Data Warehousing & OLAP

What is Data Warehouse?

- “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
- A Data Warehouse is used for On-Line-Analytical-Processing:
  “Class of tools that enables the user to gain insight into data through interactive access to a wide variety of possible views of the information”
- 3 Billion market worldwide [1999 figure, olapreport.com]
  - Retail industries: user profiling, inventory management
  - Financial services: credit card analysis, fraud detection
  - Telecommunications: call analysis, fraud detection
Data Warehouse Initiatives

• Organized around major subjects, such as *customer, product, sales*
  – integrate multiple, heterogeneous data sources
  – exclude data that are not useful in the decision support process
• Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing
  – emphasis is on complex, exploratory analysis not day-to-day operations
• Large time horizon for trend analysis (current and past data)
• Non-Volatile store
  – physically separate store from the operational environment

Data Warehouse Architecture

• Extract data from operational data sources
  – clean, transform
• Bulk load/refresh
  – warehouse is offline
• OLAP-server provides multidimensional view
• Multidimensional-olap
  (Essbase, oracle express)
• Relational-olap
  (Redbrick, Informix, Sybase, SQL server)
Why do we need all that?

• Operational databases are for On Line Transaction Processing
  – automate day-to-day operations (purchasing, banking etc)
  – transactions access (and modify!) a few records at a time
  – database design is application oriented
  – metric: transactions/sec

• Data Warehouse is for On Line Analytical Processing (OLAP)
  – complex queries that access millions of records
  – need historical data for trend analysis
  – long scans would interfere with normal operations
  – synchronizing data-intensive queries among physically separated databases would be a nightmare!
  – metric: query response time

Examples of OLAP

• Comparisons (this period v.s. last period)
  – Show me the sales per region for this year and compare it to that of the previous year to identify discrepancies

• Multidimensional ratios (percent to total)
  – Show me the contribution to weekly profit made by all items sold in the northeast stores between may 1 and may 7

• Ranking and statistical profiles (top N/bottom N)
  – Show me sales, profit and average call volume per day for my 10 most profitable salespeople

• Custom consolidation (market segments, ad hoc groups)
  – Show me an abbreviated income statement by quarter for the last four quarters for my northeast region operations
Multidimensional Modeling

- Example: compute total sales volume per product and store

<table>
<thead>
<tr>
<th>Total Sales</th>
<th>Product 1</th>
<th>Product 2</th>
<th>Product 3</th>
<th>Product 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$454</td>
<td>-</td>
<td>-</td>
<td>$925</td>
</tr>
<tr>
<td>2</td>
<td>$468</td>
<td>$800</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>$296</td>
<td>-</td>
<td>$240</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>$652</td>
<td>-</td>
<td>$540</td>
<td>$745</td>
</tr>
</tbody>
</table>

Dimensions and Hierarchies

- A cell in the cube may store values (measurements) relative to the combination of the labeled dimensions
Common OLAP Operations

- **Roll-up**: move up the hierarchy
  - e.g. given total sales per city, we can roll-up to get sales per state

- **Drill-down**: move down the hierarchy
  - more fine-grained aggregation
  - lowest level can be the detail records (drill-through)

```
<table>
<thead>
<tr>
<th>category</th>
<th>region</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>country</td>
<td>quarter</td>
</tr>
<tr>
<td>state</td>
<td>month</td>
<td>week</td>
</tr>
<tr>
<td>city</td>
<td>store</td>
<td></td>
</tr>
</tbody>
</table>
```

Pivoting

- **Pivoting**: aggregate on selected dimensions
  - usually 2 dims (cross-tabulation)

```
<table>
<thead>
<tr>
<th>Store</th>
<th>Product</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>454</td>
<td>-</td>
<td>-</td>
<td>925</td>
<td>1379</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>468</td>
<td>800</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1268</td>
</tr>
<tr>
<td>3</td>
<td>296</td>
<td>-</td>
<td>240</td>
<td>-</td>
<td>536</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>652</td>
<td>-</td>
<td>540</td>
<td>745</td>
<td>1937</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>1870</td>
<td>800</td>
<td>780</td>
<td>1670</td>
<td>5120</td>
<td></td>
</tr>
</tbody>
</table>
```
Slice and Dice Queries

- **Slice and Dice**: select and project on one or more dimensions

Roadmap

- What is a data warehouse and what it is for
- What are the differences between OLTP and OLAP
- Multi-dimensional data modeling
- Data warehouse design
  - the star schema, bitmap indexes
- The Data Cube operator
  - semantics and computation
- Aggregate View Selection
- Other Issues
Data Warehouse Design

- Most data warehouses adopt a star schema to represent the multidimensional model
- Each dimension is represented by a dimension-table
  - LOCATION(location_key, store, street_address, city, state, country, region)
  - dimension tables are not normalized
- Transactions are described through a fact-table
  - each tuple consists of a pointer to each of the dimension-tables (foreign-key) and a list of measures (e.g. sales $$)
Advantages of Star Schema

• Facts and dimensions are clearly depicted
  – dimension tables are relatively static, data is loaded (append mostly) into fact table(s)
  – easy to comprehend (and write queries)

"Find total sales per product-category in our stores in Europe"

SELECT PRODUCT.category, SUM(SALES.amount)
FROM SALES, PRODUCT, LOCATION
WHERE SALES.product_key = PRODUCT.product_key
AND SALES.location_key = LOCATION.location_key
AND LOCATION.region="Europe"
GROUP BY PRODUCT.category
Indexing OLAP Data: Bitmap Index

- Each value in the column has a bit vector:
  - The $i$-th bit is set if the $i$-th row of the base table has the value for the indexed column
  - The length of the bit vector: # of records in the base table
- Mainly intended for small cardinality domains

<table>
<thead>
<tr>
<th>location_key</th>
<th>Region</th>
<th>Index on Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>Asia</td>
<td>1 0 0</td>
</tr>
<tr>
<td>L2</td>
<td>Europe</td>
<td>0 1 0</td>
</tr>
<tr>
<td>L3</td>
<td>Asia</td>
<td>1 0 0</td>
</tr>
<tr>
<td>L4</td>
<td>America</td>
<td>0 0 1</td>
</tr>
<tr>
<td>L5</td>
<td>Europe</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

Join-Index

- Join index relates the values of the dimensions of a star schema to rows in the fact table.
  - a join index on region maintains for each distinct region a list of ROW-IDs of the tuples recording the sales in the region
- Join indices can span multiple dimensions OR
  - can be implemented as bitmap-indexes (per dimension)
  - use bit-op for multiple-joins
Problem Solved?

• “Find total sales per product-category in our stores in Europe”
  - Join-index will prune ¾ of the data (uniform sales), but the remaining ¼ is still large (several millions transactions)
    • Index is unclustered
  • High level aggregations are expensive!!!!!
    - long scans to get the data
    - hashing or sorting necessary for group-bys

⇒ Long Query Response Times
⇒ Pre-computation is necessary

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

<table>
<thead>
<tr>
<th>Store</th>
<th>Product</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>454</td>
<td>-</td>
<td>-</td>
<td>925</td>
<td>1379</td>
</tr>
<tr>
<td>2</td>
<td>468</td>
<td>800</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1268</td>
</tr>
<tr>
<td>3</td>
<td>296</td>
<td>-</td>
<td>240</td>
<td>-</td>
<td>-</td>
<td>536</td>
</tr>
<tr>
<td>4</td>
<td>652</td>
<td>-</td>
<td>-</td>
<td>745</td>
<td>-</td>
<td>1937</td>
</tr>
<tr>
<td>ALL</td>
<td>1870</td>
<td>800</td>
<td>780</td>
<td>1670</td>
<td>5120</td>
<td></td>
</tr>
</tbody>
</table>

4 Group-bys here:
- (store, product)
- (store)
- (product)
- ()

Need to write 4 queries!!!
The Data Cube Operator (Gray et al)

- All previous aggregates in a single query:

```sql
SELECT LOCATION.store, SALES.product_key, SUM(amount)
FROM SALES, LOCATION
WHERE SALES.location_key=LOCATION.location_key
CUBE BY SALES.product_key, LOCATION.store
```

**Challenge:** Optimize Aggregate Computation

Relational View of Data Cube

<table>
<thead>
<tr>
<th>Store</th>
<th>Product</th>
<th>Sales</th>
<th>Product</th>
<th>sum(amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>ALL</td>
<td>1870 800 780 1670 5120</td>
<td>454</td>
<td>1379</td>
</tr>
<tr>
<td>ALL</td>
<td>1</td>
<td>454 - - 925</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ALL</td>
<td>2</td>
<td>468 800 - -</td>
<td>-</td>
<td>1268</td>
</tr>
<tr>
<td>ALL</td>
<td>3</td>
<td>296 - 240 -</td>
<td>-</td>
<td>536</td>
</tr>
<tr>
<td>ALL</td>
<td>4</td>
<td>652 - 540 745</td>
<td>-</td>
<td>1937</td>
</tr>
</tbody>
</table>

```sql
SELECT LOCATION.store, SALES.product_key, SUM(amount)
FROM SALES, LOCATION
WHERE SALES.location_key=LOCATION.location_key
CUBE BY SALES.product_key, LOCATION.store
```
Data Cube: Multidimensional View

Product
- DVD
- PC
- VCR
- sum

Quarter
- 1Qtr
- 2Qtr
- 3Qtr
- 4Qtr
- sum

Region
- America
- Europe
- Asia
- sum

Total annual sales of DVDs in America

Other Extensions to SQL

- Complex aggregation at multiple granularities (Ross et. al 1998)
  - Compute multiple dependent aggregates

```
SELECT LOCATION.store, SALES.product_key, SUM (amount)
FROM SALES, LOCATION
WHERE SALES.location_key=LOCATION.location_key
CUBE BY SALES.product_key, LOCATION.store: R
SUCH THAT R.amount = max(amount)
```

- Other proposals: the MD-join operator (Chatziantoniou et. al 1999)
Data Cube Computation

- Model dependencies among the aggregates:

  - Most detailed "view"

  can be computed from view (product,store,quarter) by summing-up all quarterly sales

Computation Directives

- Hash/sort based methods (Agrawal et. al. VLDB’96)
  1. Smallest-parent
  2. Cache-results
  3. Amortize-scans
  4. Share-sorts
  5. Share-partitions
Alternative Array-based Approach

- Model data as a sparse multidimensional array
  - partition array into chunks (a small sub-cube which fits in memory).
  - fast addressing based on (chunk_id, offset)
- Compute aggregates in “multi-way” by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces memory access and storage cost.

What is the best traversing order to do multi-way aggregation?

Reality check: too many views!

- $2^n$ views for $n$ dimensions (no-hierarchies)
- Storage/update-time explosion
- More pre-computation doesn’t mean better performance!!!!
How to choose among the views?

- Use some notion of benefit per view
- Limit: disk space or maintenance-time

Hanarayan et al SIGMOD’96:

$$B(v, S) = \sum \left( C_S(u) - C_V(u) \right)$$

Pick views greedily until space is filled

Catch: quadratic in the number of views, which is exponential!!!

View Selection Problem

- Selection is based on a workload estimate (e.g. logs) and a given constraint (disk space or update window)
- NP-hard, optimal selection can not be computed > 4-5 dimensions
  - greedy algorithms (e.g. [Harinarayan96]) run at least in polynomial time in the number of views i.e. exponential in the number of dimensions!!!
- Optimal selection can not be approximated [Karloff99]
  - greedy view selection can behave arbitrary bad
- Lack of good models for a cost-based optimization!
Other Issues

• Fact+Dimension tables in the DW are views of tables stored in the sources
• Lots of view maintenance problems
  – correctly reflect asynchronous changes at the sources
  – making views self-maintainable
• Interactive queries (on-line aggregation)
  – e.g. show running-estimates + confidence intervals
• Computing Iceberg queries efficiently
• Approximation
  – rough-estimates for hi-level aggregates are often good-enough
  – histogram, wavelet, sampling based techniques (e.g. AQUA)