College of Computing

Data Warehousing: Data Summarization Technology

Ling Liu
Professor
College of Computing
Georgia Tech

Adapted from Slides of *Hector Garcia-Molina*, George Kollios, Sudarshan

Outline

- What is a data warehouse?
- Why a warehouse?
- Data Summarization: Models & operations
- Implementing a warehouse
- Future Outlook

What is a Warehouse?

A data warehouse is a

- database that is maintained separately from an operational database
- subject oriented
  - aimed at executive, decision maker
  - often a copy of operational data with value-added data (e.g., summaries, history)
- Integrated
- time-varying
- non-volatile

collection of detailed and summary data used to support the strategic decision making process for the enterprise

[W.H.Inmon]

more

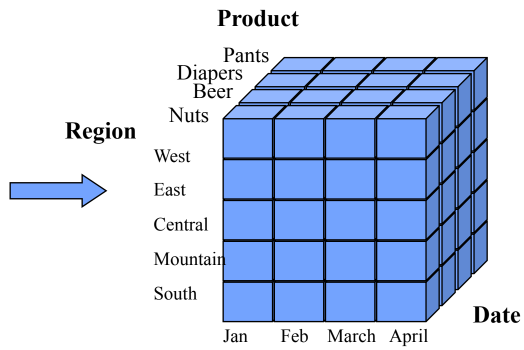
What is a Warehouse?

- Collection of tools
  - gathering data
  - cleansing, integrating, ...
  - querying, reporting, analysis
  - data mining
  - monitoring, administering warehouse

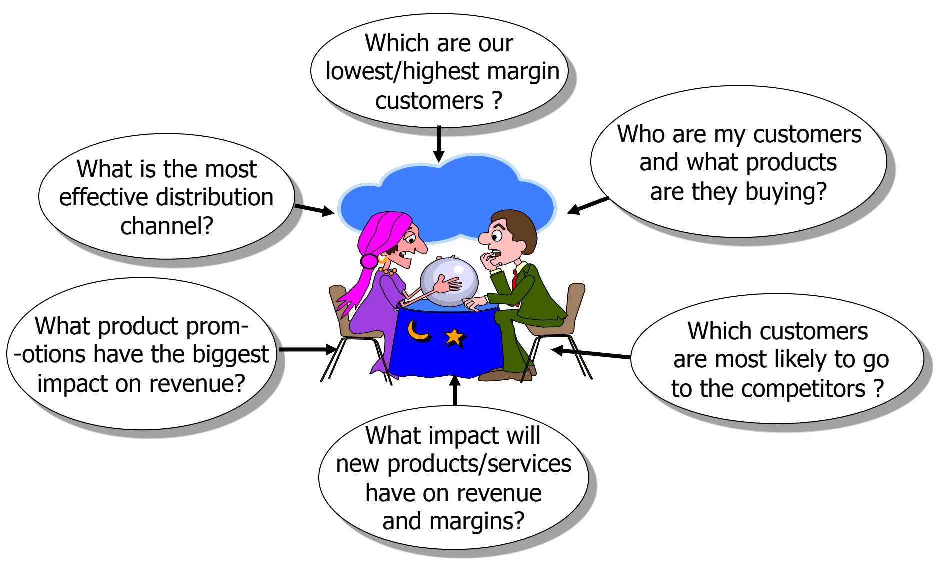
Subject-view

- Example: regional product monthly sales comparison

Product	store	Date	Sale
acron	Rolla,MO	7/3/99	325.24
budwiser	LA,CA	5/22/99	833.92
large pants	NY,NY	2/12/99	771.24
3' diaper	Cuba,MO	7/30/99	81.99



Why Data Warehousing?



Data Warehouse vs. Operational DBMS

- **OLTP (on-line transaction processing)**
  - Major task of traditional relational DBMS
  - Day-to-day operations: purchasing, inventory, banking, manufacturing, payroll, registration, accounting, etc.
  - Describes processing at operational sites
- **OLAP (on-line analytical processing)**
  - Major task of data warehouse system
  - Data analysis and decision making
  - Describes processing at warehouse

Examples of OLAP

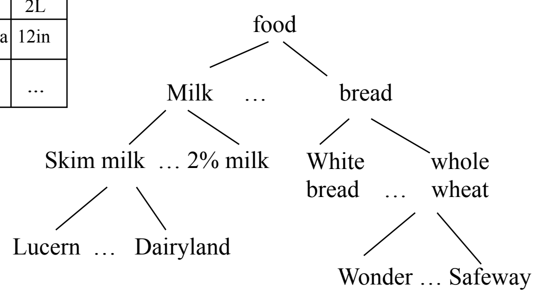
- Comparisons (this period v.s. last period)
  - Show me the sales per region for this year and compare it to that of the previous year to identify discrepancies
- Multidimensional ratios (percent to total)
  - Show me the contribution to weekly profit made by all items sold in the northeast stores between 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
- Customer consolidation (market segments, ad hoc groups)
  - Show me an abbreviated income statement by quarter for the last four quarters for my northeast region operations

Summarization Technique: Attribute-Oriented Induction

- Generalization using Concept hierarchy (taxonomy)

barcode	category	brand	content	size
14998	milk	diaryland	Skim	2L
12998	mechanical	MotorCraft	valve 23a	12in
…	…	…	…	...

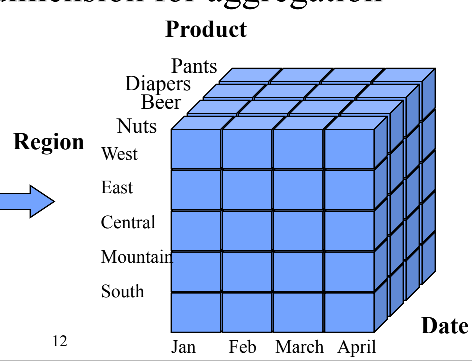
Category	Content	Count
milk	skim	280
milk	2%	98
…	…	...



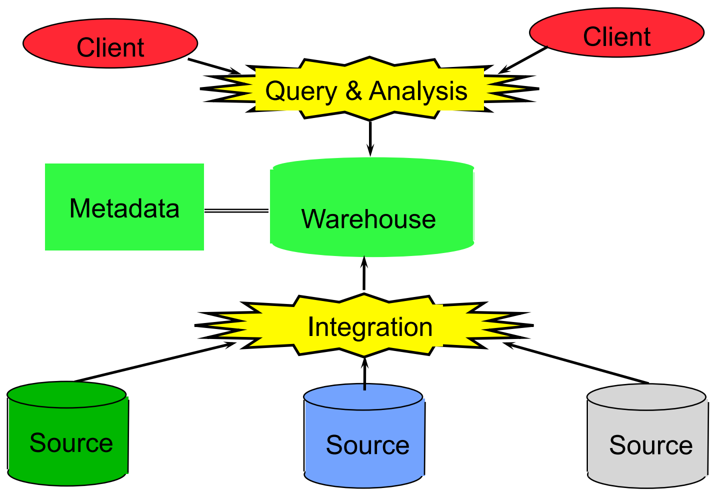
On-Line Analytical Processing

- Data Cube
  - a multidimensonal array
  - Summariztion/aggregation attributes v.s. generalization attributes
  - each generation attribute represents a dimension
  - variable - a special dimension for aggregation
- Example

Product	store	Date	Sale
acron	Rolla,MO	7/3/99	325.24
budwiser	LA,CA	5/22/99	833.92
large pants	NY,NY	2/12/99	771.24
3' diaper	Cuba,MO	7/30/99	81.99



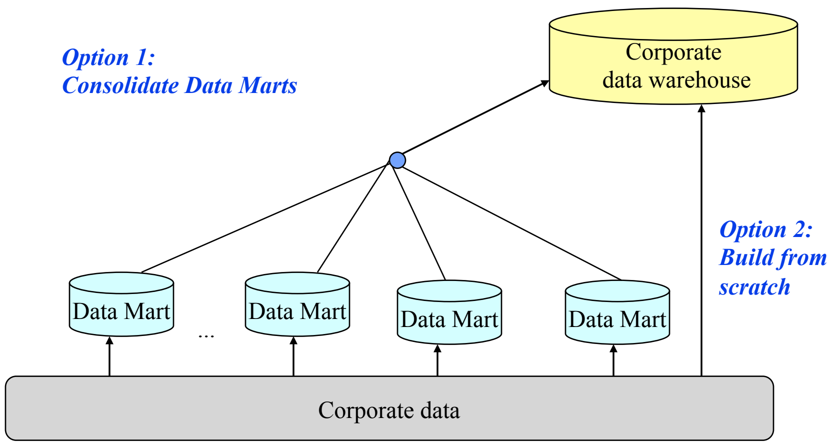
Warehouse Architecture



Advantages of Warehousing

- High query performance
- Queries not visible outside warehouse
- Local processing at sources unaffected
- Can operate when sources unavailable
- Can query data not stored in a DBMS
- Extra information at warehouse
  - Modify, summarize (store aggregates)
  - Add historical information

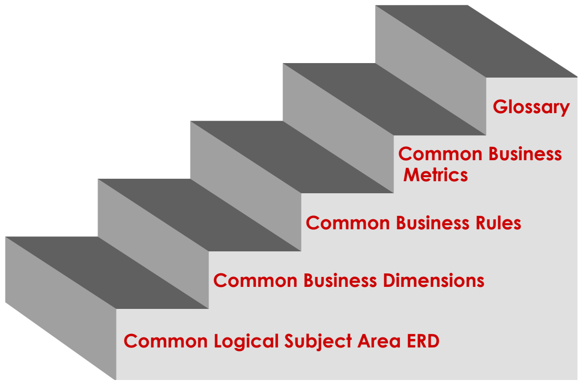
Building a Data Warehouse



Data Marts

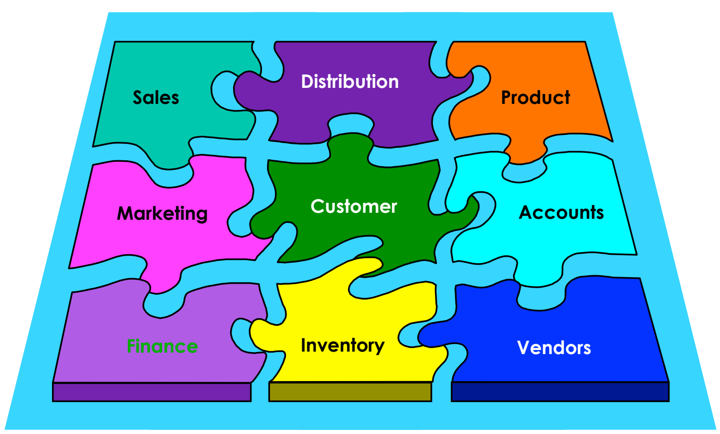
- Smaller warehouses
- Spans part of organization
  - e.g., marketing (customers, products, sales)
- Do not require enterprise-wide consensus
  - but long term integration problems?

An Incremental Approach



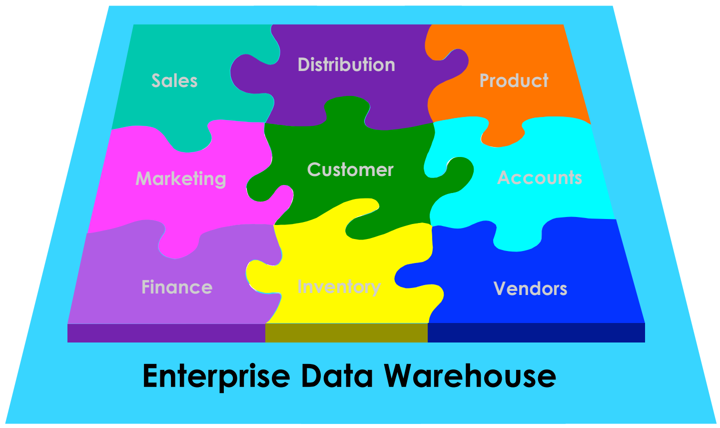
Individual Architected Data Marts

An Incremental Approach

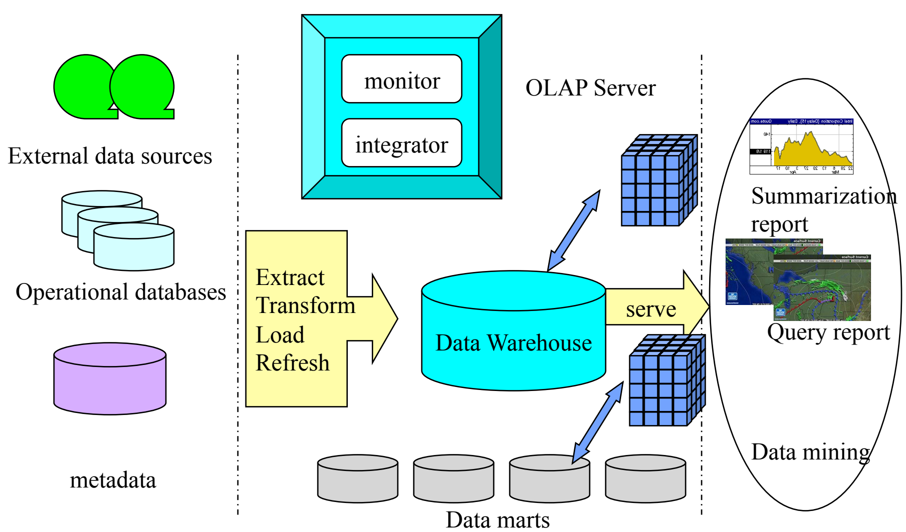


Individual Architected Data Marts

The Eventual Result



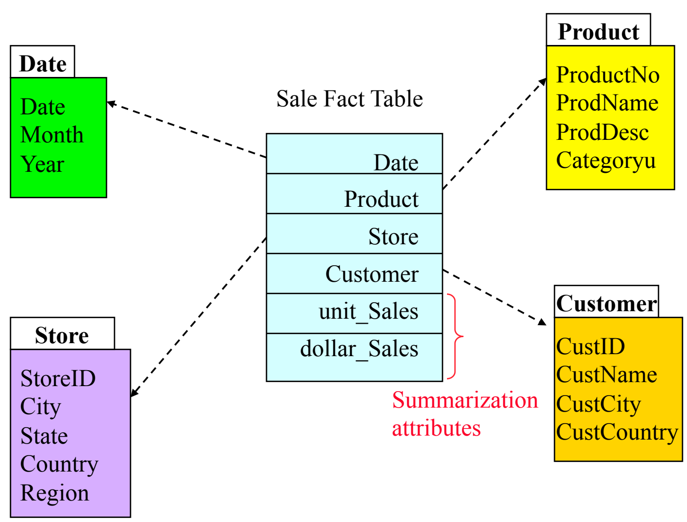
Three Tier Architecture



Data Warehouse Design (1)

- Most of data warehouses use **a star schema** to represent multi-dimensional data model
- Each dimension is represented by **a dimension table** that provides its multidimensional coordinates
  - LOCATION(location_key,store,street_address,city,state,country,region)
  - dimension tables are not normalized
- The star fact table stores **measures** for those coordinates

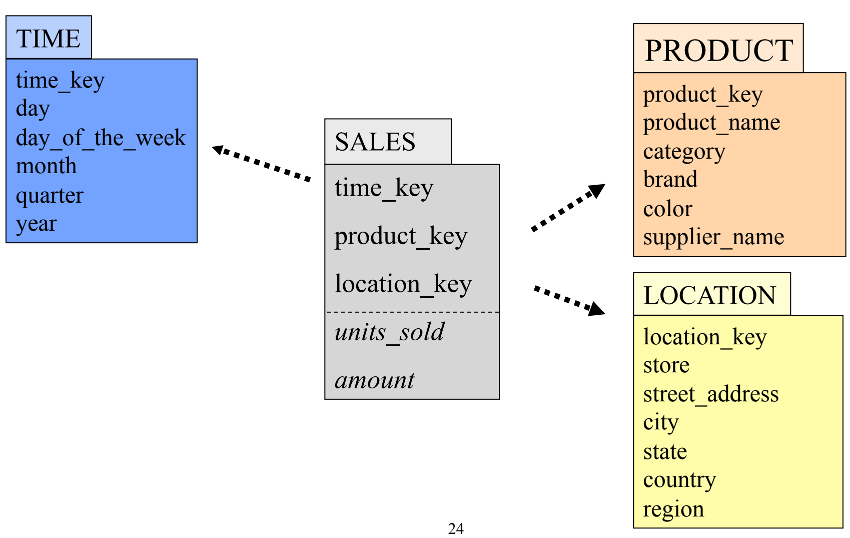
Star Schema: Example 1



Data Warehouse Design (2)

- A fact table
  - connects to all dimension tables with a multiple join. Each tuple in the fact table consists of a pointer to each of the dimensional tables
- 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 $$$)
- The links between the fact table and the dimensional tables form a shape like a star

Star Schema: Example 2

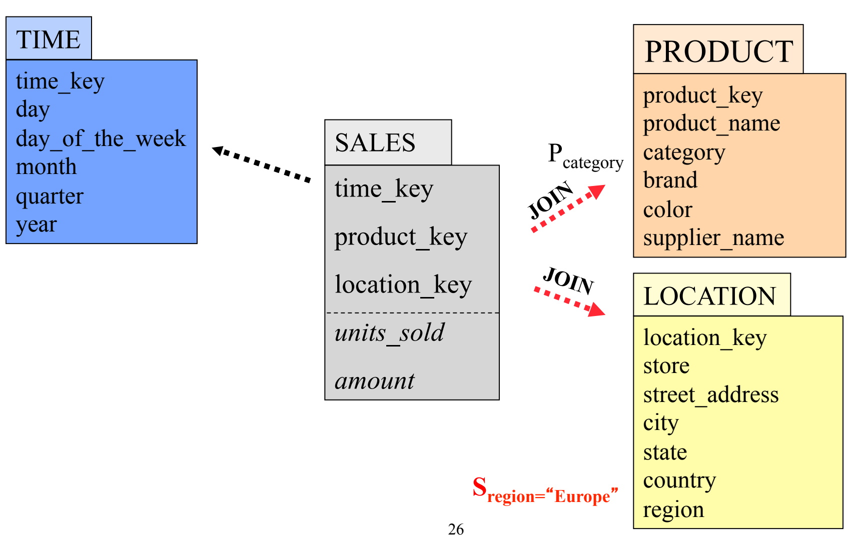


Advantages of Star Schema

- 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)
- Example

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

Star Schema Query Processing



Example

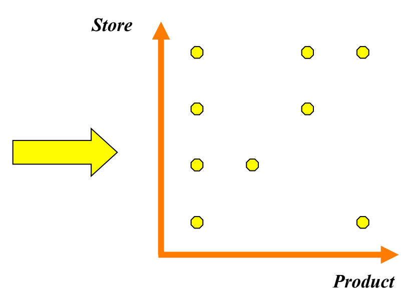
- *“Find total sales per product-category in our stores in Europe”*

```
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
```


Multidimensional Modeling

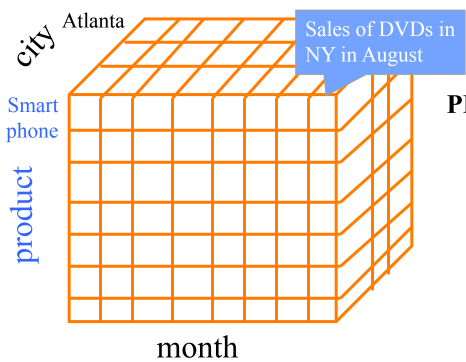
- Example: compute total *sales* volume per *product* and *store*

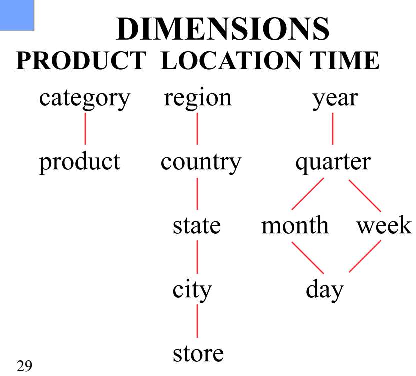
Total Sales		Product			
		1	2	3	4
Store	1	$454	-	-	$925
	2	$468	$800	-	-
	3	$296	-	$240	-
	4	$652	-	$540	$745



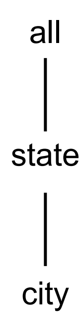
Dimensions and Hierarchies

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



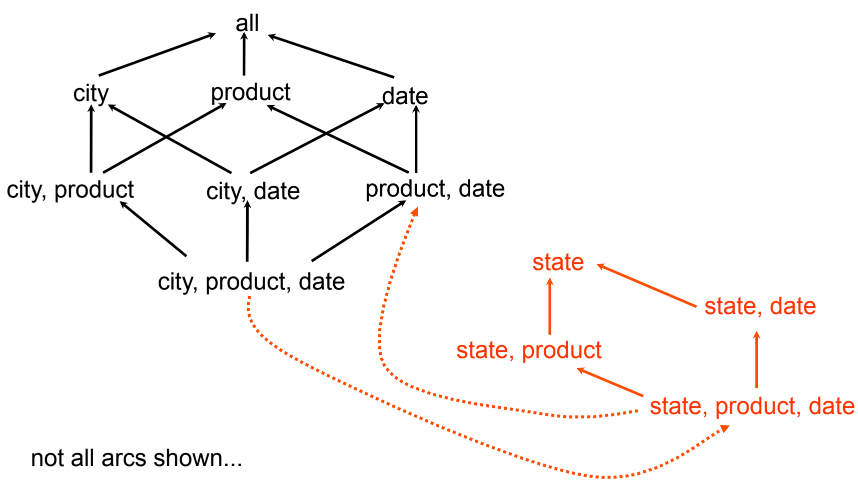


Dimension Hierarchies

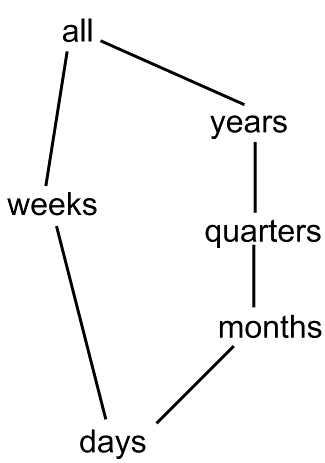


cities	city	state
	c1	GA
	c2	NY

Dimension Hierarchies



Interesting Hierarchy



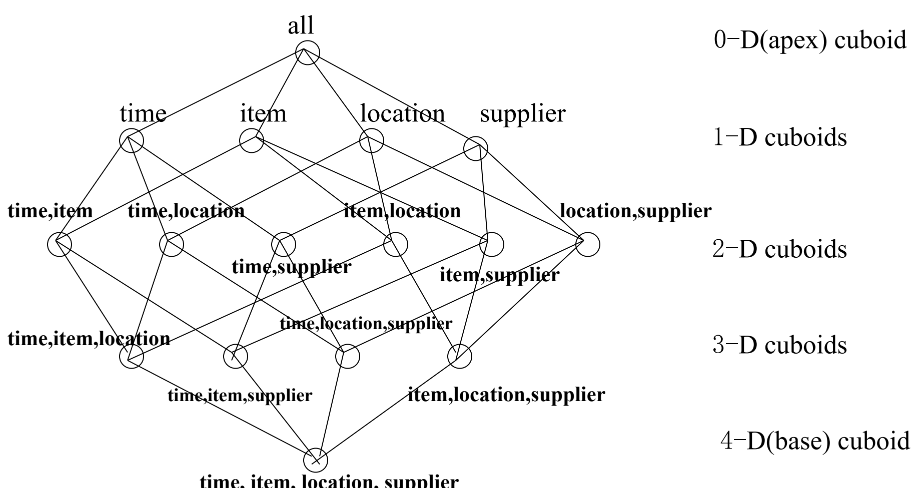
time	day	week	month	quarter	year
	1	1	1	1	2000
	2	1	1	1	2000
	3	1	1	1	2000
	4	1	1	1	2000
	5	1	1	1	2000
	6	1	1	1	2000
	7	1	1	1	2000
	8	2	1	1	2000

conceptual dimension table

From Tables and Spreadsheets to Data Cubes

- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
  - **Dimension tables**, such as product (name, brand, type), or time(day, week, month, quarter, year), store(name, city, state, country)
  - **Fact table** contains measures (such as dollars_sold) and keys to each of the related dimension tables
- In data warehousing literature, an n-D base cube is called a **base cuboid**. The top most 0-D cuboid, which holds the highest-level of summarization, is called the apex cuboid. The lattice of cuboids forms a data cube.

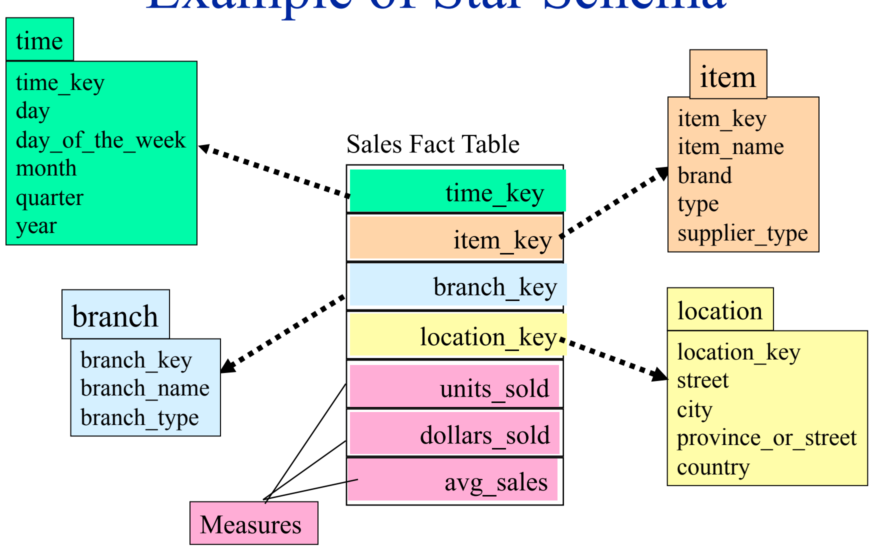
Cube: A Lattice of Cuboids



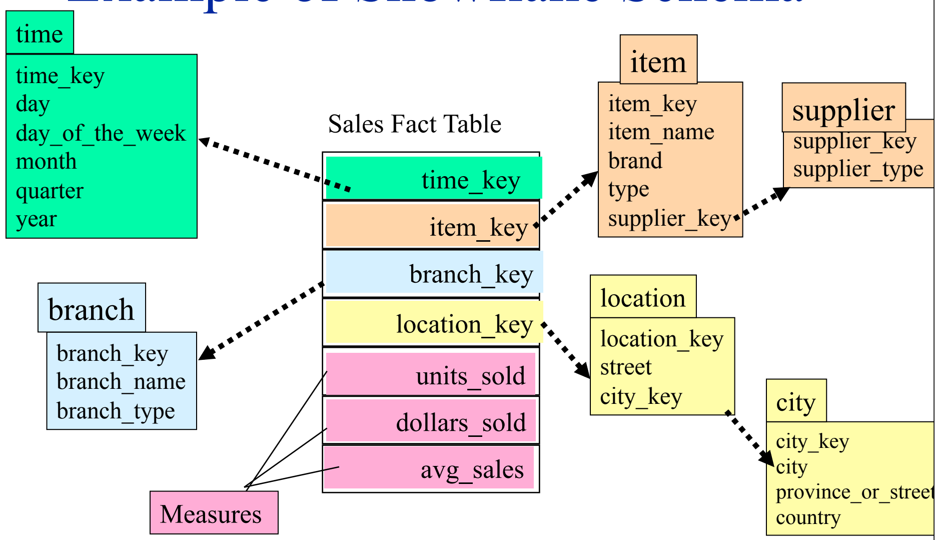
Conceptual Modeling of Data Warehouses

- Modeling data warehouses: dimensions & measures
  - Star schema: A fact table in the middle connected to a set of dimension tables
  - Snowflake schema: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
  - Fact constellations: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation

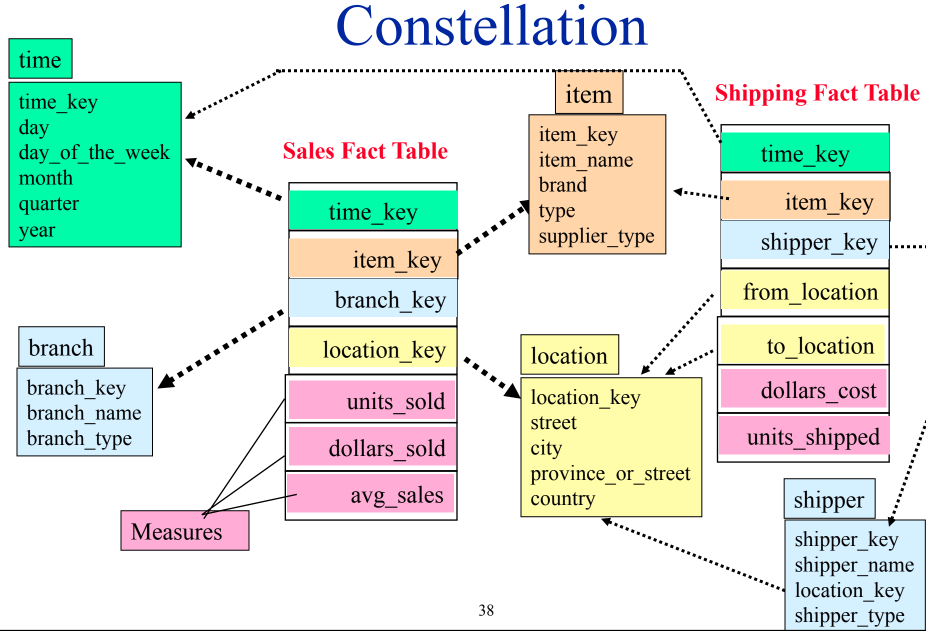
Example of Star Schema



Example of Snowflake Schema



Example of Fact Constellation



A Data Mining Query Language, DMQL: Language Primitives

- Cube Definition (Fact Table)

  define cube <cube_name> [<dimension_list>]: <measure_list>

- Dimension Definition (Dimension Table)

  define dimension <dimension_name> as (<attribute_or_subdimension_list>)

- Special Case (Shared Dimension Tables)
  - First time as "cube definition"
  - define dimension <dimension_name> as <dimension_name_first_time> in cube <cube_name_first_time>

Defining a Star Schema in DMQL

define cube sales_star [time, item, branch, location]:

dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier_type)

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city, province_or_state, country)

Defining a Snowflake Schema in DMQL

define cube sales_snowflake [time, item, branch, location]:

dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier(supplier_key, supplier_type))

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city(city_key, province_or_state, country))

Defining a Fact Constellation in DMQL

define cube sales [time, item, branch, location]:

dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier_type)

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city, province_or_state, country)

define cube shipping [time, item, shipper, from_location, to_location]:

dollar_cost = sum(cost_in_dollars), unit_shipped = count(*)

define dimension time as time in cube sales

define dimension item as item in cube sales

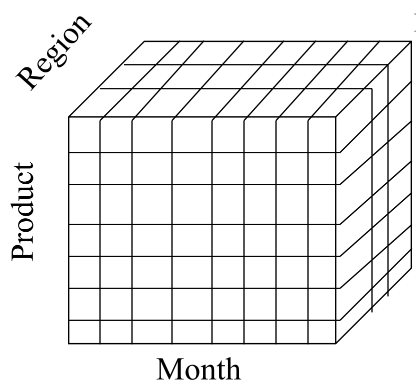
define dimension shipper as (shipper_key, shipper_name, location as location in cube sales, shipper_type)

define dimension from_location as location in cube sales

define dimension to_location as location in cube sales

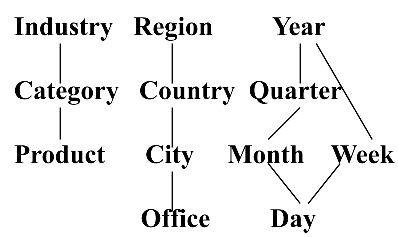
Multidimensional Data

- Sales volume as a function of product, month, and region

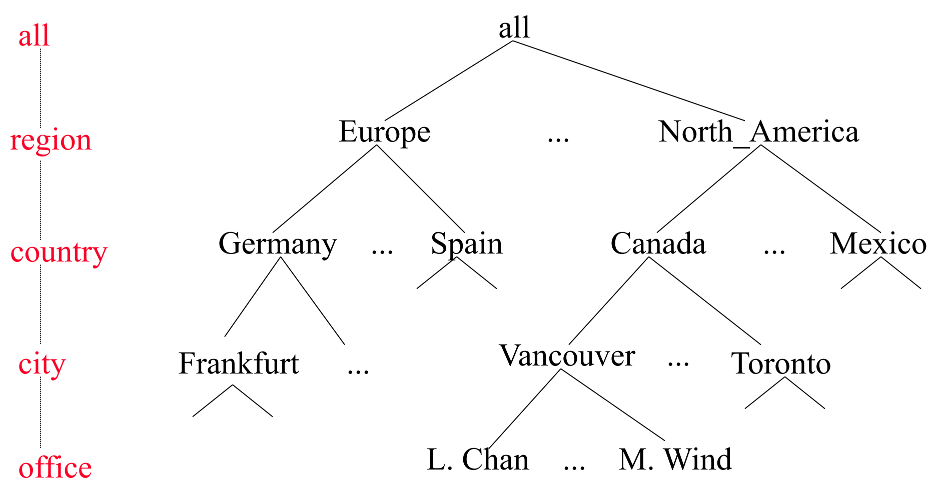


Dimensions: Product, Location, Time

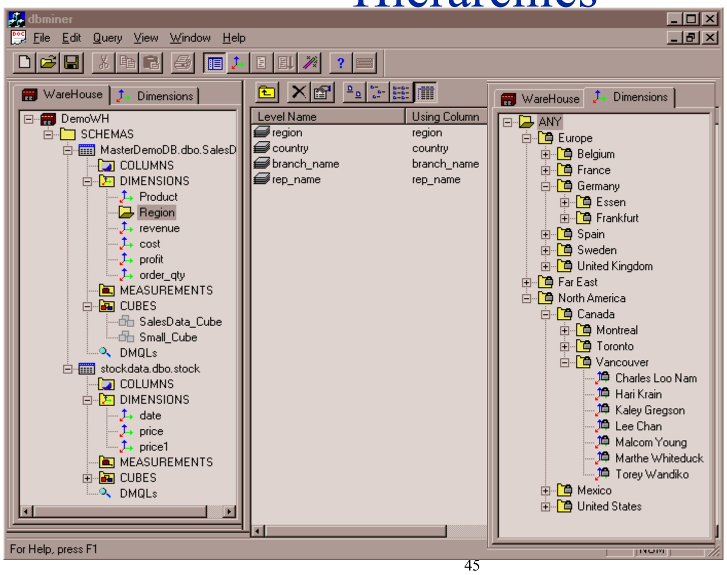
Hierarchical summarization paths



A Concept Hierarchy: Dimension (location)



View of Warehouses and Hierarchies



Relational View of Data Cube

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

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

Store	Product_key	sum(amount)
1	1	454
1	4	925
2	1	468
2	2	800
3	1	296
3	3	240
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

Multiple Simultaneous Aggregates

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

Need to write 4 queries!!!

Cross-Tabulation (products/store)

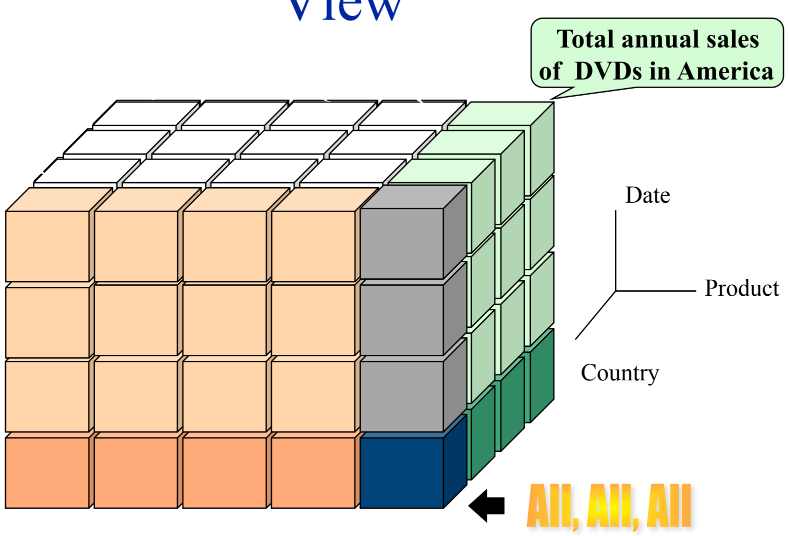
Sales		Product				
		1	2	3	4	ALL
Store	1	454	-	-	925	1379
	2	468	800	-	-	1268
	3	296	-	240	-	536
	4	652	-	540	745	1937
	ALL	1870	800	780	1670	5120

Sub-totals per store

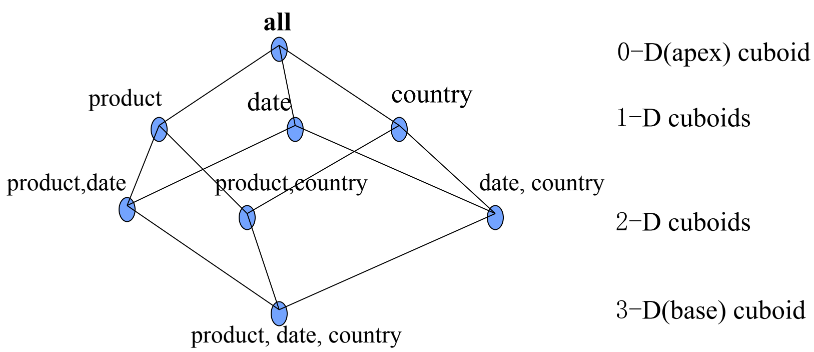
Total sales

Sub-totals per product

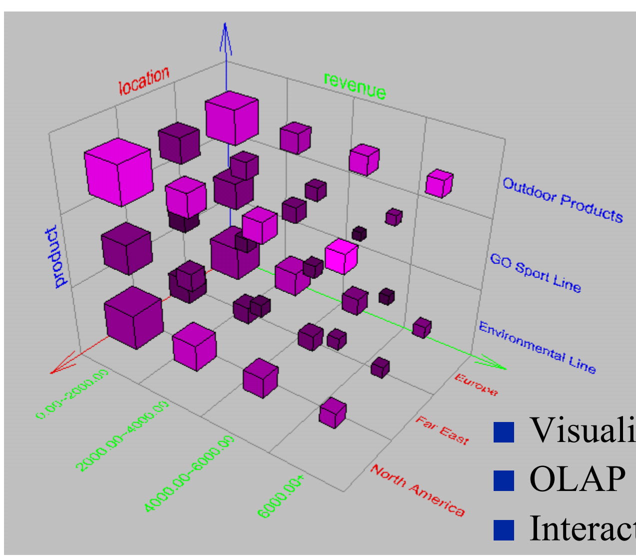
Data Cube: Multidimensional View



Cuboids Corresponding to the Cube



Browsing a Data Cube



- Visualization
- OLAP capabilities
- Interactive manipulation

Warehouse Models

- Data Models
  - relations
  - stars & snowflakes
  - cubes

Typical OLAP Operations

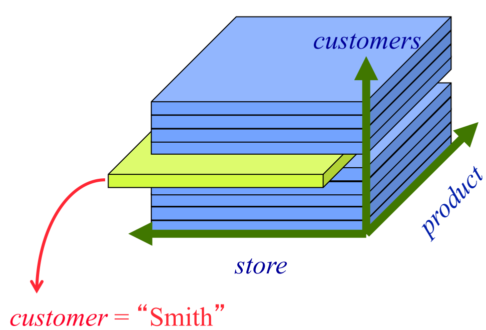
- Roll up (drill-up): summarize data
  - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice:
  - *project and select*
- Pivot (rotate):
  - *reorient the cube, visualization, 3D to series of 2D planes.*
- Other operations
  - *drill across: involving (across) more than one fact table*
  - *drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*

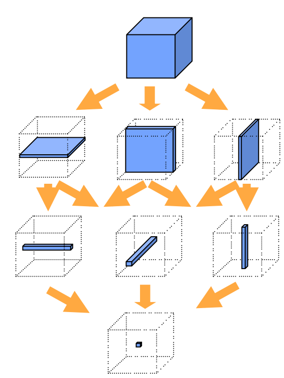
Common OLAP Operations (1)

- Slice: **selection condition set on one of the cube dimensions.**
  - Find product sales of atlanta
  - Find 2006 sales of all states
- **Dice: selection condition set on all three dimensions**
  - Find digital camera sale of 2007 in Atlanta

Slice and Dice Queries

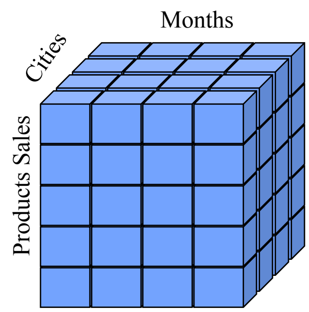
- Slice and Dice: select and project on one or more dimensions



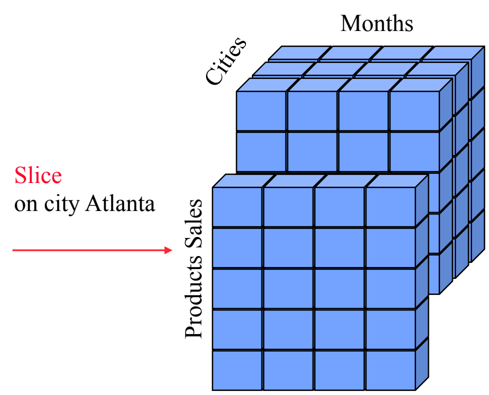


Example of Slice

- Data Warehouses

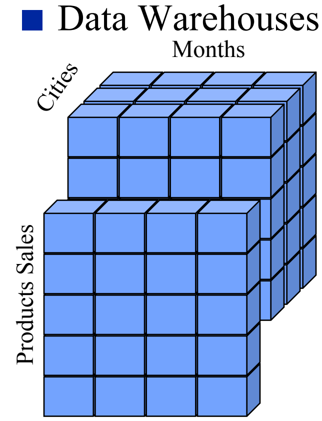


Multi-dimensional data cube

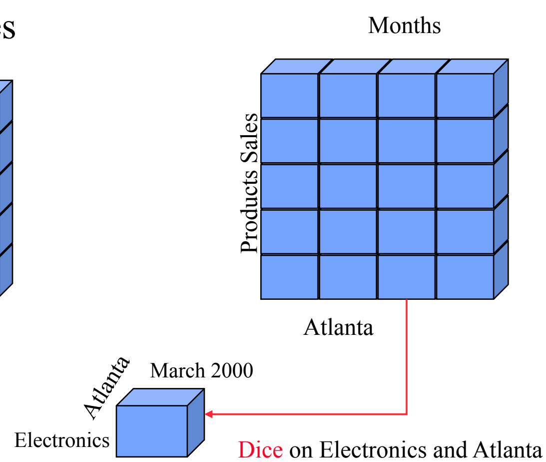


Example of Dice

- Data Warehouses

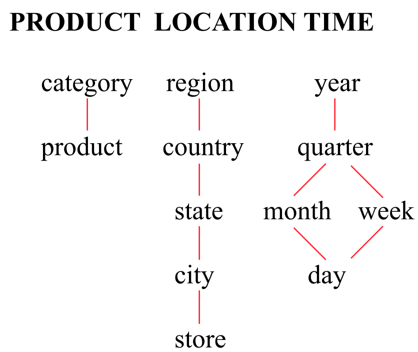


Multi-dimensional data cube



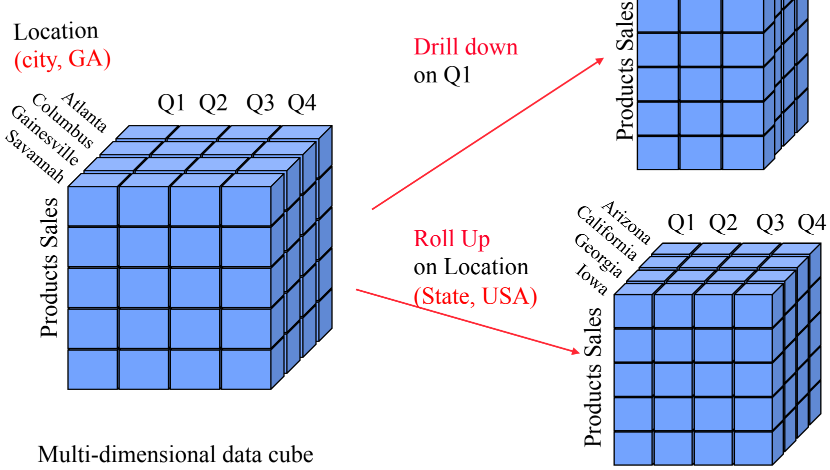
Common OLAP Operations (2)

- Roll-up: **move up the hierarchy**
  - 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)



Examples of OLAP and DW

- Data Warehouses



Pivoting

- <u>Pivoting</u>: aggregate on selected dimensions
  - usually 2 dims (cross-tabulation)

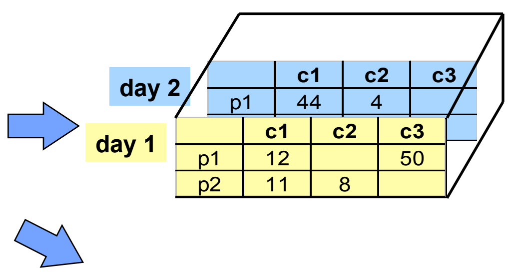
Sales		Product				
		1	2	3	4	ALL
Store	1	454	-	-	925	1379
	2	468	800	-	-	1268
	3	296	-	240	-	536
	4	652	-	540	745	1937
	ALL	1870	800	780	1670	5120

Pivoting: An example

Fact table view:

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

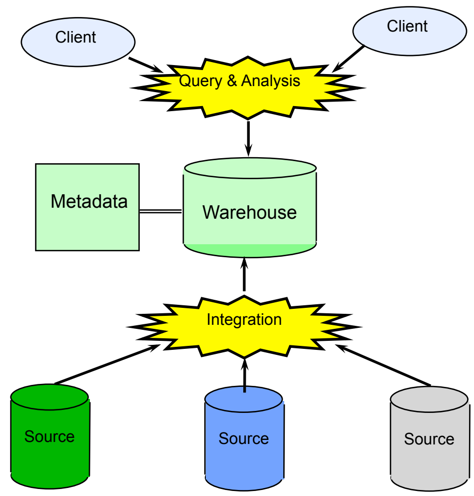
Multi-dimensional cube:



	c1	c2	c3
p1	56	4	50
p2	11	8	

Implementing a Warehouse

- *Monitoring*:
  - Sending data from sources
- *Integrating*:
  - Loading, cleansing, deriving data ...
- *Processing*:
  - Query processing, indexing, ...
- *Managing*:
  - Metadata, Design, ...



Query performance issues

- On Line Transaction Processing.

  Read & write from/to database.

- Data Warehousing, Decision Support System.

  Read mostly environments, with high selectivity factor.

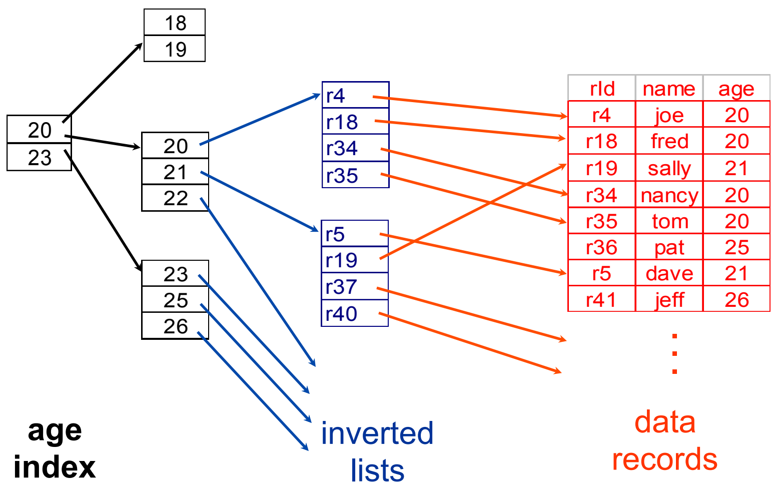
Overview of Index Data Structures

- One dimensional index data structures:
  - Total order for one-dimension
  - Hash-based:
    - *Optimised for exact match queries, e.g. jetE = 106*
  - Tree-based:
    - *Optimised for range queries, e.g. jetE < 106*
    - *Most widely used: B+-tree (1972):*
- Multidimensional index data structures
  - No total order for all dimensions
  - Hash-based:
    - *Grid-File, Bang-File, ...*
  - Tree based:
    - *R-Trees, Pyramid-Tree, ...*
  - Bitmap Indices:
    - *Applied in Data Warehouses for typical read-only environments*

Index Structures

- Traditional Access Methods
  - B-trees, hash tables, R-trees, grids, …
- Popular in Warehouses
  - inverted lists
  - bit map indexes
  - join indexes
  - text indexes

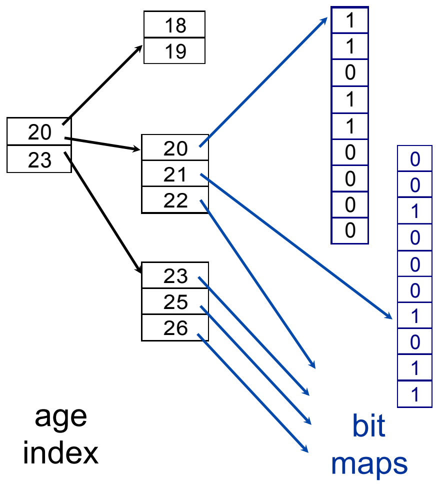
Inverted Lists



Using Inverted Lists

- Query:
  - Get people with age = 20 and name = "fred"
- List for age = 20: r4, r18, r34, r35
- List for name = "fred": r18, r52
- Answer is intersection: r18

Bit Maps



id	name	age
1	joe	20
2	fred	20
3	sally	21
4	nancy	20
5	tom	20
6	pat	25
7	dave	21
8	jeff	26

⋮

data records

Using Bit Maps

- Query:
  - Get people with age = 20 and name = “fred”
- List for age = 20: 1101100000
- List for name = “fred”: 0100000001
- Answer is intersection: 010000000000

- Good if domain cardinality is not too big (small)
- Bit vectors can be compressed

Bitmap Index

- A bitmap is simply an array of bits
  - Consists of a collection of bitmap vectors each created to represent a distinct value.
  - More than one conditions in a query can be replied by Boolean operation on the respective bitmaps.
- Bitmap indices are a special type of index designed for efficient querying on multiple keys

Properties:

- Suited for low cardinality column.
- Utilizes bitwise operation.
- Easy to build and add new indexed value.
- Whole bitmap segment is locked at index updating.
- Less space for storing indexes. More indexes can be cached in memory.

Bitmap Indices (Cont.)

- A bitmap is simply an array of bits
- In its simplest form a bitmap index on an attribute has a bitmap for each value of the attribute
  - Bitmap has as many bits as records
  - In a bitmap for value v, the bit for a record is 1 if the record has the value v for the attribute, and is 0 otherwise
  - The cardinalty of the Attribute is the number of bitmap vectors the bitmap index maintains.

record number	*name*	*gender*	*address*	*income _level*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Bitmaps for *gender*	
m	10010
f	01101

Bitmaps for *income_level*	
L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Bitmap Indices (Cont.)

- Bitmap indices are useful for queries on multiple attributes
  - not particularly useful for single attribute queries
- Queries are answered using bitmap operations
  - Intersection (and), Union (or), Complementation (not)
- Each operation takes two bitmaps of the same size and applies the operation on corresponding bits to get the result bitmap
  - E.g. 100110 AND 110011 = 100010
    100110 OR 110011 = 110111
    NOT 100110 = 011001
  - Males with income level L1:
    10010 AND 10100 = 10000
    - *Can then retrieve required tuples.*
    - *Counting number of matching tuples is even faster*

Bitmap Indices (Cont.)

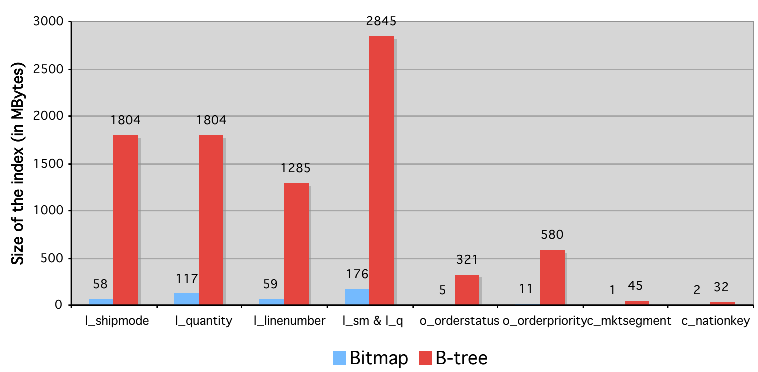
- Bitmap indices generally very small compared with relation size
  - E.g. if record is 100 bytes, space for a single bitmap is 1/800 of space used by relation.
    - *If number of distinct attribute values is 8, bitmap is only 1% of relation size (8*1/800)*
- Deletion needs to be handled properly
  - Existence bitmap to note if there is a valid record at a record location
  - Needed for complementation
    - *not(*A=v*):* (NOT bitmap-A-v) AND ExistenceBitmap
- Should keep bitmaps for all values, even null value
  - To correctly handle SQL null semantics for NOT(*A*=*v*):
    - *intersect above result with (NOT* bitmap-A-Null*)*

Advantages

- Compact size.
- Efficient hardware support for bitmap operations (AND, OR, XOR, NOT).
- Fast search.
- Multiple differentiate bitmap indexes for different kind of queries.

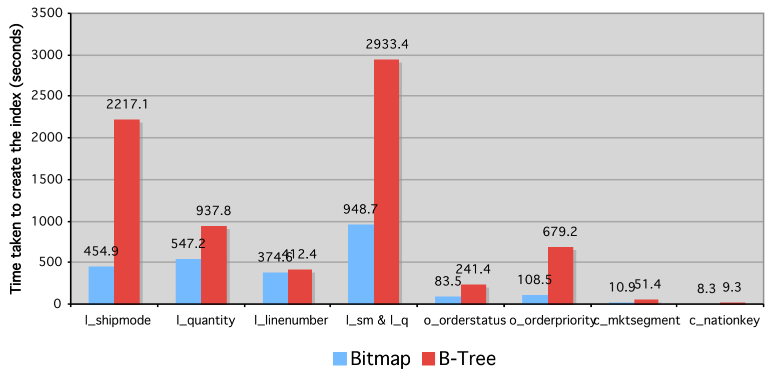
On-disk Bitmap Index - Index Size Performance

- Index size - Size of bitmap index is a fraction of B-tree index



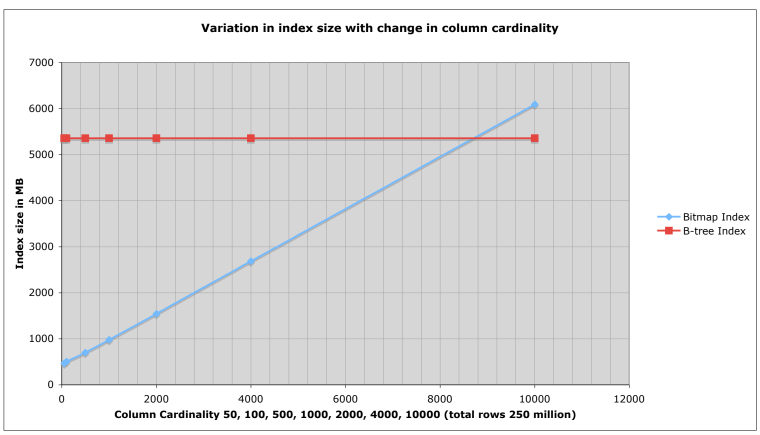
On-disk Bitmap Index - Creation Time Performance

- Index creation time: Up to 7 times faster index creation



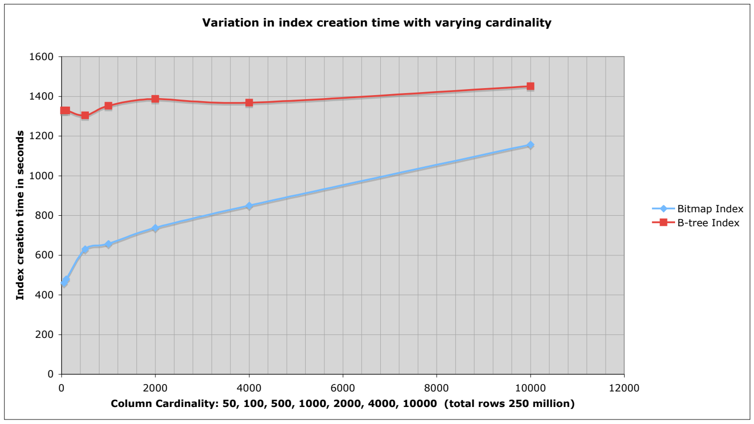
On-disk Bitmap Index - Performance with varying cardinality

- Index size with varying cardinality: Total rows 250 million



On-disk Bitmap Index - Performance with varying cardinality

- Index creation time with varying cardinality: Total rows 250 million



On-disk Bitmap Index - Query Performance

Query 1

```
SELECT sum(lineitem.l_discount)
FROM
lineitem, orders, customer, nation
WHERE
nation.n_name='UNITED STATES' AND
customer.c_mktsegment='AUTOMOBILE' AND
orders.o_orderstatus='P' AND
orders.o_orderpriority='2-HIGH' AND
lineitem.l_quantity=5 AND
lineitem.l_shipmode='AIR' AND
lineitem.l_linenumber=5 AND
customer.c_custkey=orders.o_custkey AND
orders.o_orderkey=lineitem.l_orderkey AND
nation.n_nationkey=customer.c_nationkey;
```

Query 2

```
SELECT avg(lineitem.l_tax)
FROM
lineitem, orders
WHERE
orders.o_orderstatus='F' AND
orders.o_orderpriority='4-NOT SPECIFIED' AND
lineitem.l_linenumber=5 AND
lineitem.l_shipmode='TRUCK' AND
lineitem.l_quantity=2 AND
orders.o_orderkey=lineitem.l_orderkey;
```

Query 3

```
SELECT count(*) FROM lineitem WHERE l_linenumber=1;
```

Query 4

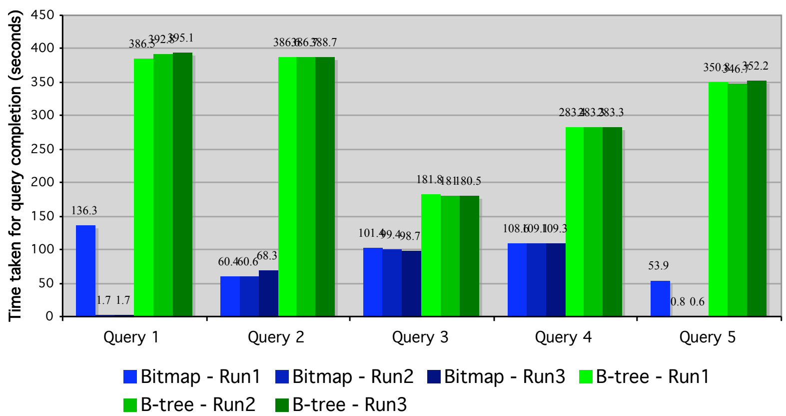
```
SELECT count(*) FROM lineitem WHERE l_linenumber in (1,2) AND l_shipmode IN ('RAIL','TRUCK');
```

Query 5

```
SELECT count(*) FROM lineitem WHERE l_linenumber=5 AND l_shipmode='RAIL' AND l_quantity=18;
```

On-disk Bitmap Index - Query Performance

- Query Performance:
  - Run1, Run2 and Run3 indicate that the same query has been run consecutively three-times



Pros and Cons of Bitmap Indices

- Pros:
  - Easy to build and to maintain
  - Easy to identify records that satisfy a complex multi-attribute predicate (multi-dim. ad-hoc queries)
  - Very space efficient for attributes with low cardinality (number of distinct attribute values, e.g. “Yes”, “No”)
- Cons:
  - Space inefficient for attributes with high cardinality
  - A possible solution:
    - *Bitmap Encoding + Bitmap Compression*

Problems with Bitmap Indexes

- Space inefficient for attributes with high cardinality (sparsity of bitmap vectors)
- Increase the complexity of the software

Solution:

1. Bitmap Encoding
2. Bitmap Compression

Encoded bitmap indexing

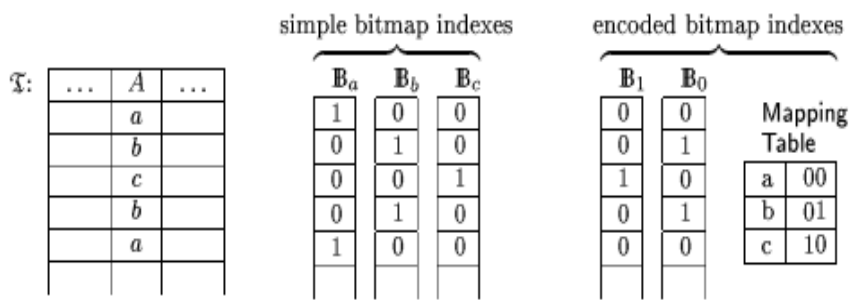


Figure 2: An example of encoded bitmap indexing

Encoded Bitmap Indexes

- We assume that our attribute domain is given by the table T is {a, b, c}.
- The encoding schema of EBI is stored in a separate table called mapping table and simply encodes the values from a SBI by means of Huffman encoding ($\log_2 m$ for m bitmaps on attribute X).
- Therefore reduces the number of bitmaps vectors. In particular, we use only ceil($\log_2 3$)= 2 Encoded Bitmap vectors instead of 3 simple bitmap vectors.
- This means that 2 bits are used to encode the domain {a, b, c}.

Encoded Bitmap Indexes

- We assume that we have a fact table SALES with N tuples and a dimension table PRODUCT with 12,000 different products.
- If we build a simple bitmap index (SBI)on PRODUCT, It will require 12,000 bitmap vectors of N bits in length.
- However, if we use encoded bitmap indexing (EBI) we only need ceil($\log_2 12{,}000$)= 14 bitmap vectors plus a mapping table which is a very significant reduction of the space complexity.

Applications and variations of encoding indexing

- Hierarchy encoding
- Total order preserving
- Using encoding indexes for range encoding

Mapping Table

[6,8)	000
[8,10)	001
[10,12)	101
[12,13)	100
[13,16)	010
[16,20)	110

(a)Range encoding

$$
\begin{aligned}
6 \le A < 10 &\quad : \quad \mathbf{B}_2'\mathbf{B}_1' \\
8 \le A < 12 &\quad : \quad \mathbf{B}_1'\mathbf{B}_0 \\
10 \le A < 13 &\quad : \quad \mathbf{B}_2\mathbf{B}_1' \\
16 \le A < 20 &\quad : \quad \mathbf{B}_2\mathbf{B}_1
\end{aligned}
$$

(b)Retrieval functions

Space time tradeoff of bitmap indexes, for selection queries.

- Space optimal bitmap index.
- Time optimal bitmap index under a given space constraint.
- Bitmap index with optimal space time tradeoff.
- Time optimal bitmap index.

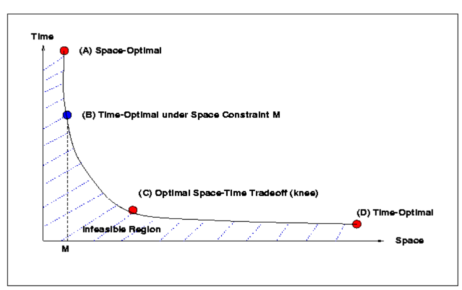
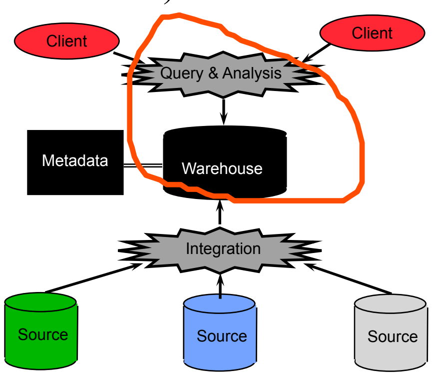


Figure 2: Space-Time Tradeoff Issues

Processing

- ROLAP (Relational OLAP) servers vs. MOLAP servers (Multi-dimensional OLAP)
- Index Structures
- What to Materialize?
- Algorithms



Materialized Views

- Define new warehouse relations using SQL expressions

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

product	id	name	price
	p1	bolt	10
	p2	nut	5

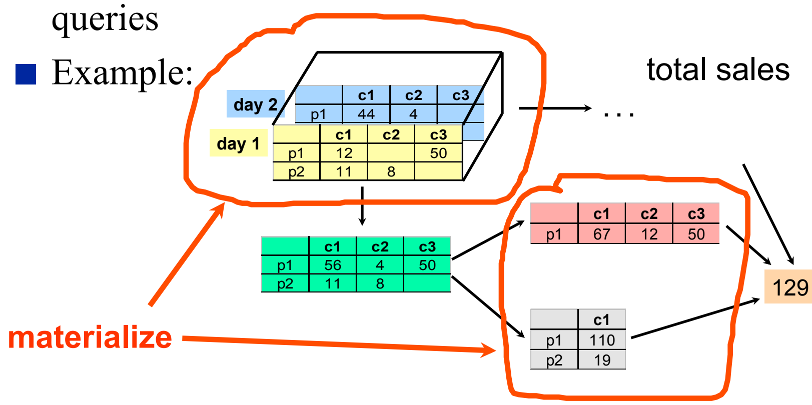
joinTb	prodId	name	price	storeId	date	amt
	p1	bolt	10	c1	1	12
	p2	nut	5	c1	1	11
	p1	bolt	10	c3	1	50
	p2	nut	5	c2	1	8
	p1	bolt	10	c1	2	44
	p1	bolt	10	c2	2	4

does not exist at any source

View Materialization

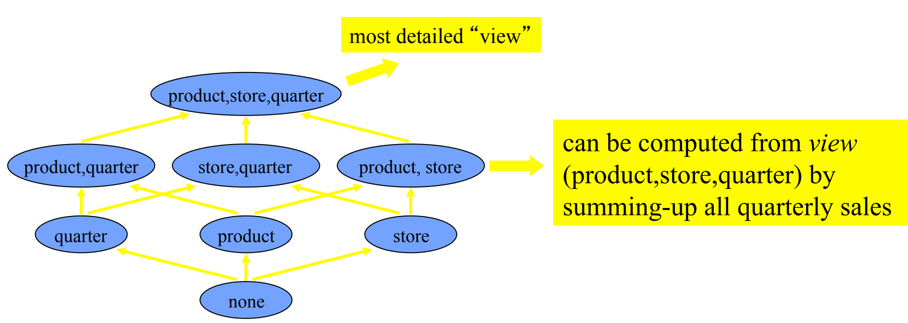
What to Materialize?

- Store in warehouse results useful for common queries
- Example:

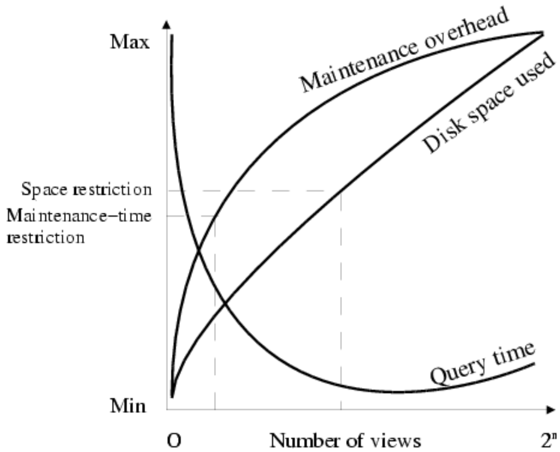


Data Cube Computation

- Model dependencies among the aggregates:



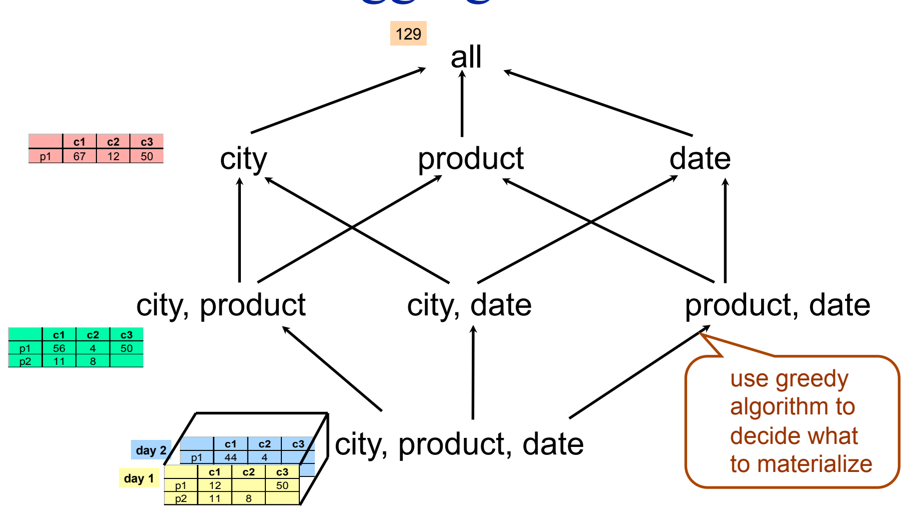
Reality check:too many views!



[note: original graph at http://www.olapreport.com

- 2^n views for n dimensions (no-hierarchies)
- Storage/update-time explosion
- More pre-computation doesn’t mean better performance!!!!

Cube Aggregates Lattice

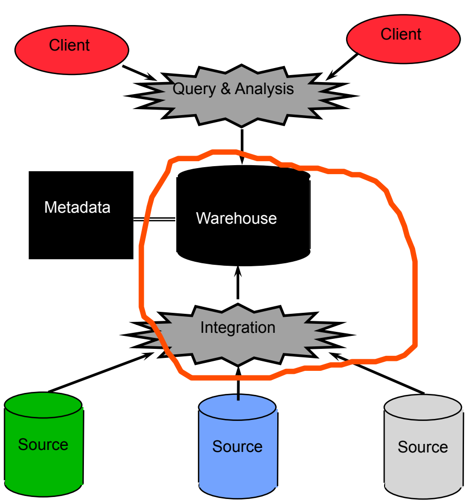


Materialization Factors

- Type/frequency of queries
- Query response time
- Storage cost
- Update cost

Integration

- Data Cleaning
- Data Loading
- Derived Data



Data Cleaning

- Migration (e.g., yen ⇨ dollars)
- Scrubbing: use domain-specific knowledge (e.g., social security numbers)
- Fusion (e.g., mail list, customer merging)

billing DB → customer1(Joe) → merged_customer(Joe)

service DB → customer2(Joe) → merged_customer(Joe)

- Auditing: discover rules & relationships (like data mining)

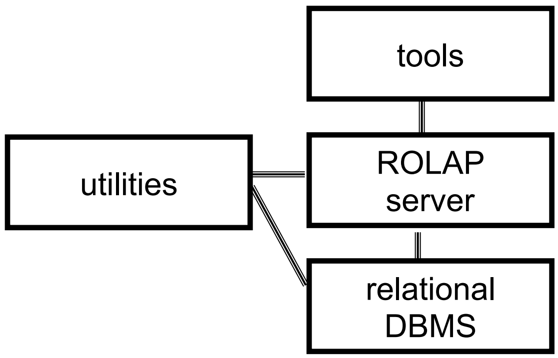
Loading Data

- Incremental vs. refresh
- Off-line vs. on-line
- Frequency of loading
  - At night, 1x a week/month, continuously
- Parallel/Partitioned load

ROLAP Server

- Relational OLAP Server

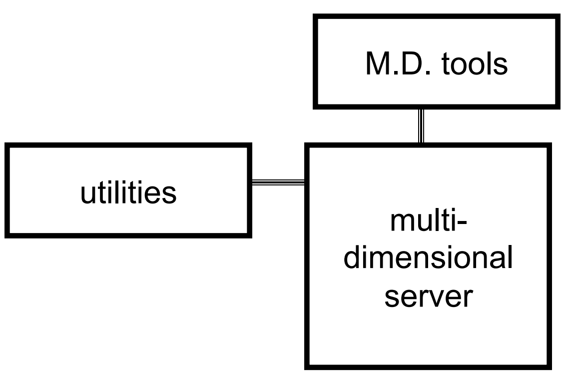
sale	prodId	date	sum
	p1	1	62
	p2	1	19
	p1	2	48

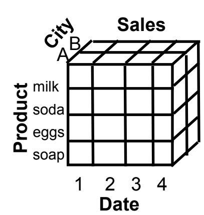


Special indices, tuning;
Schema is "denormalized"

MOLAP Server

- Multi-Dimensional OLAP Server





could also sit on relational DBMS

Data Warehousing

- Growing industry: $8 billion in 1998, predicted > > $30~50 billions today
- Range from desktop to huge:
  - Walmart: 900-CPU, 2,700 disk, 23TB Teradata system
- Lots of buzzwords, hype
  - slice & dice, rollup, MOLAP, pivot, ...

Questions?

References (I)

- S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In Proc. 1996 Int. Conf. Very Large Data Bases, 506-521, Bombay, India, Sept. 1996.
- D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. In Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data, 417-427, Tucson, Arizona, May 1997.
- R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data, 94-105, Seattle, Washington, June 1998.
- R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In Proc. 1997 Int. Conf. Data Engineering, 232-243, Birmingham, England, April 1997.
- K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. In Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99), 359-370, Philadelphia, PA, June 1999.
- S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26:65-74, 1997.
- OLAP council. MDAPI specification version 2.0. In http://www.olapcouncil.org/research/apily.htm, 1998.
- J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Mining and Knowledge Discovery, 1:29-54, 1997.

References (II)

- V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data, pages 205-216, Montreal, Canada, June 1996.
- Microsoft. OLEDB for OLAP programmer's reference version 1.0. In http://www.microsoft.com/data/oledb/olap, 1998.
- K. Ross and D. Srivastava. Fast computation of sparse datacubes. In Proc. 1997 Int. Conf. Very Large Data Bases, 116-125, Athens, Greece, Aug. 1997.
- K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. In Proc. Int. Conf. of Extending Database Technology (EDBT'98), 263-277, Valencia, Spain, March 1998.
- S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In Proc. Int. Conf. of Extending Database Technology (EDBT'98), pages 168-182, Valencia, Spain, March 1998.
- E. Thomsen. OLAP Solutions: Building Multidimensional Information Systems. John Wiley & Sons, 1997.
- Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data, 159-170, Tucson, Arizona, May 1997.