

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

2

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

3

Data Warehouse Architecture

- Extract data from operational data sources
 - clean, transform
- Bulk load/refresh
 - warehouse is offline
- OLAP-server provides multidimensional view
- Multidimensional-olap (Eskbase, oracle express)
- Relational-olap (Redbrick, Informix, Sybase, SQL server)



4

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

5

Examples of OLAP

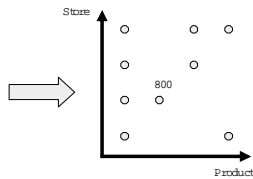
- Comparisons (this period v.s. last period)
 - Show me the sales per region for this year and compare it to that of the previous year to identify discrepancies
- Multidimensional ratios (percent to total)
 - Show me the contribution to weekly profit made by all items sold in the northeast stores between 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 from my northeast region operations

6

Multidimensional Modeling

- Example: compute total sales volume per product and store

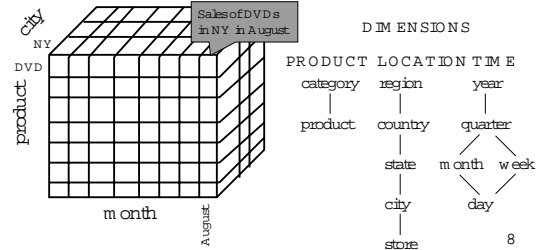
Total Sales	Product				
	1	2	3	4	
Store	1	\$454	-	-	\$925
	2	\$468	\$800	-	-
	3	\$296	-	\$240	-
	4	\$652	-	\$540	\$745



7

Dimensions and Hierarchies

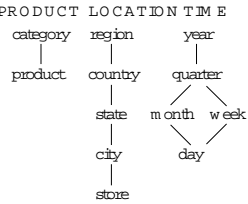
- A cell in the cube may store values (measurements) relative to the combination of the labeled dimensions



8

Common OLAP Operations

- **Rollup**: move up the hierarchy
 - eg 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)



9

Pivoting

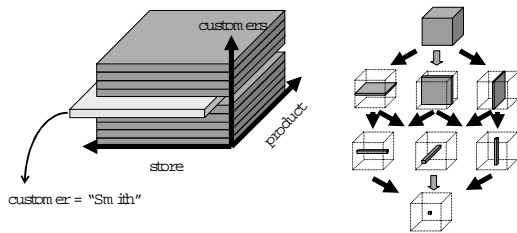
- **Pivoting**: aggregate on selected dimensions
 - usually 2 dims (cross-tabulation)

Sales		Product				
		1	2	3	4	ALL
Store	1	454	-	-	925	1379
	2	468	800	-	-	1268
	3	296	-	240	-	536
	4	652	-	540	745	1937
	ALL	1870	800	780	1670	5120

10

Slice and Dice Queries

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



11

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

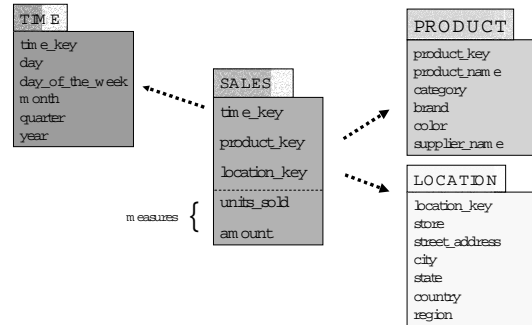
12

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 \$\$\$)

13

Star Schema Example



Advantages of Star Schema

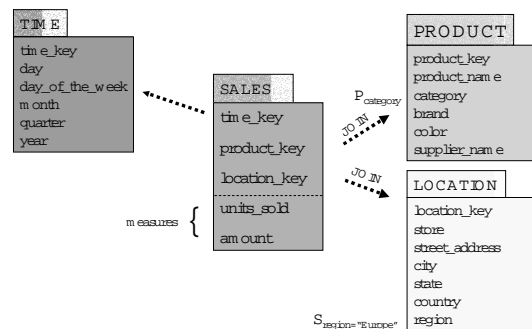
- Facts and dimensions are clearly depicted
 - dimension tables are relatively static, data is loaded (appended mostly) into fact table(s)
 - easy to comprehend (and write queries)
- Find total sales per product-category in our stores in Europe"
- ```

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

```

15

## Star Schema Query Processing



## 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

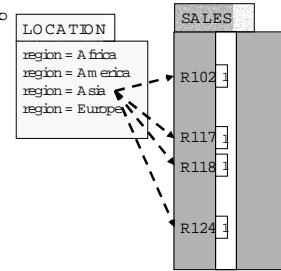
- Mainly intended for an all-cardinality domains

| LOCATION     |         | Index on Region |        |         |
|--------------|---------|-----------------|--------|---------|
| location_key | Region  | Asia            | Europe | America |
| L1           | Asia    | 1               | 0      | 0       |
| L2           | Europe  | 0               | 1      | 0       |
| L3           | Asia    | 1               | 0      | 0       |
| L4           | America | 0               | 0      | 1       |
| L5           | Europe  | 0               | 1      | 0       |

17

## 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-ops for multiple-joins



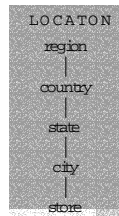
18

## Problem Solved?

- "Find total sales per product-category in our stores in Europe"
  - Join-index will prune  $\frac{3}{4}$  of the data (uniform sales), but the remaining  $\frac{1}{4}$  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



19

## Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

| Sales | Product |     |     |      |      |
|-------|---------|-----|-----|------|------|
|       | 1       | 2   | 3   | 4    | ALL  |
| 1     | 454     | -   | -   | 925  | 1379 |
| 2     | 468     | 800 | -   | -    | 1268 |
| 3     | 296     | -   | 240 | -    | 536  |
| 4     | 652     | -   | 540 | 745  | 1937 |
| ALL   | 1870    | 800 | 780 | 1670 | 5120 |

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

Need to write 4 queries!!!

Sub-totals per store

Sub-totals per product

Totalsales

20

## The Data Cube Operator (Gray et al)

- All previous aggregates in a single query:

```
SELECT LOCATDN store, SALES product_key, SUM (am out)
FROM SALES, LOCATDN
WHERE SALES .lcatdn_key=LOCATDN .lcatdn_key
CUBE BY SALES product_key, LOCATDN store
```

**Challenge:** Optimize Aggregate Computation

21

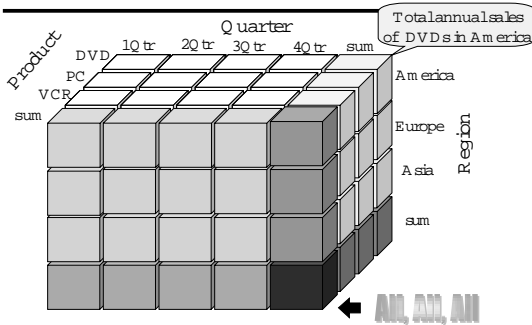
## Relational View of Data Cube

| Sales |     | Product |     |     |      |      | Store | Product_key | am (am out) |
|-------|-----|---------|-----|-----|------|------|-------|-------------|-------------|
|       |     | 1       | 2   | 3   | 4    | ALL  |       |             |             |
| Store | 1   | 454     | -   | -   | 925  | 1379 | 1     | 1           | 454         |
|       | 2   | 468     | 800 | -   | -    | 1268 | 1     | 4           | 925         |
|       | 3   | 296     | -   | 240 | -    | 536  | 2     | 1           | 468         |
|       | 4   | 652     | -   | 540 | 745  | 1937 | 2     | 2           | 800         |
|       | ALL | 1870    | 800 | 780 | 1670 | 5120 | 3     | 1           | 296         |
|       |     |         |     |     |      | 4    | 1     | 625         |             |
|       |     |         |     |     |      | 4    | 3     | 240         |             |
|       |     |         |     |     |      | 4    | 4     | 745         |             |
|       |     |         |     |     |      | 1    | ALL   | 1379        |             |
|       |     |         |     |     |      | 1    | ALL   | 1268        |             |
|       |     |         |     |     |      | 1    | ALL   | 536         |             |
|       |     |         |     |     |      | 1    | ALL   | 1937        |             |
|       |     |         |     |     |      | ALL  | 1     | 1870        |             |
|       |     |         |     |     |      | ALL  | 2     | 800         |             |
|       |     |         |     |     |      | ALL  | 3     | 780         |             |
|       |     |         |     |     |      | ALL  | 4     | 1670        |             |
|       |     |         |     |     |      | ALL  | ALL   | 5120        |             |

```
SELECT LOCATDN store, SALES product_key, SUM (am out)
FROM SALES, LOCATDN
WHERE SALES .lcatdn_key=LOCATDN .lcatdn_key
CUBE BY SALES product_key, LOCATDN store
```

22

## Data Cube: Multidimensional View



23

## Other Extensions to SQL

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

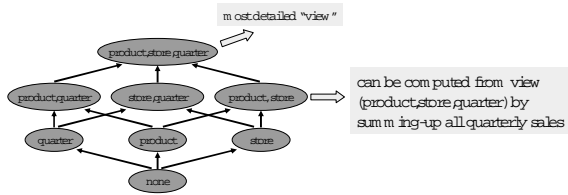
```
SELECT LOCATDN store, SALES product_key, SUM (am out)
FROM SALES, LOCATDN
WHERE SALES .lcatdn_key=LOCATDN .lcatdn_key
CUBE BY SALES product_key, LOCATDN store :R
SUCH THAT R am out = max(am out)
```

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

24

## Data Cube Computation

- Model dependencies among the aggregates:

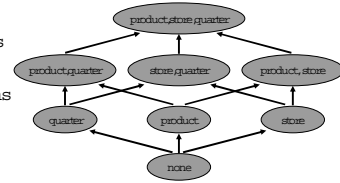


25

## Computation Directives

- Hash/sort-based methods (Agrawal et al. VLDB '96)

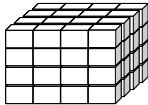
- Smallest-parent
- Cache-results
- Amortize-scans
- Share-sorts
- Share-partitions



26

## Alternative Array-based Approach

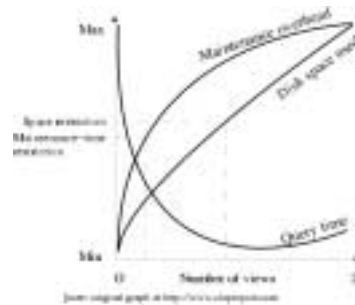
- Model data as a sparse multidimensional array
  - partition array into chunks (as small sub-cube which fits in memory).
  - fast addressing based on (chunk\_id, offset)
- Compute aggregates in "multi-way" by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces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



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

27

## Reality check: too many views!

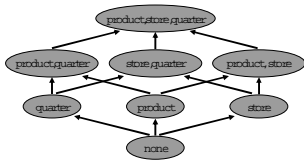


- 2<sup>n</sup> views form dimensions (no hierarchies)
- Storage/update-time explosion
- More computation doesn't mean better performance!!!!

28

## How to choose among the views?

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



Hanarayan et al SIGMOD '96:

$$B(v, S) = \sum_{u \in V, C_v(u) < C_v(u)} (C_S(u) - C_v(u))$$

Pick views greedily until space is filled

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

29

## 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 algorithm (e.g., [Hanarayan96]) 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!

30

## 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)

31