Data Warehousing
ETL

Esteban Zimányi
ezimanyi@ulb.ac.be
Slides by Toon Calders
Until now

Data Warehouse

Extract
Transform
Load
Refresh

Metadata

Monitor & Integrator

OLAP Server

Serve

Data Sources

Operational DBs

other sources

Conceptual modeling (DFM)
Logical model (star, snowflake)
Indices; view materialization

Data cube; slice & dice; cube browsing

Data Marts

Operational DBs

other sources

Metadata

Monitor & Integrator

OLAP Server

Serve

Data Sources

Conceptual modeling (DFM)
Logical model (star, snowflake)
Indices; view materialization

Data cube; slice & dice; cube browsing

Analysis

Query/Reporting

Data Mining

Front-End Tools

Operational DBs

other sources

Metadata

Monitor & Integrator

OLAP Server

Serve

Data Sources

Conceptual modeling (DFM)
Logical model (star, snowflake)
Indices; view materialization

Data cube; slice & dice; cube browsing

Analysis

Query/Reporting

Data Mining

Front-End Tools
This lecture

Data
Warehouse

Extract
Transform
Load
Refresh

Monitor
&
Integrator

OLAP
Engine

Metadata

OLAP
Server

Serve

Conceptual
modeling (DFM)

Logical model (star,
snowflake)

Indices; view
materialization

How to fill
the DW?

How keep it
up to date?

Data Sources

other
sources

Operational
DBs

Data Marts

Operational
DBs

other
sources

Data Sources

Analysis

Query/Reporting

Data Mining

Front-End Tools
Summary

• Data Warehouse architectures
  – Single-, two-, and three-layer architectures

• ETL process
  – Extract
  – Transform/Cleanse
  – Load

Sec. 1.3, 1.4; Ch. 10 of Golfarelli and Rizzi book
Single Layer Architecture

- Source layer
- Data warehouse layer
- Middleware
- Operational data
- Analysis
- Reporting tool
- OLAP tool
Single Layer Architecture

• Not frequently used
• Reconciled data is not materialized
  – Middleware makes heterogeneity transparent to the user

+ No data duplication
- No separation of analytical and transactional processing
Two-Layer Architecture

Source layer
- Data staging
- Data warehouse layer
- Analysis

Operational data
- ETL tools
- Data Warehouse
- Meta-data
- Data marts
- Reporting tool
- OLAP tool
Two-Layered Architecture

- Introduction of data warehouse layer
  - Data is materialized
- Historical data management
- ETL runs regularly to keep data warehouse up-to-date

- Data duplication
+ Separation of analytics and operations
Three-Layer Architecture

Source layer
- Data staging
- Reconciled layer
- Loading
- Data warehouse layer
- Analysis

Data staging

Reconciled layer
- ETL tools
- Reconciled data
- ETL tools
- Data Warehouse
- Loading

Data warehouse layer

Analysis

Report tool

OLAP tool

Data marts

Meta-data

Operational data

ETL tools
Three-Layer Architecture

• Data from different sources needs to be reconciled
  – Different schemas need to be integrated
  – Source data needs to be cleansed

• In 3-tier architecture the reconciled data is materialized as well
  – Useful in its own respect
  – Makes ETL more transparent
  – Neither historical, nor dimensional
Data Staging

• Often the data staging area is materialized as well
  – Store helper tables
  – Take burden away from source systems
Summary

• Data Warehouse architectures
  – Single-, two-, and three-layer architectures

• ETL process
  – Extract
  – Transform/Cleanse
  – Load
Extract

• Relevant data obtained from data sources
  – First load: extract all data
  – Subsequent loads: extract changes

• In 3-Tier infrastructure:
  – From source data to reconciled database
    • Schema integration, transformation, cleansing
  – From reconciled database to data warehouse
    • De-normalization, introduction of surrogate keys
    • Calculation of derived data
Extract

• Static vs incremental extraction
• Incremental & immediate
  – Application-assisted
  – Trigger-based extraction
  – Log-based
• Incremental & delayed
  – Timestamp-based
  – File Comparison
Application-Assisted

- Update of data warehouse deeply integrated into the application software
  - Immediate
  - Requires adaptation of existing applications
  - Hard to maintain
Trigger-Based

- Closely related to application-based
  - Triggers to store updates
  - Huge performance hit for database
  - More transparent for application layer
  - Only possible for data maintained in database
Log-Based

• Many database systems keep change logs
  – To ensure durability; in-between checkpoints all transactions to the database are logged
  – Use these change logs for updating data warehouse
  – Transparent to the user
Timestamp-Based

- Operational database can be changed to keep whole history
  - Different levels of timestamps
    - Timestamp per tuple ↔ Temporal database
Timestamp-Based

- Only last update date may lose information

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Karl</td>
<td>M</td>
<td>52</td>
<td>2</td>
<td>D1</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>
## Timestamp-Based

- Only last update date may lose information

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Karl</td>
<td>M</td>
<td>52</td>
<td>2</td>
<td>D1</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>2</td>
<td>D2</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>
### Timestamp-Based

- Only last update date may lose information

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Karl</td>
<td>M</td>
<td>52</td>
<td>2</td>
<td>D1</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>2</td>
<td>D2</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>3</td>
<td>D3</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Martin</td>
<td>M</td>
<td>64</td>
<td>2</td>
<td>D3</td>
</tr>
</tbody>
</table>
Timestamp-Based

- Only last update date may lose information

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Karl</td>
<td>M</td>
<td>52</td>
<td>2</td>
<td>D1</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>2</td>
<td>D2</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Gender</th>
<th>Age</th>
<th>Segment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>M</td>
<td>36</td>
<td>3</td>
<td>D3</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>18</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>Martin</td>
<td>M</td>
<td>64</td>
<td>2</td>
<td>D3</td>
</tr>
</tbody>
</table>
Timestamp-Based

• Based on the application field and update frequency, one timestamp per tuple may be acceptable
  – Type-2 change for attributes that change several times between updates?!
• If more fine-grained behavior is required we can go to full temporal database
  – Register valid time (start-end)
File Comparison

- Easy; works even for flat files
- Does not capture intermediate changes

Diagram:

1. Application
   - DBMS
   - Database
   - Previous Copy
   - Compare
   - Changes
   - ETL tools
   - Data warehouse

2. Database
   - Previous Copy
   - Compare
   - Changes
   - ETL tools
   - Data warehouse
# Extraction Methods - Overview

<table>
<thead>
<tr>
<th>Method</th>
<th>Maintain full history?</th>
<th>Impact on performance operational level</th>
<th>Complexity extraction procedures</th>
<th>DBMS dependent</th>
<th>Requires changes to application layer</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>No</td>
<td>No</td>
<td>Low</td>
<td>No</td>
<td>None</td>
<td>Easy</td>
</tr>
<tr>
<td>Application assisted</td>
<td>Yes</td>
<td>Medium</td>
<td>High</td>
<td>No</td>
<td>Many</td>
<td>Difficult</td>
</tr>
<tr>
<td>Trigger based</td>
<td>Yes</td>
<td>Medium</td>
<td>Medium</td>
<td>Yes</td>
<td>None</td>
<td>Medium</td>
</tr>
<tr>
<td>Log based</td>
<td>Yes</td>
<td>No</td>
<td>Low</td>
<td>Limited</td>
<td>None</td>
<td>Easy</td>
</tr>
<tr>
<td>Timestamp based</td>
<td>(yes)</td>
<td>Some</td>
<td>Low</td>
<td>Limited</td>
<td>Some</td>
<td>Medium</td>
</tr>
<tr>
<td>File comparison</td>
<td>Not all</td>
<td>No</td>
<td>Medium</td>
<td>No</td>
<td>None</td>
<td>Easy</td>
</tr>
</tbody>
</table>
• Data Warehouse architectures
  – Single, two-, and three-layer architectures

• ETL process
  – Extract
  – Transform/Cleanse
  – Load
Transform/Cleanse

• Conversion
  – Different date types

• Enrichment
  – Combine information of different attributes to create a new one

• Joining data without key;
  – Entity resolution

• Correcting errors
  – De-duplication
De-duplication & Entity Resolution

• De-duplication
  – Recognize duplicate entries

• Entity resolution
  – Recognize if two entities from different sources are actually referring to the same object
    • Not always obvious if there is no key shared between different data sources
  – In most extreme case need for methods to directly compare strings
Example: Cleansing customer data

John White
Downing St. 10
TW1A 2AA London (UK)

Fname: John
Sname: White
Adress: Downing St. 10
ZIP: TW1A 2AA
City: London
Country: UK

normalizing

Example from Golfarelli and Rizzi book
Comparing Strings

• How to compare strings?
  Patrick Smith vs. Patrik Smyth

• Popular distance measure: edit distance
  – Insert, Delete, Overwrite
  – Shortest sequence of operations to turn one string into another

Patrick Smith $\rightarrow_D$ Patrik Smith $\rightarrow_R$ Patrik Smyth
Edit Distance

- Relatively easy to compute via dynamic programming

Smyth vs Simte

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>m</th>
<th>y</th>
<th>t</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>t</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

$d(S, S)$

$d(S, Si)$

$d(Smyth, Simte)$
Edit Distance

• Compute content of a cell as follows:

Example:

\[ d(\text{Smyth}h, \text{Simte}) : \]
\[ d(\text{Smyth}, \text{Simt}) + \text{insert } e \text{ at the end} \]
\[ \text{or delete } h + d(\text{Smyt}, \text{Simte}) \]
\[ \text{or } d(\text{Smyt}, \text{Simt}) + \text{overwrite } h \text{ with } e \]
Edit Distance

• Compute content of a cell as follows:

Example:
\[ d(\text{Smyth}, \text{Simte}) = \min( \]
\[ d(\text{Smyth}, \text{Simt}) + 1, \]
\[ d(\text{Smyt}, \text{Simte}) + 1, \]
\[ d(\text{Smyt}, \text{Simt}) + 1 \]
\[ ) \]
Edit Distance

• Compute content of a cell as follows:

Example:

\[ d(Symt, Simt) : \]

\[ d(Symt, Sim) + \text{insert } t \text{ at the end} \]

\[ \text{or delete } t + d(Sym, Simt) \]

\[ \text{or } d(Sym, Sim) \]
Edit Distance

• Compute content of a cell as follows:

Example:
\[
\text{d(Smyt,Simt)} = \min( \\
\text{d(Smyt,Sim) + 1} \\
\text{d(Smy,Simt) + 1,} \\
\text{d(Smy,Sim) + 0} \\
) \\
\]
Edit Distance

• Compute content of a cell as follows:

\[
\text{if } \text{string}_1[k] = \text{string}_2[l] \\
M[k,l] = \min(M[k-1,l-1], M[k-1,l]+1, M[k,l-1]+1)
\]

\text{else:} \\
M[k,l] = \min(M[k-1,l-1]+1, M[k-1,l]+1, M[k,l-1]+1)
Summary

• Data Warehousing architectures
  – Single, two-, and three-layer architectures

• ETL process
  – Extract
  – Transform/Cleanse
  – Load
Populating Dimension Tables

• Type 1 and type 2 changes
  – May require dimension table to be loaded in the data staging area

• Introduction of surrogate keys
  – Permanently maintain mapping tables in the staging area
    • In case of a type-1 change: change the value in the dimension table
    • In case of a type-2 change: introduce a new tuple with a new surrogate key
Populating Fact Tables

• Always update dimension tables before fact table (if possible)
  – Referential integrity
  – Not always possible: introduce “dummy” value in dimension tables if dimension key lookup fails

• Update materialized views
  – Can be optimized if measures are distributive
Computation of Aggregations

- Part of the data warehouse loading involves computation of aggregations → materialized views
  - Order in which they are computed can be important
  - Sort-based or Hash-based

See, e.g.: Agarwal et al. On the computation of multidimensional aggregates. VLDB 1996
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

### SQL Query

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

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

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

<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

<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>
<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>

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

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>14</td>
</tr>
</tbody>
</table>
Sort-Based Aggregation

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

SCAN
Sort-Based Aggregation

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

```

SCAN
Sort-Based Aggregation

### Table 1: Raw Data

<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 Query

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

### Result 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

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

SCAN
Sort-Based Aggregation

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

\[
\begin{array}{cccc}
A & B & C & \text{SUM} \\
1 & 5 & 6 & 14 \\
1 & 8 & 6 & 10 \\
2 & 1 & 4 & 17 \\
3 & 3 & 3 & 5 \\
\end{array}
\]

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

SCAN
Sort-Based Aggregation

• Observation:
  – Table sorted on ABC
    = sorted on AB
    = sorted on A
  – One sort supports 3 aggregations

<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>
Pipe-Sort

• Sort on ABC
• Scan sorted relation
  – As long as next tuple is the same as previous
    • Update aggregated tuple; add count of current
  – Else
    • Ship aggregated tuple to disk
    • Pipe tuple to procedure computing aggregation on AB
Pipe-Sort

- Key problem: divide materialized views lattice into “pipes”, minimizing sorts
Pipe-Sort

• Key problem: divide materialized views lattice into “pipes”, minimizing sorts

```
   BCDA
     ▼
    /   \
   BCD
     ▼
  AB   BC
    ▼   ▼
   A   B   D
    ▼   ▼   ▼
   ()   ()   ()
```
Hash-Based Optimization

• If aggregate tables fit into memory

<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>
Hash-Based Optimization

- If aggregate tables fit into memory

<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>

<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>
Hash-Based Optimization

- If aggregate tables fit into memory

<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>

<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>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
Hash-Based Optimization

- If aggregate tables fit into memory

<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>

<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>
<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>
</tbody>
</table>
Hash-Based Optimization

• If aggregate tables fit into memory

<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>

<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><strong>14</strong></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>
</tbody>
</table>

SCAN
Hash-Based Optimization

- If aggregate tables fit into memory

<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>

<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>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>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
### Hash-Based Optimization

- If aggregate tables fit into memory

#### 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>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>

#### SUM

<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>2</td>
<td>1</td>
<td>4</td>
<td><strong>17</strong></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

**SCAN**
Hash-Based Optimization

• Multiple hash tables may fit at the same time

<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>

<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>2</td>
<td>1</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

63
Conclusion

• ETL is often the most time consuming parts of a datawarehousing project
  – Reported to take up to 80% of time

• Different ways to capture changes
  – Application-assisted
  – Trigger-based extraction
  – Log-based
  – Timestamp-based
  – File Comparison
Conclusion

• Transform/Cleanse involves
  – Normalization
  – Standardization
  – De-duplication
  – Entity resolution

• Load
  – Surrogate keys (lookup tables in staging area)
  – First update dimension tables, only then fact table