed using a collection of SQL queries:

```
SELECT T.year, L.state, SUM(S.amt)
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.locid=L.locid
GROUP BY T.year, L.state

SELECT T.year, SUM(S.amt)
FROM Sales S, Times T
WHERE S.timeid=T.timeid
GROUP BY T.year

SELECT L.state, SUM(S.amt)
FROM Sales S, Location L
WHERE S.locid=L.locid
GROUP BY L.state

SELECT SUM(S.amt)
FROM Sales S
```
The CUBE Operator

- If there are \( k \) dimensions, we have \( 2^k \) possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.

- \textbf{GROUP BY CUBE}(\text{pid, locid, timeid})
  - Equivalent to rolling up Sales on all eight subsets of the set \{\text{pid, locid, timeid}\}

```sql
SELECT grouping-list, \text{SUM}(S.amt)
FROM   Sales S
GROUP BY CUBE(grouping-list)
ORDER BY grouping-list
```
Cube Operator (cont’d)

- GROUP BY CUBE(pid, locid, timeid) -- SUM Sales
  - Equivalent to rolling up Sales on all eight subsets of \{pid, locid, timeid\}; each roll-up amounts to a query of the form:

  ```sql
  SELECT SUM(S. amt)
  FROM Sales S
  GROUP BY grouping-list
  ```

- The *lattice* of group-by operations

[Diagram showing the lattice of group-by operations with nodes for pid, locid, timeid, pid, locid, pid, timeid, locid, timeid, pid, locid, pid, timeid, locid, timeid, pid, locid, pid, locid, locid, timeid, timeid, ALL]
Overview of Topics

1. Introduction
   - Operational vs. Warehouse

2. Multidimensional Data
   - Data model & schema

3. Queries
   - OLAP Queries
   - CUBE Operator
   - Window Operator

4. Implementation Algorithms
   - Bitmap Index
   - MOLAP vs ROLAP

5. Materialized Views
   - Query answering using views
   - View maintenance
   - View selection

6. Constructing a Data Warehouse (ETL)

7. An example of data mining
   - Association rule mining
A. Relational Techniques

- ROLAP systems:
  - We know how to implement each group by: hashing or sorting
  - How do we implement the lattice?
    - Reuse group by results from the previous layer
    - Given multiple choices from the previous layer, choose one that gives the smallest cost
B. Array-based Algorithms for CUBE

- **MOLAP systems:**
  - Implement the lattice using a multi-dimensional array
  - Carefully walk through the array to compute all projected aggregates *in one pass* (or minimum number of passes) with minimal memory requirements

---

- An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. Zhao, Deshpande, Naughton, ACM SIGMOD 1997.
1) Converting A Relation to Cube on Disk

- How do we construct a cube from a relation?
  - Data does not fit in memory
  - No bias towards a particular dimension. Sort by (locid, date)?
Converting A Relation to Array on Disk

- Chunking algorithm
  - Divide a n-dimensional array into smaller n-dimensional chunks, each of which fits in memory. Store each chunk as an object on disk.
2) A Simple Cubing Algorithm

- Compute each group by in a separate pass

- Given: cube of dimensions A,B,C. Compute: aggregates for BC.
- "Sweep a plane" through the A dimension, and bring the values into the BC plane. Do this in a chunk-wise fashion.
- This uses only 1 chunk-size piece of memory at a time.
3) A Multi-Way Array Algorithm

- Compute all sub-aggregates in one pass of the disk-based cube, using minimum memory.
  - *Dimension Order*, O = (D_{j1}, D_{j2}, ..., D_{jn}), is an ordering on the dimensions corresponding to a "row major" traversal of chunks

  - ABC in the example below.

  - |Di| = size of dimension i
  - |Ci| = size of chunk for dimension i
  - |Ci| << |Di| in general
Multi-Way Array Algorithm (cont’d)

- Dimension order determines memory requirements:
  - BC values require 1 chunk BC each to aggregate away the A's
  - AC values require 4 chunks ACs each to aggregate away the B's
  - AB values require 16 chunks ABs each to aggregate away the C's
  - If a unit cell size is $u$, chunk size of dimension $X$ is $X_c$, and dimension size of $Y$ is $Y_d$, we need
    \[ |B_c| |C_c| u \ + \ |A_d| |C_c| u \ + \ |A_d| |B_d| u \]
    memory to do this simultaneously.
Multi-Way Array Algorithm (cont’d)

- Dimension order determines memory requirements:
  - If a unit cell size is $u$, chunk size of dimension $X$ is $X_c$, and dimension size of $Y$ is $Y_d$, we need
    $$|B_c| |C_c| u + |A_d| |C_c| u + |A_d| |B_d| u$$
    memory to do this simultaneously.
  - **Memory Rule**: to compute a projection on one dimension, use product of dimension sizes in the *ordering prefix* before the projected dimension, times the chunk sizes in the *ordering suffix* after projection.
    - It’s easier to aggregate away dimensions earlier in the dimension order.
Minimum Memory Spanning Tree

Minimum Memory Spanning Tree Algorithm (MMST)

- For each node in the lattice, choose the parent that requires the least memory during traversal, according to the Memory Rule.
  - That is, try to project an earlier dimension.
  - (Note: this depends on the dimension order.)
Minimum Memory Spanning Tree

Minimum Memory Spanning Tree Algorithm (MMST)

- For each node in the lattice, choose the parent that requires the least memory during traversal, according to the Memory Rule.
  - That is, try to project an earlier dimension.
  - (Note: this depends on the dimension order.)

- To project out more, use the same logic to go from \( k \) dimensions to \( k-1 \).
  - E.g. to compute \( A \) from \( AC \), need to look at \( |A_d| \) values simultaneously.
  - There’s a simple formula \( F \) for total memory requirements for MMST.

- What is the optimal dimension order?
  - Can be computed by optimizing the simple formula \( F \).
  - Turns out to be simple: order by increasing dimension size!
  - Memory need is independent of the size of the largest dimension (which is first aggregated away or becomes the suffix after projection).