Data Warehousing

Conclusion

Toon Calders
Motivation for the Course

• Database = a piece of software to handle data:
  – Store, maintain, and query

• Most ideal system situation-dependent
  – data type: simple / semi-structured / complex / ...
  – types of queries: simple lookup / analytical / ...
  – type of usage: multi-user / single-user / distributed / ...
Example Question

• What are the ACID properties? Illustrate these properties with your own examples. Why are they important for OLTP?
  – Atomicity
  – Consistency
  – Isolation
  – Durability
What About Decision Support?

• Concurrent access
  → not really
  → read-only

• Data consistency, non-redundancy
  → data comes from consistent sources (sort of)
  → data does not change during analysis; once clean, always clean
What About Decision Support?

• Ad-hoc Querying
  → No longer true;
  → Spread-sheet like queries
  → Long-running queries, touching large parts of the database
    → In combination with transactions, kills the database

• Efficiency
  → Relational DBMS optimized for other types of queries
What About Decision Support?

• OLTP systems not very efficient for data analysis tasks
  – analysis queries might stall operational systems
  – architecture suboptimal
    • different indexing structures
    • denormalization
  – need of historical data versus only current data
Three-Tier Architecture

**Getting the data inside**
- Extract
- Transform
- Load

**Problems:**
- Noisy data
- Data not consistent

**Storing** the data; general-purpose computer
- ROLAP & MOLAP
- Indexing, view-materialization, partitioning, column-stores

**Special purpose appliances**
- IBM Pure Data for Analytics
- Teradata

**Analyzing data:**
- Conceptually a cube
- MDX as a cube-oriented query languages
- Supporting complex analytical SQL queries
- Usually not as ad-hoc as regular queries

**Parallelization to speed up querying:**
- E.g., map-reduce

**OLAP Engine**
- Analyzing data

**Data Storage**
- Storing the data; general-purpose computer
- Indexing, view-materialization, partitioning, column-stores

**Front-End Tools**
- Getting the data inside
- Storing the data
- Analyzing data

**Data Sources**
- Extract
- Transform
- Load

**Data Marts**
- Special purpose appliances
- IBM Pure Data for Analytics
- Teradata

**OLAP Server**
- Analyzing data
- Conceptually a cube
- MDX as a cube-oriented query languages
- Supporting complex analytical SQL queries
- Usually not as ad-hoc as regular queries

**Data Mining**
- Parallelization to speed up querying
- E.g., map-reduce
Three-Tier Architecture

Data Sources
- Operational DBs
- other sources

Data Storage
- Metadata
- Data Warehouse
- Extract Transform Load Refresh
- Data Marts

OLAP Engine
- Monitor & Integrator
- Analyzing data:
  - Conceptually a cube
  - MDX as a cube-oriented query languages
  - Supporting complex analytical SQL queries
  - Usually not as ad-hoc as regular queries

Front-End Tools
- Data Marts
- OLAP Server
- ROLAP Server
- Data Mining
- Query/Reporting
- Data Mining
- Analyzing data:
  - Conceptually a cube
  - MDX as a cube-oriented query languages
  - Supporting complex analytical SQL queries
  - Usually not as ad-hoc as regular queries

Data Marts
- Data Warehouse
- Extract Transform Load Refresh
- Monitor & Integrator
- ROLAP Server
- Data Marts
- OLAP Server
- Data Mining
- Analyzing data:
  - Conceptually a cube
  - MDX as a cube-oriented query languages
  - Supporting complex analytical SQL queries
  - Usually not as ad-hoc as regular queries

Data Sources
- Operational DBs
- other sources

Data Storage
- Metadata
- Data Warehouse
- Extract Transform Load Refresh
- Data Marts

OLAP Engine
- Monitor & Integrator
- Analyzing data:
  - Conceptually a cube
  - MDX as a cube-oriented query languages
  - Supporting complex analytical SQL queries
  - Usually not as ad-hoc as regular queries

Front-End Tools
- Data Marts
- OLAP Server
- ROLAP Server
- Data Mining
- Analyzing data:
  - Conceptually a cube
  - MDX as a cube-oriented query languages
  - Supporting complex analytical SQL queries
  - Usually not as ad-hoc as regular queries
Data Cubes as the Conceptual Model

Country
- Ireland
- France
- Germany

Product
- TV
- PC
- VCR

Date
- 1Qtr
- 2Qtr
- 3Qtr
- 4Qtr

sum
Aggregated over all
Querying the Cube

• Slice, Dice, Roll-up, Drill-Down
  – Cube browsing tools

• MDX as the SQL for OLAP
  – Select dimensions for display
  – Define new aggregates
  – Select slices
  – ...

Example Question

• Explain the meaning of the following MDX query:

```mdx
select
    [Customer].[Gender].members on columns,
    ( { [France], [Germany] }, education.members ) on rows
from [Adventure Works]
where ( [Customer Count],
    { [Commute Distance].[0-1 Miles],
      [Commute Distance].[1-2 Miles] } )
```
<table>
<thead>
<tr>
<th>Country</th>
<th>Type</th>
<th>All Customers</th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>All Customers</td>
<td>1,161</td>
<td>573</td>
<td>588</td>
</tr>
<tr>
<td>France</td>
<td>Bachelors</td>
<td>282</td>
<td>132</td>
<td>150</td>
</tr>
<tr>
<td>France</td>
<td>Graduate Degree</td>
<td>160</td>
<td>83</td>
<td>77</td>
</tr>
<tr>
<td>France</td>
<td>High School</td>
<td>199</td>
<td>108</td>
<td>91</td>
</tr>
<tr>
<td>France</td>
<td>Partial College</td>
<td>329</td>
<td>158</td>
<td>171</td>
</tr>
<tr>
<td>France</td>
<td>Partial High School</td>
<td>191</td>
<td>92</td>
<td>99</td>
</tr>
<tr>
<td>Germany</td>
<td>All Customers</td>
<td>1,225</td>
<td>592</td>
<td>633</td>
</tr>
<tr>
<td>Germany</td>
<td>Bachelors</td>
<td>343</td>
<td>175</td>
<td>168</td>
</tr>
<tr>
<td>Germany</td>
<td>Graduate Degree</td>
<td>168</td>
<td>82</td>
<td>86</td>
</tr>
<tr>
<td>Germany</td>
<td>High School</td>
<td>153</td>
<td>64</td>
<td>89</td>
</tr>
<tr>
<td>Germany</td>
<td>Partial College</td>
<td>383</td>
<td>185</td>
<td>198</td>
</tr>
<tr>
<td>Germany</td>
<td>Partial High School</td>
<td>178</td>
<td>86</td>
<td>92</td>
</tr>
</tbody>
</table>
Three-Tier Architecture

**Data Sources**
- Other sources
- Operational DBs

**Data Storage**
- Extract
- Transform
- Load
- Refresh

- Storing the data; general-purpose computer
  - ROLAP & MOLAP
  - Indexing, view-materialization, partitioning, column-stores

- Special purpose appliances
  - IBM Netezza
  - Teradata

- Data Marts

**OLAP Engine**
- OLAP Server
  - Serve
  - ROLAP Server

**Analysis**
- Query/Reporting
  - Data Mining

**Front-End Tools**
- Serve

**Data Marts**
- OLAP Server
- ROLAP Server
- OLAP Engine
Logical Model - ROLAP

• Modeling your data
  – Dimensional modeling
    • Fact, dimension, measure

• Cubes and pre-materialized views need to be stored in a convenient format
  – ROLAP
    • Star schema / snowflake schema
    • Dimensional modeling
  – MOLAP
Dimensional Fact Model

- Sale
  - quantity
  - unitPrice
  - VAT-rate
- customer
  - country
  - city
  - age-category
- date
  - year
  - month
- day-of-week
- product
  - brand
  - type
Corresponding Star-Schema

Customer
- CustKey
- OLTP_Custkey
- City
- Country
- Age-Cat

Sale
- CustKey
- DateKey
- ProdKey
- Quantity
- UnitPrice
- VATrate

Date
- DateKey
- Date
- DayOfWeek
- Month
- Year

Product
- ProdKey
- OLTP_Prodkey
- Brand
- Type
Dimensional Modeling

• Different ways to deal with
  – Slowly changing dimensions
  – Unbalanced hierarchies; non-covering hierarchies
  – Junk dimensions

• These are best-practices for storing dimensional data in a relational database
  – New technologies may change the rules of the game
Slowly Changing Dimension – Type 2

• Whenever there is a change, create a new version of the affected row
  – Need for surrogate key!

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
<td>John</td>
<td>Dallas</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>Mary</td>
<td>Dallas</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>Pete</td>
<td>New York</td>
</tr>
</tbody>
</table>

John and Mary move to New York
Mark is a new client
Exam Question

• Explain the concept of a mini-dimension and illustrate this concept with an original example.


<table>
<thead>
<tr>
<th>Customer dimension (original)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustID</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>PostalAddress</td>
</tr>
<tr>
<td>Gender</td>
</tr>
<tr>
<td>DateofBirth</td>
</tr>
<tr>
<td>Customerside</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>NoKids</td>
</tr>
<tr>
<td>MaritalStatus</td>
</tr>
<tr>
<td>CreditScore</td>
</tr>
<tr>
<td>BuyingStatus</td>
</tr>
<tr>
<td>Income</td>
</tr>
<tr>
<td>Education</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

| CustID                        |
| Name                          |
| PostalAddress                |
| Gender                        |
| DateofBirth                   |
| Customerside                  |
| ...                           |

| DemographyID                  |
| NoKids                        |
| MaritalStatus                 |
| CreditScoreGroup              |
| BuyingStatusGroup             |
| IncomeGroup                   |
| EducationGroup                |
| ...                           |

**Customer dimension (new):**
- relatively static attributes

**Demographics dimension:**
- often-changing attributes
## Mini-dimension

### Facts

<table>
<thead>
<tr>
<th>Skey</th>
<th>Date</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>D2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>D3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>D4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>D5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>D6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D7</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>D8</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>D9</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>D10</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>D11</td>
<td>3</td>
</tr>
</tbody>
</table>

### Customer

<table>
<thead>
<tr>
<th>SKey</th>
<th>CID</th>
<th>status</th>
<th>child</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>single</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>married</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>single</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>single</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>married</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>married</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>married</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>married</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>married</td>
<td>2</td>
<td>yes</td>
</tr>
</tbody>
</table>
# Mini-dimension

## Customer

<table>
<thead>
<tr>
<th>SKey</th>
<th>CID</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>single</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>married</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>single</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>married</td>
</tr>
</tbody>
</table>

## Facts

<table>
<thead>
<tr>
<th>Skey</th>
<th>Dkey</th>
<th>Date</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>D1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>D3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>D4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>D5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>D6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>D7</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>D8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>D9</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>D10</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>D11</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

## Demography

<table>
<thead>
<tr>
<th>DKey</th>
<th>child</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>yes</td>
</tr>
</tbody>
</table>
4. (2 points)
Consider a data cube with dimensions Student, Course, Semester, and Lecturer, and measure attributes Grade and Number_of_attempts. The following hierarchies are present:

- Student → Home_town → Country,
- Semester → Year, and
- Lecturer → Degree (e.g. full/associate/assistant professor, postdoc, ...)

(a) Give a snowflake scheme for this data cube.
(b) Use the schema of (a) to give an example of a join index and explain which queries would benefit from this example join index.
Three-Tier Architecture

Data Sources

- Operational DBs
- Other sources

Data Storage

- Extract
- Transform
- Load
- Refresh
- Storing the data; general-purpose computer
- ROLAP & MOLAP
- Indexing, view-materialization, partitioning, column-stores

Special purpose appliances
- IBM Netezza
- Teradata

OLAP Engine

- OLAP Server
- ROLAP Server

Analysis

- Query/Reporting
- Data Mining

Front-End Tools

- Data Marts
- Serve
Physical Level

• Speeding up typical data warehousing queries
  – Data Explosion Problem
  – Materialization
  – Indices
    • bitmap index, Join index, bitmap-join index
  – Partitioning tables
Example Question

• Explain why in general it is not possible to store fully materialized data cubes.
  – High dimensionality, sparse data
    → Cube exponentially larger than original data
  – No problem if cube is dense
  – Less of a problem if dimensionality is low

Inherent problem; impossible to pre-compute all possible ways to aggregate the data
Database Explosion Problem

2D: adding 1 tuple $\rightarrow$ affecting 4 cells of the cube
Database Explosion Problem

2D: adding 1 tuple $\rightarrow$ affecting 4 cells of the cube
3D: adding 1 tuple $\rightarrow$ affecting 8 cells of the cube
...
kD: adding 1 tuple $\rightarrow$ affecting $2^k$ cells of the cube
Data Explosion Problem

Size of cube w.r.t. number of dimensions (500 data points)
Storing the Data

• Want quick answers → pre-computation

• Straightforward solution, however, does not work → Data explosion problem

• Therefore, partially materialize the cube + smart indexing and storage structures

• ROLAP and MOLAP
  – Often hybrid form
Materialization

Example:

\[( \text{part, customer} )\]
SELECT customer, part, sum(sales)
FROM Sales
GROUP BY customer, part

\[( \text{part} )\]
SELECT part, sum(sales)
FROM Sales
GROUP BY part

<table>
<thead>
<tr>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
</tr>
</tbody>
</table>
Materialization

Example:

\[
\begin{align*}
&\text{(part, customer)} \\
&\text{SELECT customer, part, sum(sales)} \\
&\text{FROM Sales} \\
&\text{GROUP BY customer, part} \\
&\text{materialized as PC}
\end{align*}
\]

\[
\begin{align*}
&\text{(part)} \\
&\text{SELECT part, sum(sales)} \\
&\text{FROM Sales} \\
&\text{GROUP BY part} \\
&\text{|PC|}
\end{align*}
\]
Example: some materialized

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(part, supplier, customer)</code></td>
<td>6M</td>
<td>6M</td>
</tr>
<tr>
<td><code>(part, customer)</code></td>
<td>6M</td>
<td>6M</td>
</tr>
<tr>
<td><code>(part, supplier)</code></td>
<td>0.8M</td>
<td>0.8M</td>
</tr>
<tr>
<td><code>(supplier, customer)</code></td>
<td>6M</td>
<td>6M</td>
</tr>
<tr>
<td><code>(part)</code></td>
<td>0.2M</td>
<td>0.8M</td>
</tr>
<tr>
<td><code>(supplier)</code></td>
<td>0.01M</td>
<td>0.8M</td>
</tr>
<tr>
<td><code>(customer)</code></td>
<td>0.1M</td>
<td>0.1M</td>
</tr>
<tr>
<td>()</td>
<td>1</td>
<td>0.1M</td>
</tr>
</tbody>
</table>

Total cost: 20.6M
Example

• Base table a, table b, and f are materialized
  - Total: $2 \times 100 + 4 \times 50 + 2 \times 40 = 480$

- Additional benefit of materializing f
  = $70 = 1 \times (100-40) + 1 \times (50-40)$
Example

• Benefit for materializing the other tables:

<table>
<thead>
<tr>
<th>query</th>
<th>Materialized view</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>25</td>
</tr>
<tr>
<td>d</td>
<td>-</td>
</tr>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>f</td>
<td>25</td>
</tr>
<tr>
<td>g</td>
<td>-</td>
</tr>
<tr>
<td>h</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
</tr>
</tbody>
</table>

| a   | 100 |
| b   | 50  |
| c   | 75  |
| d   | 20  |
| e   | 30  |
| f   | 40  |
| g   | 1   |
| h   | 10  |
3. (3p) Consider the following lattice of views along with a representation of the number of rows in each view where A is the base cuboid:

```
A : 219 000
  /  \
B : 2 600 C : 17 000 D : 1 100
   |    |    |
  E : 300 F : 49 G : 165
   |    |    |
H : 105 I : 7
```

(a) Apply the greedy method described by Harinarayan, Rajaraman, and Ullman in their seminal paper “Implementing Data Cubes Efficiently” (SIGMOD 1996) to select 2 views from the views B-J to materialize.

(b) What is the total benefit of materializing these two views? (You can give a formula involving parentheses, multiplications, divisions, additions, subtractions, powers)
Bitmap-Join Index: Example

<table>
<thead>
<tr>
<th>Date</th>
<th>pID</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/5/12</td>
<td>1</td>
<td>Jack</td>
</tr>
<tr>
<td>10/5/12</td>
<td>1</td>
<td>Pete</td>
</tr>
<tr>
<td>13/5/12</td>
<td>3</td>
<td>John</td>
</tr>
<tr>
<td>14/5/12</td>
<td>2</td>
<td>Mary</td>
</tr>
</tbody>
</table>

```
SELECT date
FROM Sales S
join Product P
join Customer C
ON ...
WHERE
P.Category = "Food" and
C.City = "Brussels";
```
SELECT date
FROM Sales S join Product P
join Customer C on ...
WHERE
P.Category = "Food" and
C.City = "Brussels";

0011 & 1001 → 0001
Partitioning

• Separate database/tables/indices over different partitions
  – Horizontal partitioning: every partition holds a subset of the *tuples*
    E.g., partition fact table by *month*
  – Vertical partitioning: every partition holds a subset of the *attributes*
Three-Tier Architecture

Getting the data inside:
- Extract
- Transform
- Load

Problems:
- Noisy data
- Data not consistent

Data Sources

Data Storage

OLAP Engine
Front-End Tools

OLAP Engine

Front-End Tools

Analysis

Query/Reporting

Data Mining
ETL

• Extract – Transform – Load
• Many existing tools
  – Data Stage
  – Informatica
• Importance of metadata
  – Which reports cannot be trusted?
  – Impact analysis
  – Data lineage
ETL

• Important step in transformation: linking different tables

• Often difficult
  – Different keys
  – Small variations/errors
Load

• Bulk-loading data
• Typically rebuild (hard-to-update) indices

• Computing pre-aggregations
  – Sort-based
  – Hash-based
Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

```sql
SELECT A, B, C, sum(count) FROM R GROUP BY A, B, C;
```
Sort-Based Aggregation

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

`SELECT A, B, C, sum(count) FROM R GROUP BY A, B, C;`

SORT
Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

```sql
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```
Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

```
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```

SCAN

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Sort-Based Aggregation

### Table 1: Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

### SQL Query

```
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```

### Scan Result

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>
Sort-Based Aggregation

\[
\begin{array}{cccc}
\text{A} & \text{B} & \text{C} & \text{count} \\
1 & 5 & 6 & 8 \\
1 & 5 & 6 & 6 \\
1 & 8 & 6 & 10 \\
2 & 1 & 4 & 8 \\
2 & 1 & 4 & 9 \\
3 & 3 & 3 & 5 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{A} & \text{B} & \text{C} & \text{SUM} \\
1 & 5 & 6 & 14 \\
1 & 8 & 6 & 10 \\
\end{array}
\]

\[
\text{SELECT A, B, C, sum(count)}
\text{FROM R}
\text{GROUP BY A, B, C;}
\]

SCAN
Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

```sql
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```

SCAN
### Sort-Based Aggregation

```sql
SELECT A, B, C, sum(count) 
FROM R 
GROUP BY A, B, C;
```

**Table:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>17</td>
</tr>
</tbody>
</table>
Sort-Based Aggregation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Scan

```
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```
Sort-Based Aggregation

\[
\text{SELECT A, B, C, sum(count)} \\
\text{FROM R} \\
\text{GROUP BY A, B, C;}
\]

SCAN
Pipe-Sort

• Key problem: divide materialized views lattice into “pipes”, minimizing sorts
Hash-Based Aggregation

• If aggregated table fits into memory
  → Hash on grouping attributes, update measure
• Multiple hash tables fit together into the memory
  → Compute in one run
• Hash-based algorithm: selects optimal sets to be processed at the same time
Example Question

Suppose that we need to compute the aggregations Average and Min of attribute cost for the following groups of attributes:

   AB, C, BC, ABC

Give an efficient way to do this, assuming none of the aggregated tables fits into memory.
Solution

• Average and Min:
  – Average is not distributive:
    \[ \text{AVG}(A \cup B) \neq \text{AVG}([\text{AVG}(A), \text{AVG}(B)]) \]
    - AVG can be computed from SUM and COUNT
    - SUM and COUNT are distributive
  – Min is distributive:
    \[ \text{min}(A \cup B) = \text{min}([\text{min}(A), \text{min}(B)]) \]

(Why is it important that measures are distributive?)
Solution

```
\[\text{sort CBA
    \[\text{sort CB}
        \[\text{C}
    \]\[\text{AB}
\]
\]
```
### Three-Tier Architecture

**Storing** the data; general-purpose computer
- ROLAP & MOLAP
- Indexing, view-materialization, partitioning, column-stores

**Special purpose appliances**
- IBM Netezza
- Teradata

Parallelization to speed up querying:
- E.g., map-reduce

- **Data Sources**
  - other sources
  - Operational DBs

- **Data Storage**
  - Extract
  - Transform
  - Load
  - Refresh
  - Data Marts

- **OLAP Engine & Front-End Tools**
  - OLAP
  - Server
  - Analysis
  - Reporting
  - Data Mining

**Different architectures**
**Rules of the game change**
Different Architectures

• Problems:
  – Disk access is slow
  – Full table scan is faster than random read, but is slow if only part of the table is needed

Move processor close to the data;
Compress data on disk = trade in slow I/O for fast processing
Multiple processors responsible for smaller part of the data

Implement select-project into the hardware
Zone-maps could be considered as a form of indexing
Vertical partitioning avoids access to attributes that are not needed

Obviously, query optimizer needs to be able to deal with new reality!
Different Architectures

• Challenges
  – Distribute data in an intelligent way
    • Hash-based; preferably on join-keys
  – In this way: truly distribute work

• What kills performance?
  – Excessive communication between nodes
    • E.g., poor distribution; non-selective self-joins
Three-Tier Architecture

Data Sources

- other sources
- Operational DBs

Data Storage

- Metadata
- Extract Transform Load Refresh
- Data Warehouse
- OLAP Engine

OLAP Engine

- OLAP Server
- ROLAP Server
- Data Marts

Front-End Tools

- Analysis
- Query/Reporting

Data Mining

- Monitor & Integrator
- Serve
- Data mining
Conclusion

• Different ways to support data analysis
  – Traditional view
    • ETL;
    • ROLAP/MOLAP storage;
    • Logical optimizations:
      – materialized views
    • Physical optimizations:
      – indices; partitioning
    • OLAP/Data mining to do the analysis
Conclusion

– New hardware/appliances
  • Restrictions change
  • Multi processor
  • New game; different optimization strategies

Remember: 100 processors make a task at most 100 times faster; getting to this factor 100, however, is non-trivial!