DECISION SUPPORT SYSTEMS (DSS)

- better decisions, faster

- thriving vs. just surviving

  e.g.

  improved product design

  target the right market

- Based on enterprise-wide data

PROBLEMS/CHALLENGES FOR DSS

- No shortage of data
- Collection vs. analysis goals
- Problems relate to
  - access
  - consistency
  - timeliness
  - accuracy

''We have mountains of data, but we can't get access to them.''

''We want to see comparison of yesterdays sales compared to same day last week, month, etc. by mid-morning.''

''Everyone knows that some of the supplier data is dirty.''

- Data Warehousing addresses these challenges

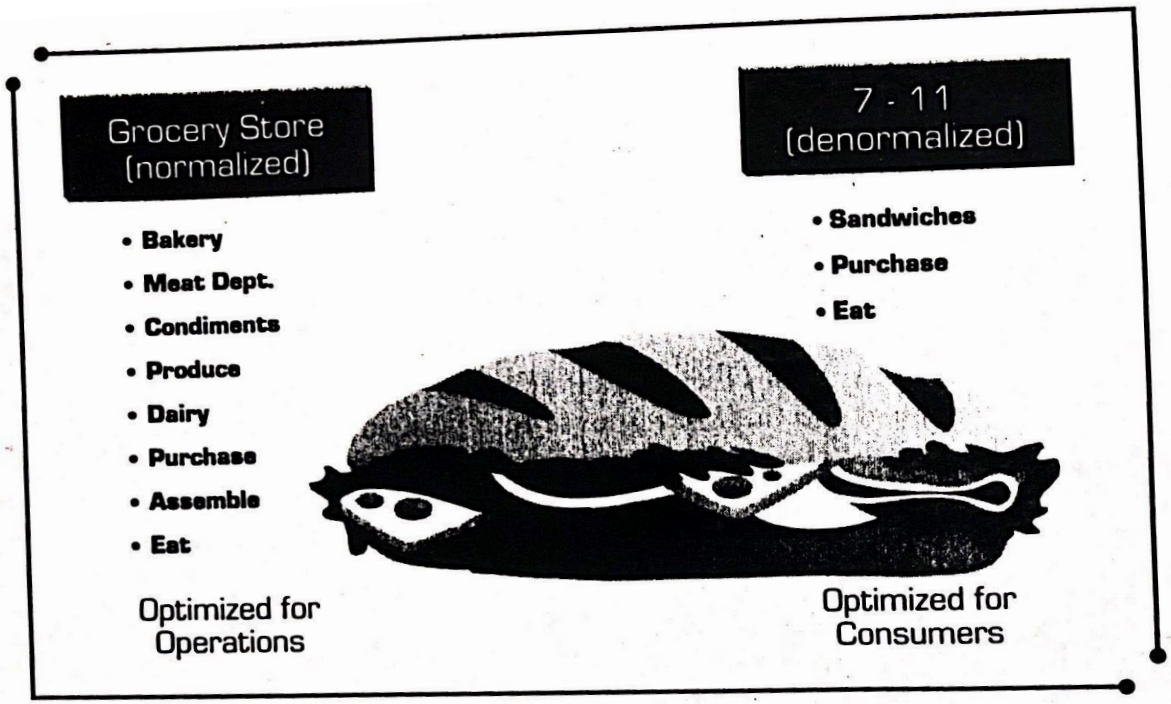


Figure 12.3 The data sandwich

A grocery store is optimized for operations. It is designed and built to make re-stocking the shelves as easy as possible. The onus is placed on the customer to know the locations of every item. A convenience store such as 7-11 is optimized for consumers. The onus is placed on the operational team to pre-build products that are desired by customers. To be successful and sustainable in the data mart business, you must build 7-11s.

[redacted] Doug Hackney [redacted]

Product	Market	Time	Units
Camera	Boston	Q1	1200
Camera	Boston	Q2	1500
Camera	Boston	Q3	1800
Camera	Boston	Q4	2100
Camera	Seattle	Q1	1000
Camera	Seattle	Q2	1100
.....	.....	.....	.....
Tuner	Denver	Q1	250
Tuner	Denver	Q2	300

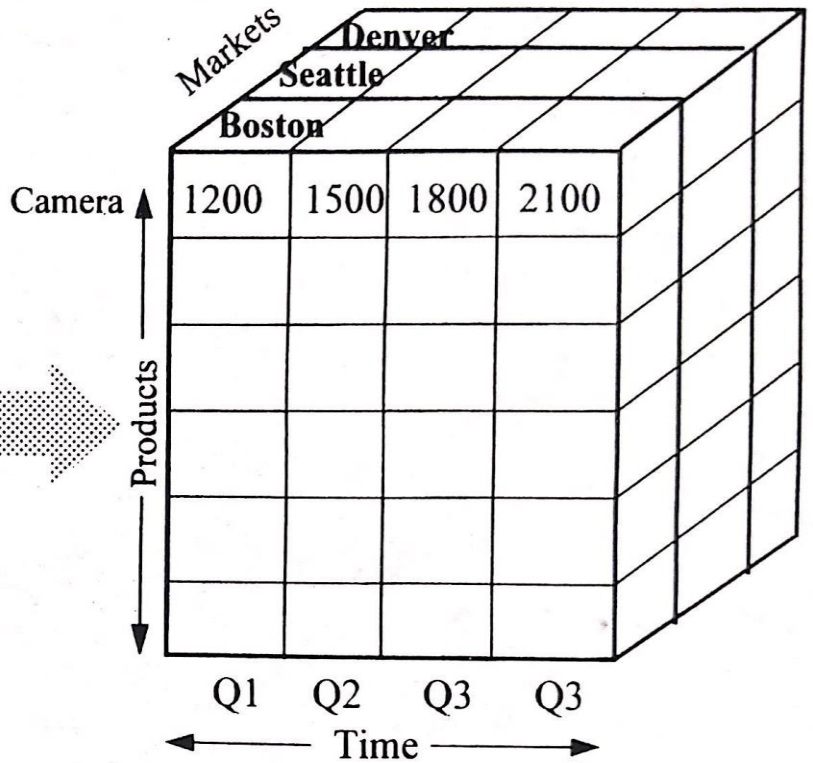


Figure 13.1 Relational tables and multidimensional cubes.

BRAND	DOLLAR SALES	UNIT SALES
Axon	780	263
Framis	1044	509
Widget	213	444
Zapper	95	39

1Q- 1995

2. Definition of a Data Warehouse

A Data Warehouse is a Subject Oriented, Integrated, Time Variant, Non-volatile collection of data that is used in the support of management's decision making tools.

- Subject Oriented

Information in the data warehouse is subect oriented. Buisiness information is classified based on those subjects that are of interest to the enterprise. For a manufacturer, these subjects may be customers, products, accounts and vendors. For a university, subjects may be students, classes, and instructors. For a hospital, subjects may be patients, medical staff, medications etc.

- Integrated

Data is defined once to meet the needs of the entire enterprise rather than a single application. The reports generated for different organizations will contain the same results.

submits

Architected

- Non-volatile

New data is always appended to the database, rather than replaced. The database continuosly absorbs new data, integrating it with the previous data.

- Time-variant

The data warehouse includes historical data to be used in identifying and evaluating trends.

We can characterize a data warehouse by contrasting how business data stored in a data warehouse differs from the operational data used by production applications.

Operational Database	*Data Warehouse*
Operational Data	Business Data for Information
Application Oriented	Subject Oriented
Current	Current Plus Historical
Details	Details Plus Summaries
Changing Continuously	Stable

Mistakes to Avoid in Defining Data Warehousing Architecture

- Do not develop "virtual" data warehouses
- Do not develop "stovepipe" data marts that are not integrated across business areas
- Do not implement enterprise data warehouse as a single, large, top-down development effort
- Do not populate data warehouse with "dirty" source data containing missing, inconsistent, and erroneous data values

Copyright © 1996 Peter R. Mineo

Specifications of User View

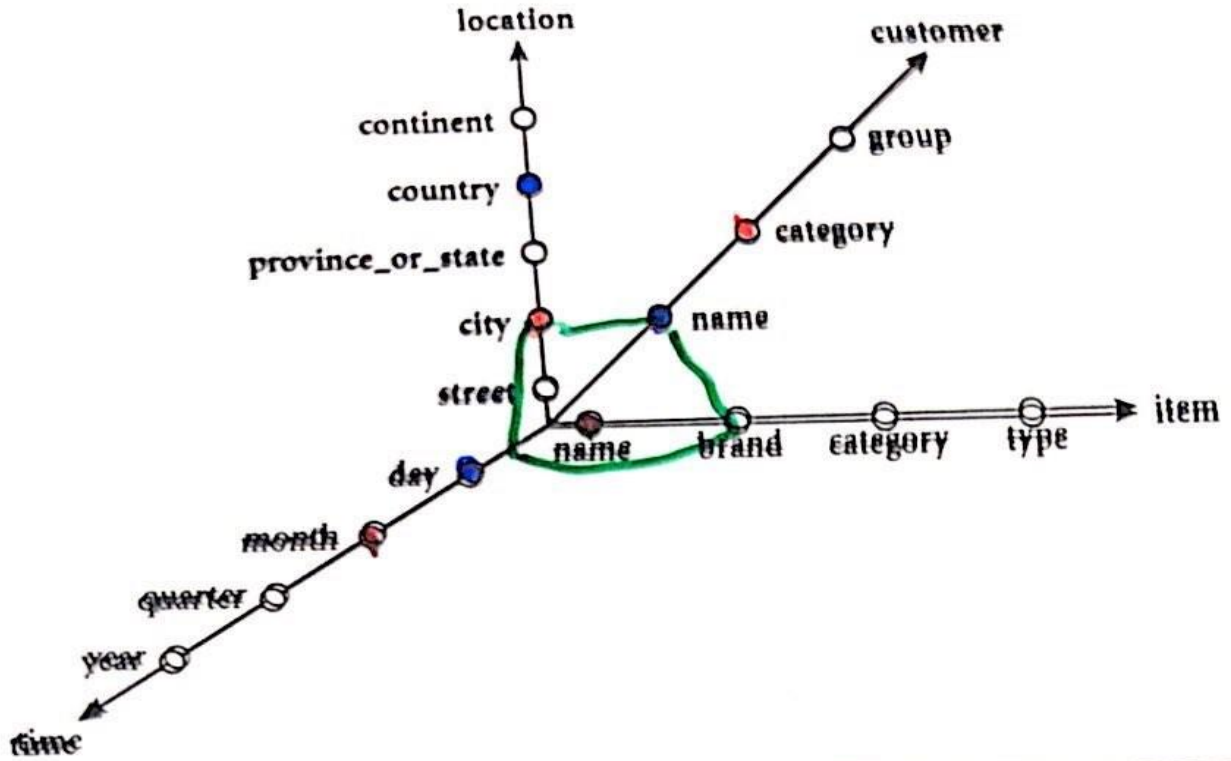
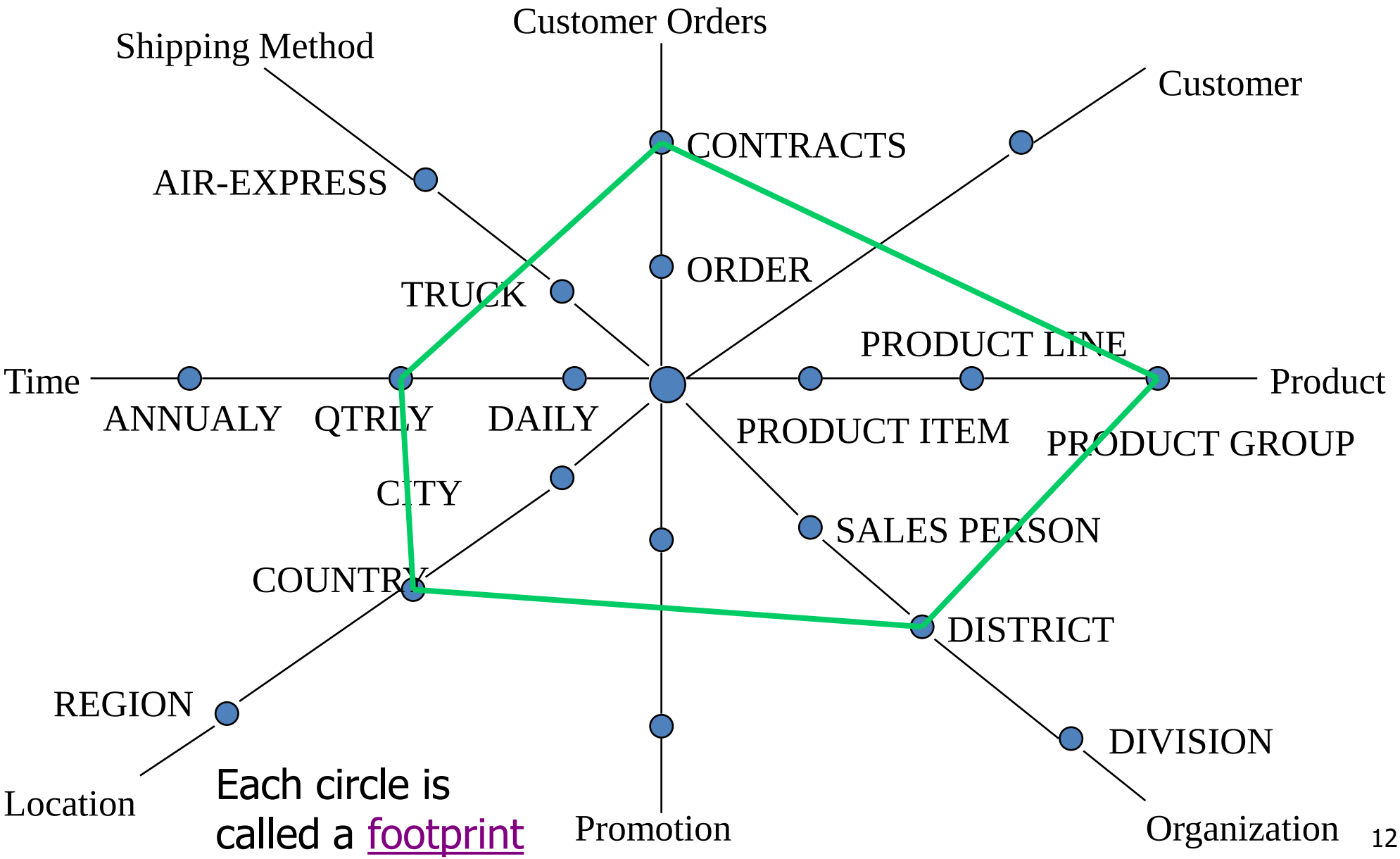


Figure 2.11 Modeling business queries: a starnet model.

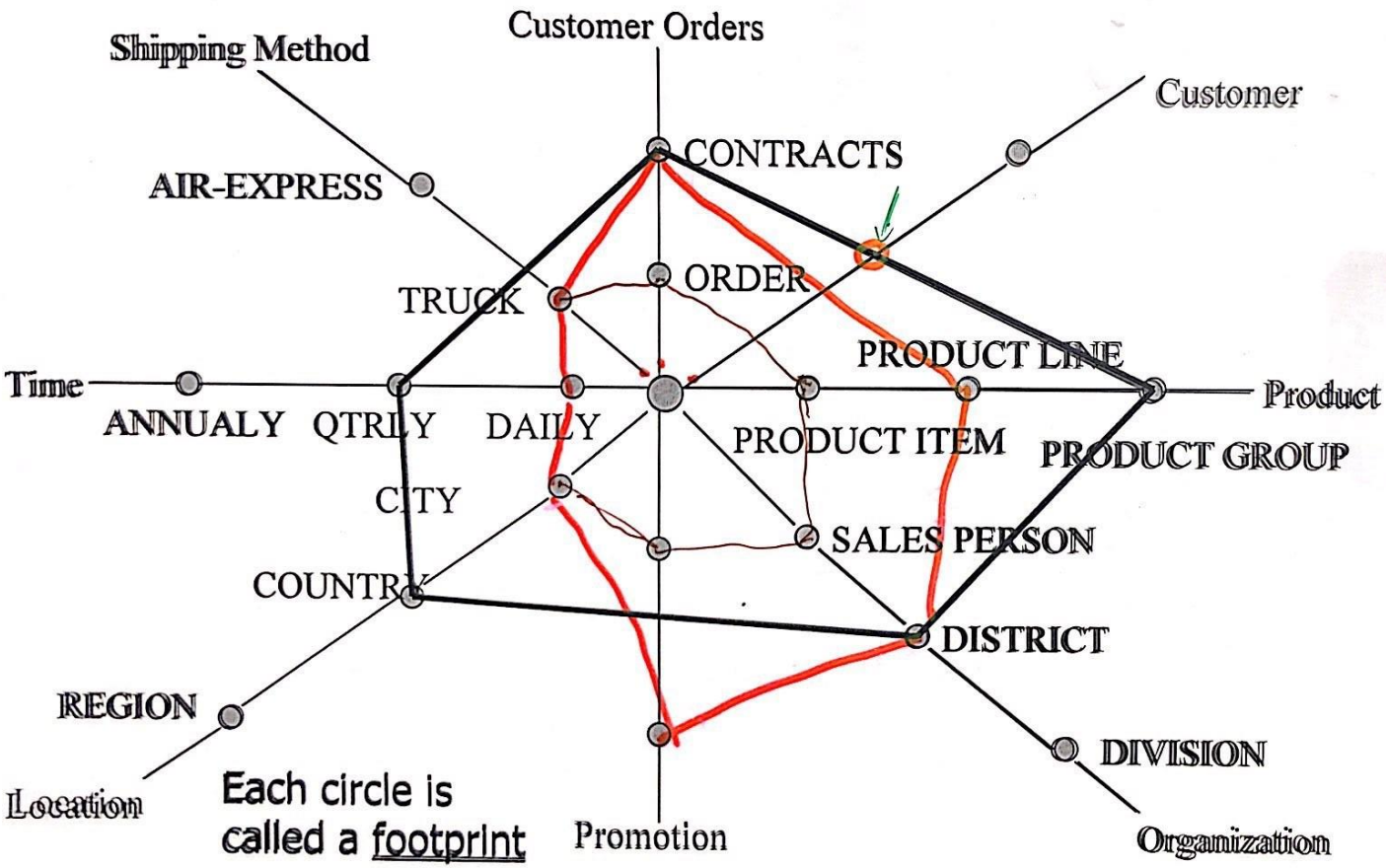
413

A Star-Net Query Model



Each circle is called a footprint

A Star-Net Query Model



Factors for Partial Cuboid Materialization

- Potential access frequency of generated cuboid,
- the size of the generated cuboid, and
- how materialization of one cuboid may benefit computation of other cuboids in the lattice.

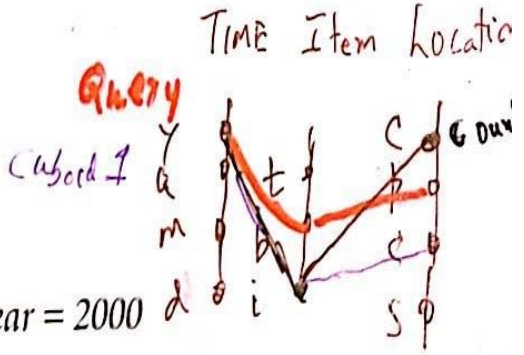
4.9 (p.163)

4.14

Example 2.15 Suppose that we define a data cube for *AllElectronics* of the form "*sales* [*time, item, location*]: **sum**(*sales_in_dollars*)". The dimension hierarchies used are "*day* < *month* < *quarter* < *year*" for *time*, "*item_name* < *brand* < *type*" for *item*, and "*street* < *city* < *province_or_state* < *country*" for *location*.

Suppose that the query to be processed is on {*brand, province_or_state*}, with the selection constant "*year* = 2000". Also, suppose that there are four materialized cuboids available, as follows:

- cuboid 1: {*item_name, city, year*}
- cuboid 2: {*brand, country, year*}
- cuboid 3: {*brand, province_or_state, year*}
- cuboid 4: {*item_name, province_or_state*} where *year* = 2000



Cuboid Indexing Specifications

Indexing OLAP Data. Bitmap Index

- Index on a particular column
- Each value in the column has a bit vector: bit-op is fast
- The length of the bit vector: # of records in the base table
- The *i*-th bit is set if the *i*-th row of the base table has the value for the indexed column
- not suitable for high cardinality domains

Base table

Cust	Region	Type
C1	Asia	Retail
C2	Europe	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europe	Dealer

Index on Region

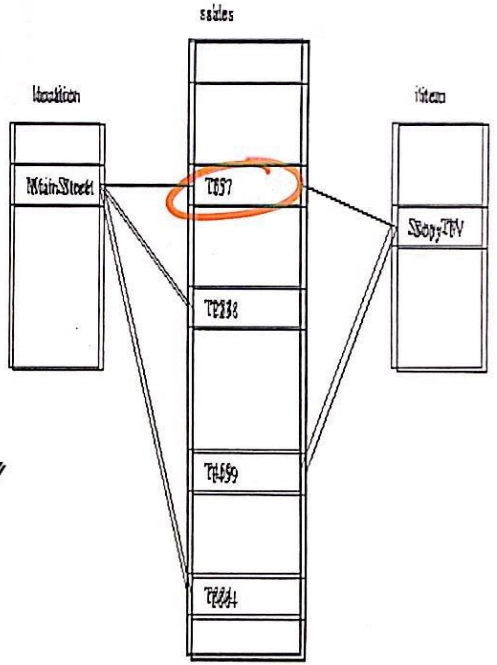
RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Index on Type

RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Indexing OLAP Data: Join Indices

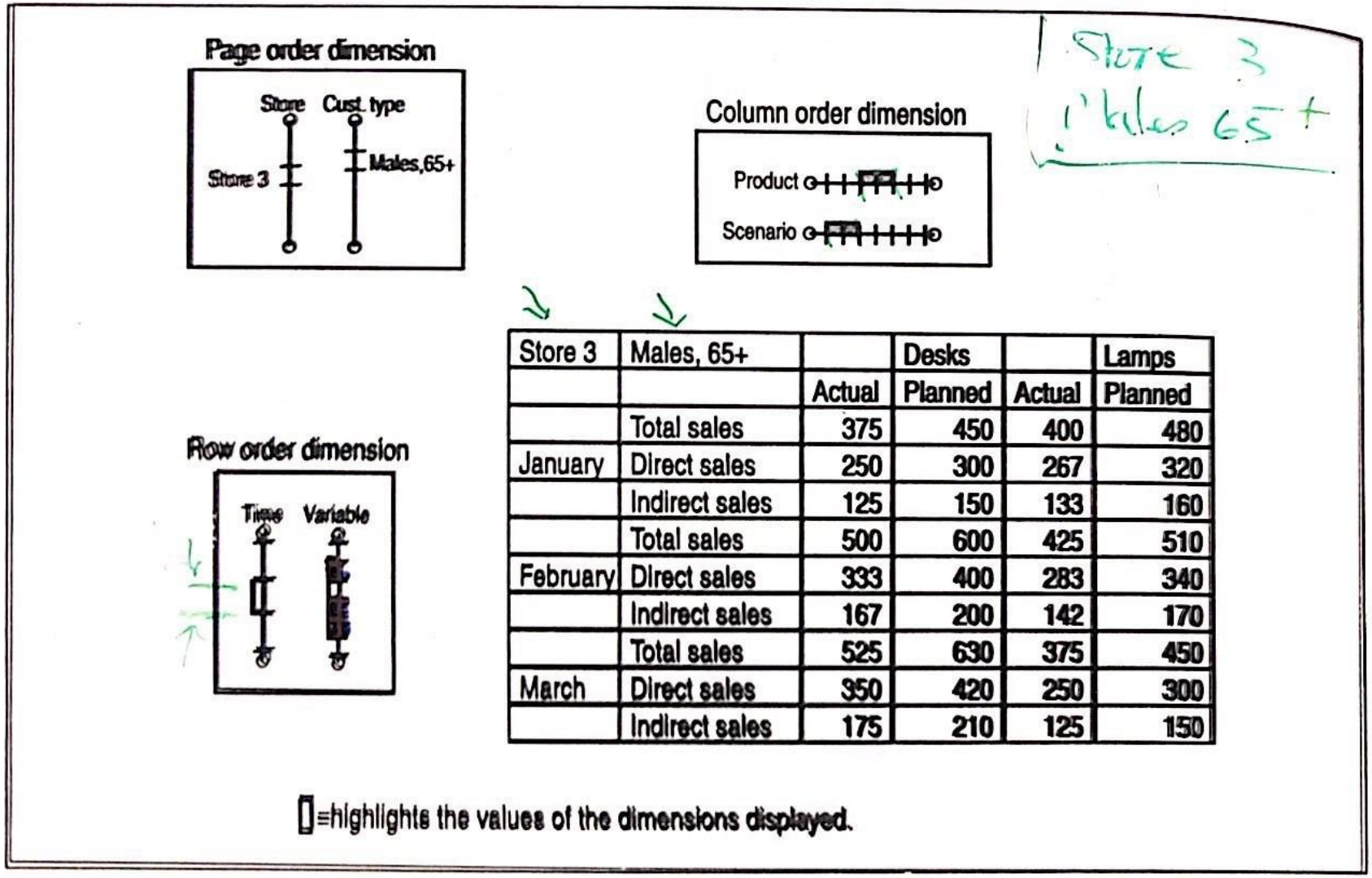
- Join index: JI(R-id, S-id) where R (R-id, …) ⋈ S (S-id, …)
- Traditional indices map the values to a list of record ids
  - It materializes relational join in JI file and speeds up relational join
- In data warehouses, join index relates the values of the dimensions of a star schema to rows in the fact table.
  - E.g. fact table: *Sales* and two dimensions *city* and *product*
    - A join index on *city* maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
  - Join indices can span multiple dimensions



Output Layout Specifications

The ability to easily change views of the same data by reconfiguring how dimensions are displayed is one of the great benefits of multidimensional systems. It is due to the separation of data structure, as represented in the MDS, from data display, as represented in the multidimensional grid. The actual method will be different from tool to tool, but the essence is the same. For example, the commands or actions to create Figure 3.21 are as follows:

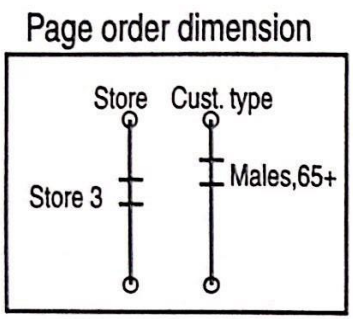
1. Show variables nested within months along the rows of the screen.
2. Show scenarios nested within products along the columns of the screen.
3. Show stores and customer type along the pages of the screen.

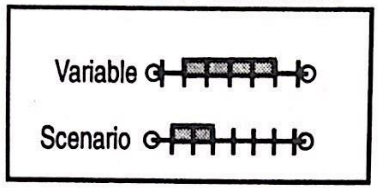


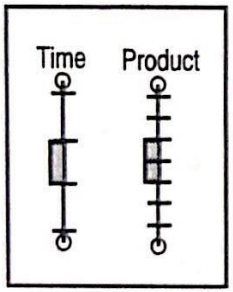
Store 3	Males, 65+		Desks		Lamps
		Actual	Planned	Actual	Planned
	Total sales	375	450	400	480
January	Direct sales	250	300	267	320
	Indirect sales	125	150	133	160
	Total sales	500	600	425	510
February	Direct sales	333	400	283	340
	Indirect sales	167	200	142	170
	Total sales	525	630	375	450
March	Direct sales	350	420	250	300
	Indirect sales	175	210	125	150

▯=highlights the values of the dimensions displayed.

Figure 3.21 A six-dimensional data display.



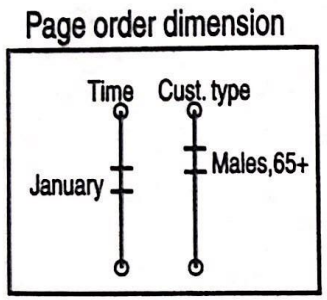


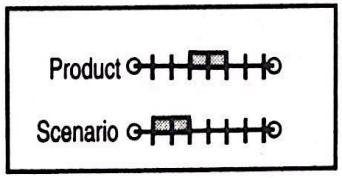


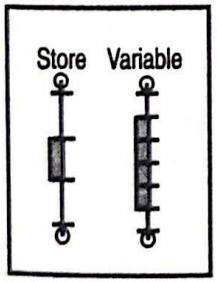
Store 3	Males,65+		Direct sales		Indirect sales		Total sales
		Actual	Planned	Actual	Planned	Actual	Planned
January	Desks	250	300	125	150	375	450
	Lamps	267	320	133	160	400	480
February	Desks	333	400	167	200	500	600
	Lamps	283	340	142	170	425	510
March	Desks	350	420	175	210	525	630
	Lamps	250	300	125	150	375	450

▯=highlights the values of the dimensions displayed.

Figure 3.22 A different six-dimensional data display of the same MDS.



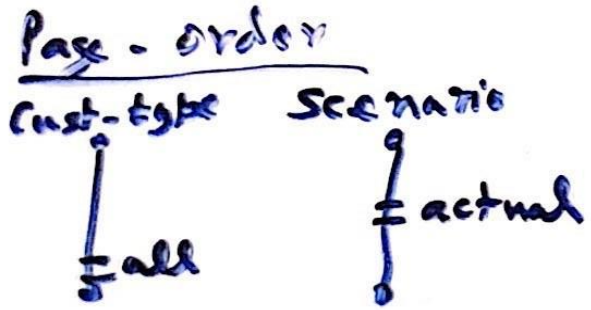


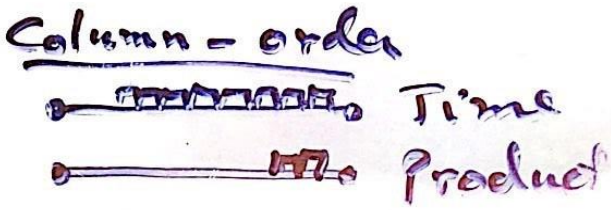


January	Males, 65+	Desks		Lamps	
		Actual	Planned	Actual	Planned
	Total sales	400	550	430	500
Store 1	Direct sales	250	315	242	290
	Indirect sales	180	200	150	180
	Total sales	500	600	425	500
Store 2	Direct sales	340	390	273	335
	Indirect sales	167	200	145	165
	Total sales	375	450	400	480
Store 3	Direct sales	250	300	267	320
	Indirect sales	125	150	133	160

▮=highlights the values of the dimensions displayed.

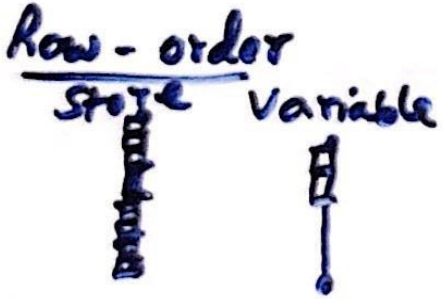
Figure 3.23 Yet a third six-dimensional data display of the same MDS.





		December		January		February	
		High Price	Low Price	High Price	Low Price	High Price	Low Price
Store 1	Sales (000)	450	200	340	280		
	Costs	267	160	265	200		
Store 2	Sales	402	400	350	450		
	Costs	283	315	310	325		
Store 3	Sales	250	460	200	500		
	Costs	225	300	165	356		

Figure 3.28 A two-dimensional comparison.



Measures-
Categorization & Computation

Data Cube Measures: Three Categories

Distributive: if the result derived by applying the function to *n* aggregate values is the same as that derived by applying the function on all the data without partitioning

E.g., count(), sum(), min(), max()

Algebraic: if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded integer), each of which is obtained by applying a distributive aggregate function

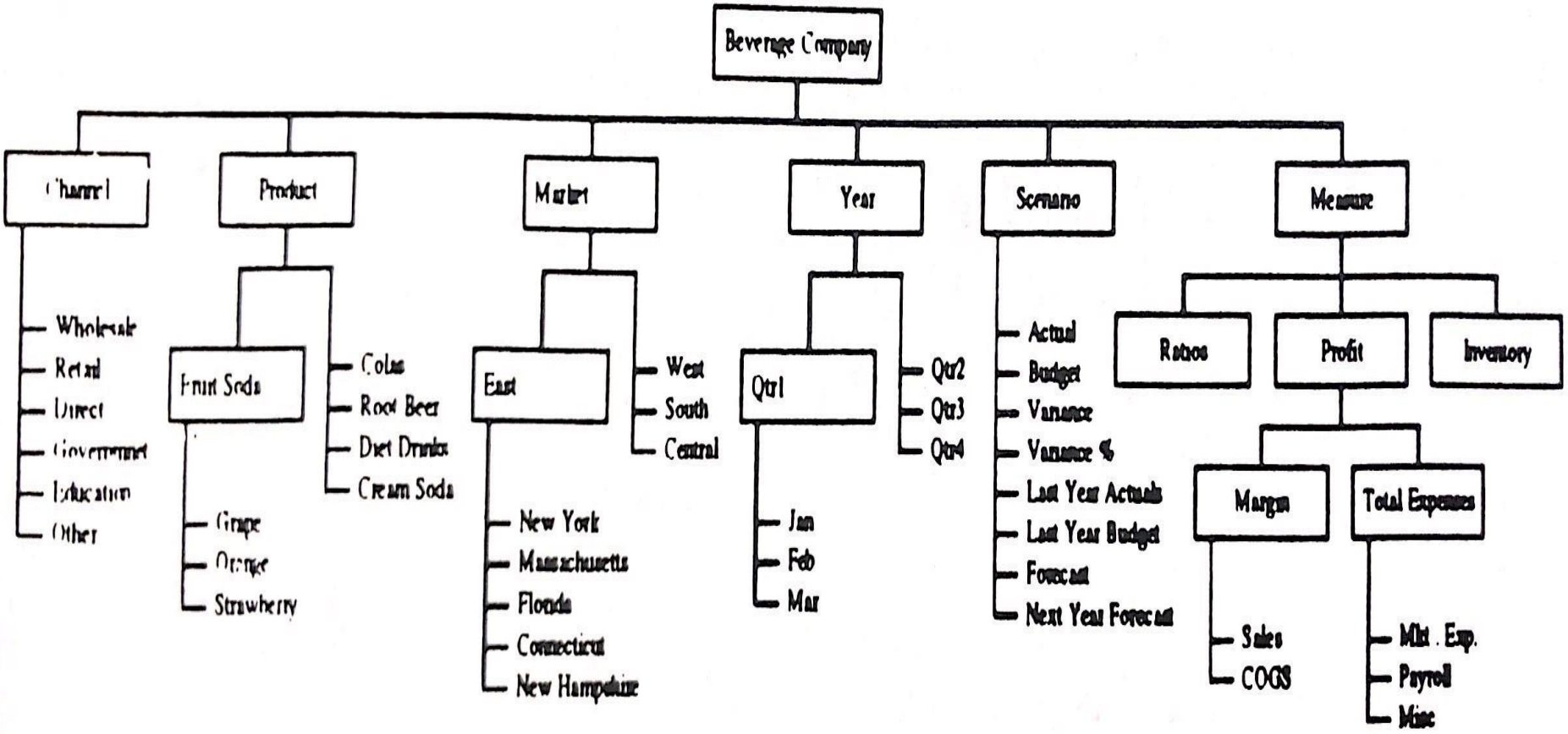
E.g., avg(), min_N(), standard_deviation()

Holistic: if there is no constant bound on the storage size needed to describe a subaggregate.

E.g., median(), mode(), rank()

Dimension Schema Variant & Output Generation- An Example

Multidimensional Model



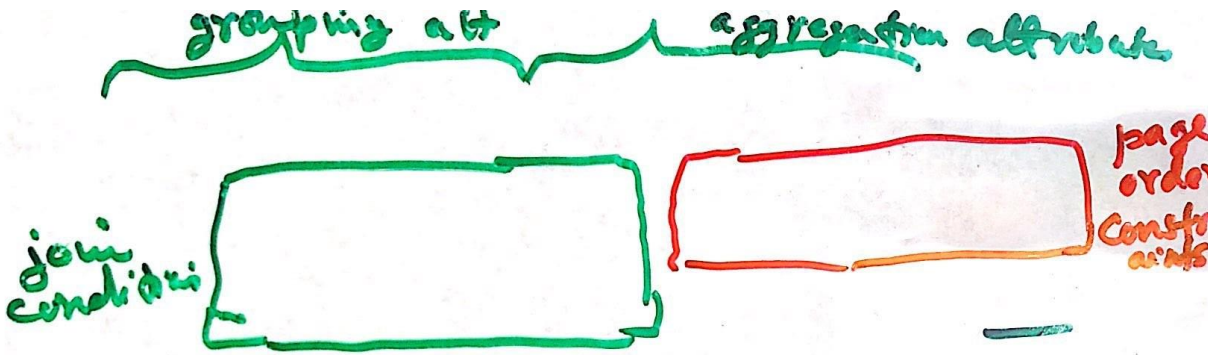
chan-id	Prod-id	Mkt-id	Time-id	Scen-id	Sales	COGS	Margin	Mkt. Exp.	Payroll	Misc	Tot. Exp.	Profit
1	2	1	1	1	$245	$100	$145	$23	$45	$12	$80	$65
3	2	2	1	1	$123	$65	$58	$12	$23	$12	$47	$11
2	105	1	1	1								
3	105	100	1	1								

Market Table

rollup	member	Mkt-id
East	New York	1
East	Florida	2
Market	East	100
Market	West	101
Market	South	102
Market	Central	103
Market	Market	500

Product Table

rollup	member	Prod-id
Fruit Soda	Grape	1
Fruit Soda	Orange	3
Colas	cola	2
Product	Fruit Soda	101
Product	Root Beer	102
Product	Cream Soda	103
Product	Diet drinks	104
Product	Colas	105
Product	Product	500



Scenario	Time	Scen	...	..
Actual	Qtr1			
Actual	Qtr2			
Actual	Qtr3			
Actual	Qtr4			
Budget	Qtr1			
Budget	Qtr2			
Budget	Qtr3			
Budget	Qtr4			

Computing cuboids, from Base Cuboid-
"No Materialization"

```
SELECT Scenario.member, Time.member, Sales, COGS, Margin,...., Profit
        FROM Fact, Channel, Product, Market, Time, Scenario
       WHERE Fact.Chan-id = Channel.Chan-id and Channel.member ='Channel'
             Fact.Prod-id = Product.Prod-id and Product.member = 'Product'
             and Fact.Mkt-id = Market.Mkt-id and Market.member = 'Market'
             and Fact.Time-id = Time.Time-id and Time.rollup = 'Year'
             and Fact.Scen-id = Scenario.Scen-id and Scenario.member = 'Actual'
UNION  SELECT Scenario.member, Time.member, Sales, COGS, Margin,...., Profit
        FROM Fact, Channel, Product, Market, Time, Scenario
       WHERE Fact.Chan-id = Channel.Chan-id and Channel.member ='Channel'
             Fact.Prod-id = Product.Prod-id and Product.member = 'Product'
             and Fact.Mkt-id = Market.Mkt-id and Market.member = 'Market'
             and Fact.Time-id = Time.Time-id and Time.rollup = 'Year'
             and Fact.Scen-id = Scenario.Scen-id and Scenario.member = 'Budget'
```

Computing Base (or Aggregate) Cuboid from Source Tables & Output Generation- An Example

Example 3.4 Star schema definition. The star schema of Example 3.1 and Figure 3.4 is defined in DMQL as follows:

```
define cube sales_star [time, item, branch, location]:
        dollars_sold = sum(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)
```

The define cube statement defines a data cube called *sales_star*, which corresponds to the central *sales* fact table of Example 3.1. This command specifies the dimensions and the two measures, *dollars_sold* and *units_sold*. The data cube has four dimensions, namely, *time*, *item*, *branch*, and *location*. A define dimension statement is used to define each of the dimensions. ■

Computing cuboids, from Source Tables- "No Materialization"

Example 3.7 Interpreting measures for data cubes. Many measures of a data cube can be computed by relational aggregation operations. In Figure 3.4, we saw a star schema for *AllElectronics* sales that contains two measures, namely, *dollars_sold* and *units_sold*. In Example 3.4, the *sales_star* data cube corresponding to the schema was defined using DMQL commands. "*But how are these commands interpreted in order to generate the specified data cube?*"

Suppose that the relational database schema of *AllElectronics* is the following:

time(time_key, day, day_of_week, month, quarter, year)
item(item_key, item_name, brand, type, supplier_type)
branch(branch_key, branch_name, branch_type)
location(location_key, street, city, province_or_state, country)
sales(time_key, item_key, branch_key, location_key, number_of_units_sold, price)

The DMQL specification of Example 3.4 is translated into the following SQL query, which generates the required *sales_star* cube. Here, the sum aggregate function, is used to compute both *dollars_sold* and *units_sold*:

```
select s.time_key, s.item_key, s.branch_key, s.location_key,
        sum(s.number_of_units_sold * s.price), sum(s.number_of_units_sold)
from time t, item i, branch b, location l, sales s,
where s.time_key = t.time_key and s.item_key = i.item_key
        and s.branch_key = b.branch_key and s.location_key = l.location_key
```

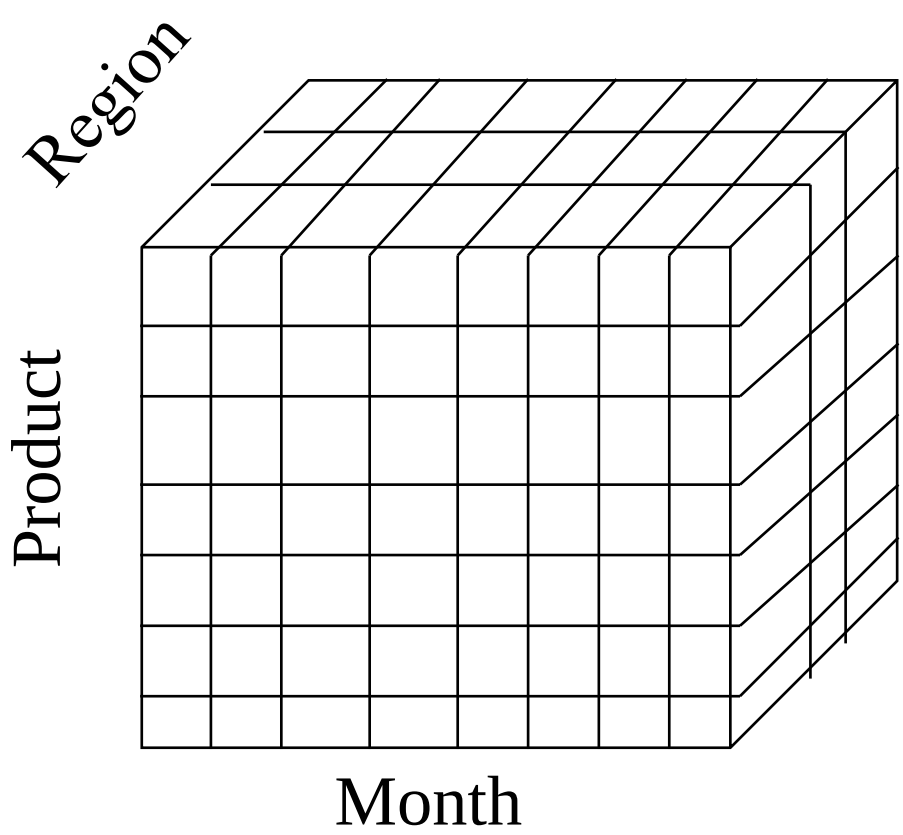
GroupBy: s.time_key, s.item_key, s.branch_key, s.location_key

The cube created in the above query is the base cuboid of the *sales_star* data cube. It contains all of the dimensions specified in the data cube definition, where the granularity of each dimension is at the join key level. A join key is a key that links a fact table and a dimension table. The fact table associated with a base cuboid is sometimes referred to as the base fact table.

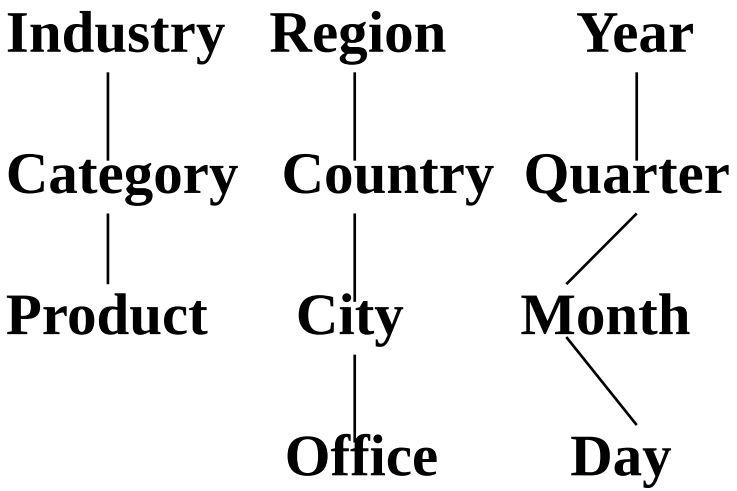
By changing the group by clauses, we can generate other cuboids for the *sales_star* data cube. For example, instead of grouping by *s.time_key*, we can group by *t.month*, which will sum up the measures of each group by month. Also, removing "group by *s.branch_key*" will generate a higher-level cuboid (where sales are summed for all branches, rather than broken down per branch). Suppose we modify the above SQL query by removing *all* of the group by clauses. This will result in obtaining the total sum of *dollars_sold* and the total count of *units_sold* for the given data. This zero-dimensional cuboid is the apex cuboid of the *sales_star* data cube. In addition, other cuboids can be generated by applying selection and/or projection operations on the base cuboid, resulting in a lattice of cuboids as described in Section 3.2.1. Each cuboid corresponds to a different degree of summarization of the given data. ■

Multidimensional Data- Short Quiz

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



Dimensions: *Product, Location, Time*
Hierarchical summarization paths



Questions

1. Draw A Starnet Query Model (SQM) diagram showing dimensional hierarchies on right.
2. Draw polygon that shows the view corresponding to the cuboid on the left (View a user wants to query).
3. Show a cuboid on the SQM diagram (by drawing another polygon) that dominates the cuboid in 2.
4. Show a cuboid that can't be used (cuboid that is **not feasible**, since it is coarser) to compute cuboid in 2.
5. Compute total number of cuboids in the data cube.