OLAP/Data Warehouses

Yannis Kotidis

What is a Database?

- **From Wikipedia:**
  - A **database** is a structured collection of records or data. A computer database relies upon software to organize the storage of data. The software models the database structure in what are known as database models. The model in most common use today is the relational model. Other models such as the hierarchical model and the network model use a more explicit representation of relationships …
  - **Database management systems** (DBMS) are the software used to organize and maintain the database. These are categorized according to the database model that they support. The model tends to determine the query languages that are available to access the database. A great deal of the internal engineering of a DBMS, however, is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. ...

Note

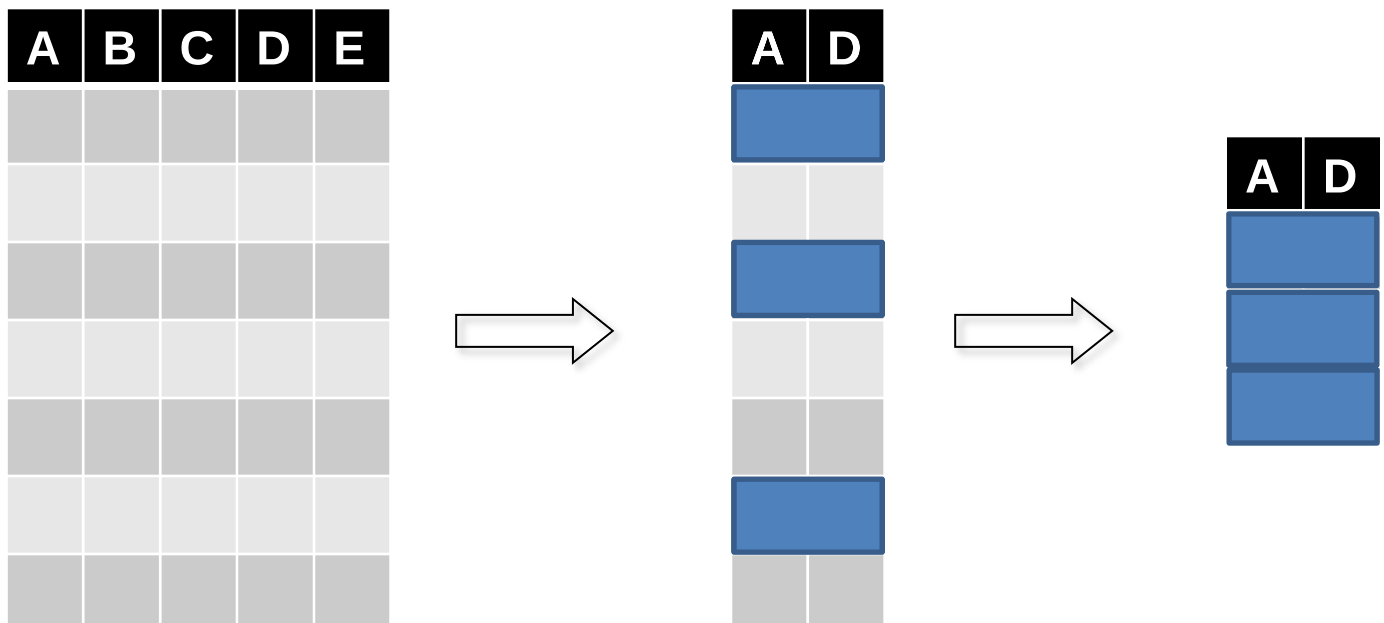
- Term "**database**" often used interchangeably for both the **data** and the **system** that manages it

Basic Database Usage (1): Querying

Relations

Statements
(select columns and rows)

Results



Basic Database Usage (2): Updates

- Banking transaction: transfer 100 euro from account A to account B
  - What can go wrong?

Account	Balance	
A	275	-100
B	64	+100

Issue 1: Partial results

- System failure prior to adding funds to account B (but after deleting them from A)

Account	Balance	
A	175	-100 ☑
B	64	+100 SYSTEM FAILURE

Issue 2: No isolation

- For an observer that monitors all funds money seem to temporality disappear (and reappear again)

Account	Balance		
A	175	-100	total funds are reduced by 100
B	64	+100	

Issue 3: lost update

- Two concurrent transactions on account A
  - T1: remove 100
  - T2: remove 50

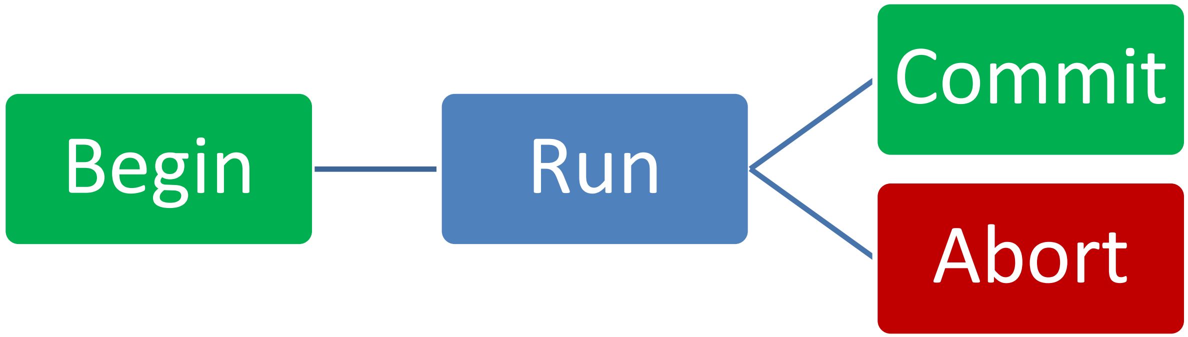
T2
Read balance (275)
Subtract 50
Write balance 225

Account	Balance
A	275
B	64

T1
Read balance (275)
Subtract 100
Write balance 175

Programming abstraction: Transactions

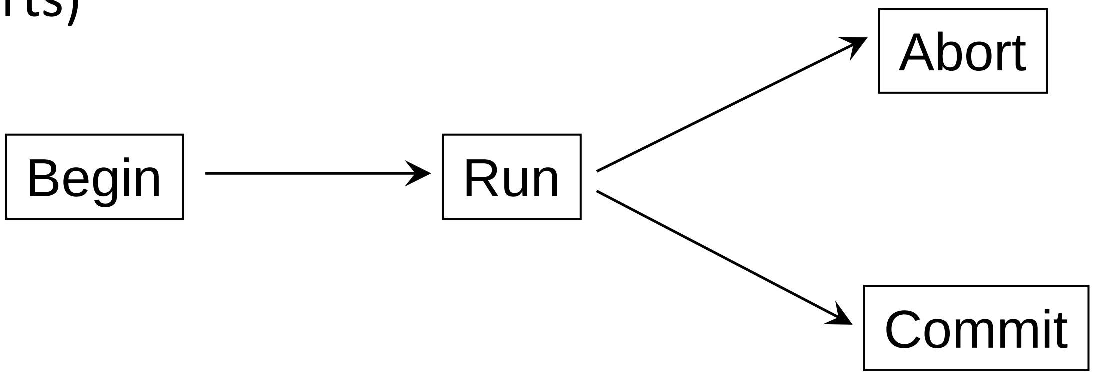
- Implement real-world transactions



- DBMSs guarantee ACID properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity (A.C.I.D.)

- The "all or nothing" property.
  - Programmer needn't worry about partial states persisting.
  - Two possible outcomes: transaction commits or rollbacks (aborts)



- Examples:
  - T1: Delete person from consultants table, insert person into employees table
  - T2: Transfer funds from account A to account B

Consistency (A.C.I.D)

- The database should start out "consistent“ (legal state), and at the end of transaction remain "consistent".
- The definition of "consistent" is up to the database administrator to define to the system
  - integrity constraints
  - other notions of consistency must be handled by the application.

Integrity or correctness of data

- Would like data to be “accurate” or “correct” at all times

EMP:

Name	Age
John	52
Jim	24
Martha	1

```
CREATE TABLE EMP (
    Name varchar(255) NOT NULL,
    Age int,
    CHECK (Age>=18)
);
```

Integrity/consistency constraints

- Predicates data must satisfy
- Examples:
  - age >= 18 and age < 65
  - x is key of relation R
  - $x \rightarrow y$ holds in R
  - Domain(x) = {Red, Blue, Green}
  - no employee should make more than twice the average salary

Isolation (A.C.I.D)

- Each transaction must appear to be executed as if no other transaction is executing at the same time.
- Transfer funds from A to B (T1).
- Another teller makes a query on A and B (T2).
- T2 could see funds on A or B but not in both!
  - Result may be independent of the time transactions were submitted

Durability (A.C.I.D.)

- Once committed, the transactions effects should not disappear.
  - Of course, they may be overwritten by subsequent committed transactions.

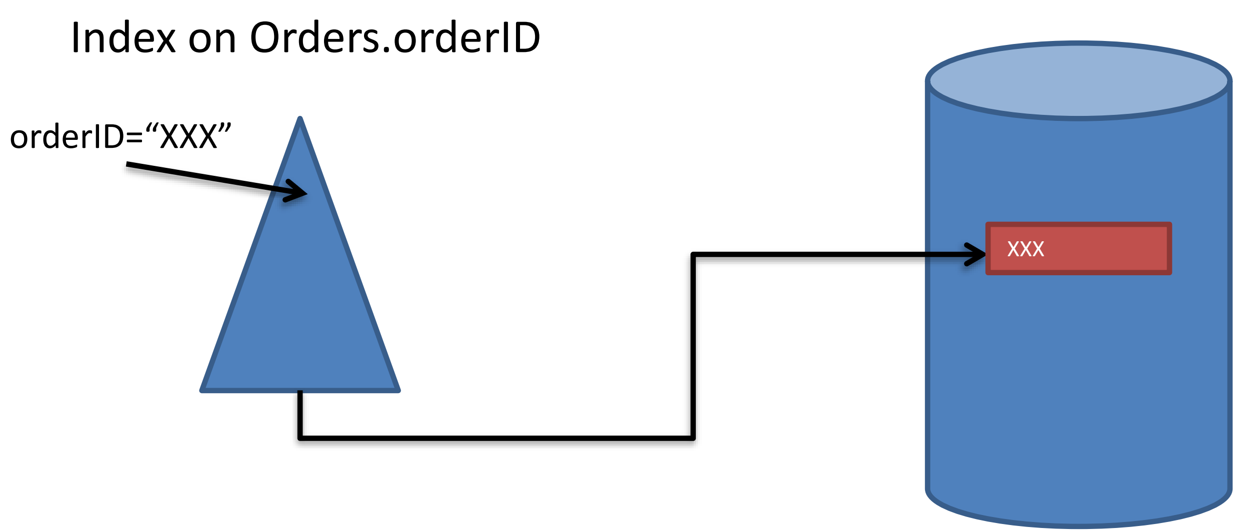
Implementation

- A, C, and D are mostly guaranteed by recovery (usually implemented via logging).
- I is mostly guaranteed by concurrency control (usually implemented via locking).
- Of course, life is not so simple. For example, recovery typically requires concurrency control and depends on certain behavior by the buffer manager…

Operational DBs: OLTP systems

- OLTP= On-Line **Transaction** Processing
  - order update: *pull up order# XXX and update status flag to "completed"*

update Orders set status="Completed"
where orderID="XXX"



Reconstruction of logical records

Employees

EmpID	Ename
101	John Smith
102	Nick Long
103	Susan Goal
104	John English
105	Alice Web
106	Patricia Kane

Projects

ProjID	Pname
2	Web_TV
3	Web_portal
4	Billing

Assignments

EmpID	ProjID	Hours
101	3	16
102	2	24
102	3	8
104	4	32
105	4	24
106	4	24

- List projects & hours assigned to employee Nick Long

```
Select Pname,Hours
From Employees E, Projects P, Assignments A
Where E.Ename = “Nick Long”
And E.EmpID=A.EmpID
And A.ProjID=P.ProjID
```

Physical Plan (step a): IndexSeek

Employees

EmpID	Ename
101	John Smith
102	Nick Long
103	Susan Goal
104	John English
105	Alice Web
106	Patricia Kane

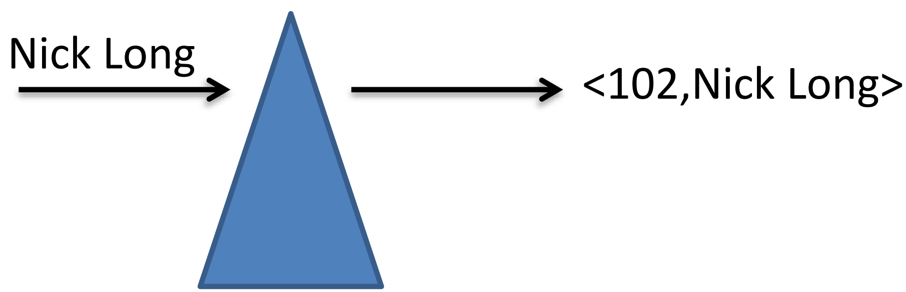
Projects

ProjID	Pname
2	Web_TV
3	Web_portal
4	Billing

Assignments

EmpID	ProjID	Hours
101	3	16
102	2	24
102	3	8
104	4	32
105	4	24
106	4	24

$\sigma_{\text{E.name=“Nick Long”}}(\text{Employees})$



Index on Employees.Ename

Physical Plan (step b): INLJ(Employees,Assignments)

Employees

EmpID	Ename
101	John Smith
102	Nick Long
103	Susan Goal
104	John English
105	Alice Web
106	Patricia Kane

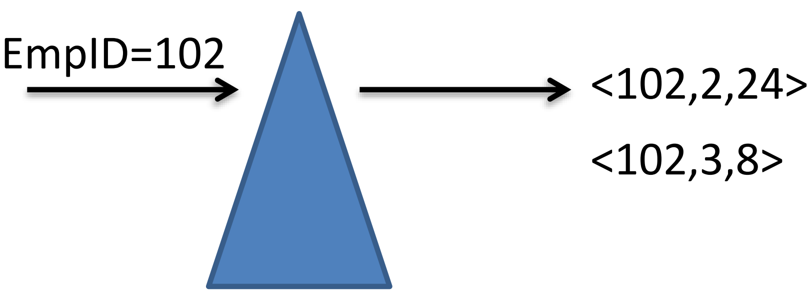
Projects

ProjID	Pname
2	Web_TV
3	Web_portal
4	Billing

Assignments

EmpID	ProjID	Hours
101	3	16
102	2	24
102	3	8
104	4	32
105	4	24
106	4	24

Employees $\bowtie$ Assignments



Index on Assignments.EmpID

Physical Plan (step c): INLJ(Assignments,Projects)

Employees

EmpID	Ename
101	John Smith
102	Nick Long
103	Susan Goal
104	John English
105	Alice Web
106	Patricia Kane

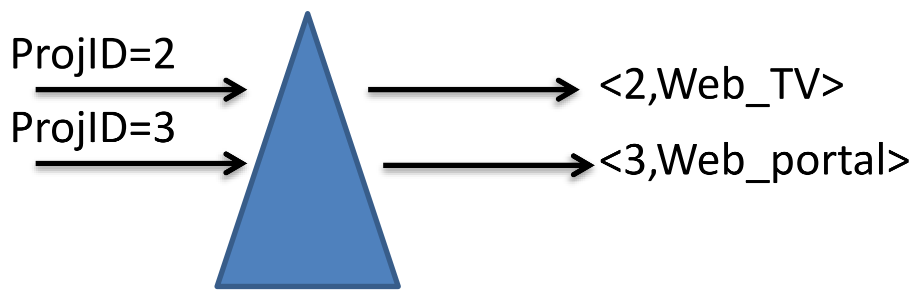
Projects

ProjID	Pname
2	Web_TV
3	Web_portal
4	Billing

Assignments

EmpID	ProjID	Hours
101	3	16
102	2	24
102	3	8
104	4	32
105	4	24
106	4	24

Assignments $\bowtie$ Projects



Index on Projects.ProjID (primary key)

On-Line **Transaction** Processing

- Examples
  - order update: *pull up order# XXX and update status flag to "completed"*
  - banking: *transfer 100 euros from account #A to account #B*
- *Transactions:*
  - Implement structured, repetitive clerical data processing tasks
  - Require detailed, up-to-date data
  - Are (most of the times) short-lived
    - read and/or update a few records
- **Integrity of the database is critical**
  - **DBMS should manage hundreds or thousands of concurrent transactions**
- Systems supporting this kind of activity are called *transactional systems*
  - Most traditional database management systems

Transactional Systems

- Transactional systems are optimized primarily for the **here** and **now**
- Can support many **simultaneous** users
  - concurrent read/write access
- Transactional systems don't necessarily record all previous data states
  - E.g. customer updates its address (moves to new town)
- Lots of data gets thrown away or archived
  - Old orders are deleted/archived to reduce size

Analytical queries on a production system?

- CEO wants to report total sales **per store** in Athens, for stores with at least 500 sales
- 3 tables: **Sales(custid, productid,storeid,amt)**

  Stores(storeid, manager,addressid)

  Addresses(addressid,number,street,city)

SELECT Stores.storeid, SUM(amt) as totalSales Aggregation

FROM Sales, Stores, Addresses

WHERE Stores.storeid = Sales.storeid Joins

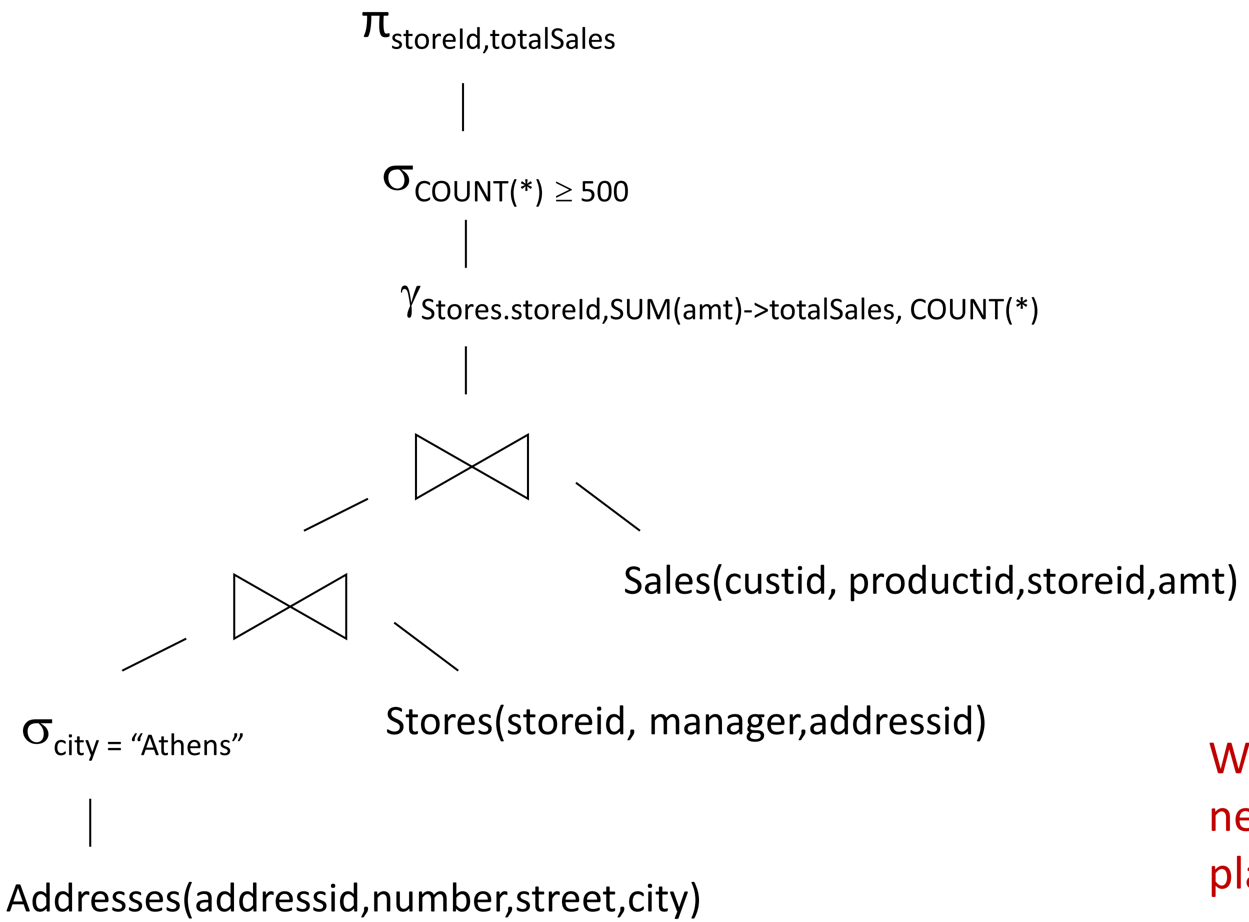
AND Stores.addressid=Addresses.addresid

AND Addresses.city="Athens"

GROUP BY Stores.storeid Group by

HAVING count(*) ≥ 500 Filter/Aggregation

Logical Plan



What happens if new sales take place while this query executes?

Sad realization

- Analytical queries on an operational database often take for ever
  - Schema favors small atomic actions
    - Excessive normalization results in costly joins
  - Need to scan LOTS of records
    - Indexes are not very useful when queries are not selective
  - Interference with daily transactions
    - Overhead of OLTP engine (logging, locking)

My employees & their projects

EmpID	Ename	ProjID	Pname	City	Hours
101	John Smith	3	Web_portal	Thessaloniki	16
102	Nick Long	2	Web_TV	Athens	24
103	Susan Goal	3	Web_portal	Thessaloniki	8
104	John English	4	Billing	Athens	32
105	Alice Web	4	Billing	Athens	24
106	Patricia Kane	4	Billing	Athens	24

- Schema is bad for OLTP (1NF)
  - Update anomalies, repetition of values
- But is all we need for reporting our employees and their projects!

OLAP:
<u>O</u>N<u>L</u>INE <u>A</u>NALYTICAL <u>P</u>ROCESSING

OLAP

- OLAP = online analytical processing
- OLAP is the process of creating and summarizing historical, multidimensional data
  - To help organizations understand their data better
  - Provide a basis for informed decisions (Decision Support Systems, Business Intelligence)
  - Allow users to manipulate and explore data easily and intuitively

Data Analytics Stack

OLAP

- Well defined computations over data categorized by multiple dimensions of interest
  - Enables users to easily and selectively extract and query data in order to analyse it from different points of view

Data Mining

- Seek to find relationships and patterns in data
  - Frequent itemset
  - Association rules
  - Clustering

Machine Learning

- Build models for prediction, classification etc.
  - Image classification
  - Speech processing
  - Sentiment analysis
  - NLP

OLAP Examples

OLAP

- Well defined computations over data categorized by multiple **dimensions** of interest
  - Enables users to easily and selectively extract and query data in order to analyse it from different points of view

A. Group sales data across different **dimensions**: Product, Customer, Location (point of sale) and Time
   - Dimensions identify what, who, where & when

B. Compute interesting stats on selected measures

Examples:

1. "Average January sales (€) for all stores in Attika"
2. "Number of shoes over 100€ sold to female customers between ages 18 and 25"
3. "Top-10 product-categories whose sales (%) increased the most over the past year"

Can you identify the dimensions in these queries???

1^{st} query in more details

OLAP

- Well defined computations over data categorized by multiple **dimensions** of interest
  - Enables users to easily and selectively extract and query data in order to analyse it from different points of view

"Average January sales (€) for all stores in Attika"

1^{st} dimension denotes when

2^{nd} dimension denotes where

A common aggregate function: AVG() over the available measure (sales)

Other examples: Max(), Min(), Count(), StDev(), Median()

OLAP vs. OLTP

	OLTP	OLAP
User	Clerk, IT professional	Knowledge worker
Function	Day to day operations	Decision support
DB design	Application-oriented (E-R based)	Subject-oriented (Star, snowflake)
Data	Current, Isolated	Historical, Consolidated
View	Detailed, Flat relational	Summarized, Multidimensional
Usage	Structured, Repetitive	Ad hoc
Unit of work	Short, simple transaction	Complex query
Access	Read/write	Read mostly
Operations	Index/hash on prim. key	Lots of scans
# Records accessed	Tens	Millions
# Users	Thousands	Hundreds
Db size	100 MB - GB	100 GB - TB
Metric	Trans. throughput	Query throughput, response

DATA WAREHOUSES

The Data Warehouse

- In order to support OLAP, data is collected from multiple data sources, cleansed and organized in data warehouses

- The data warehouse is a huge repository of enterprise data that will be used for decision making

- After data is loaded in the data warehouse, OLAP cubes are often pre-summarized across dimensions of interest to drastically improve query time

Data Warehouse definition

- A decision support database that is maintained separately from the organization's operational databases.
- A data warehouse is a
  - **subject-oriented,**
  - **integrated,**
  - **time-varying,**
  - **non-volatile**

  collection of data that is used primarily in organizational decision making.

-- *W.H. Inmon, Building the Data Warehouse, 1992.*

Subject-Oriented

- Organized around **major subjects**, such as customer, product, sales
- Focusing on the **modeling** and **analysis** of data for decision makers, not on daily operations or transaction processing
- Provide a **simple** and **concise view** around particular subject issues by excluding data that are not useful in the decision support process

Integrated

- Constructed by **integrating** multiple, **heterogeneous** data sources
  - relational databases, files, external sources
- Data cleaning and data integration techniques are applied
  - Ensure consistency in naming conventions, keys, attribute measures, etc. among different data sources
    - E.g., Hotel price: currency, tax, breakfast covered, etc.
  - When data is moved to the warehouse, it is **transformed**

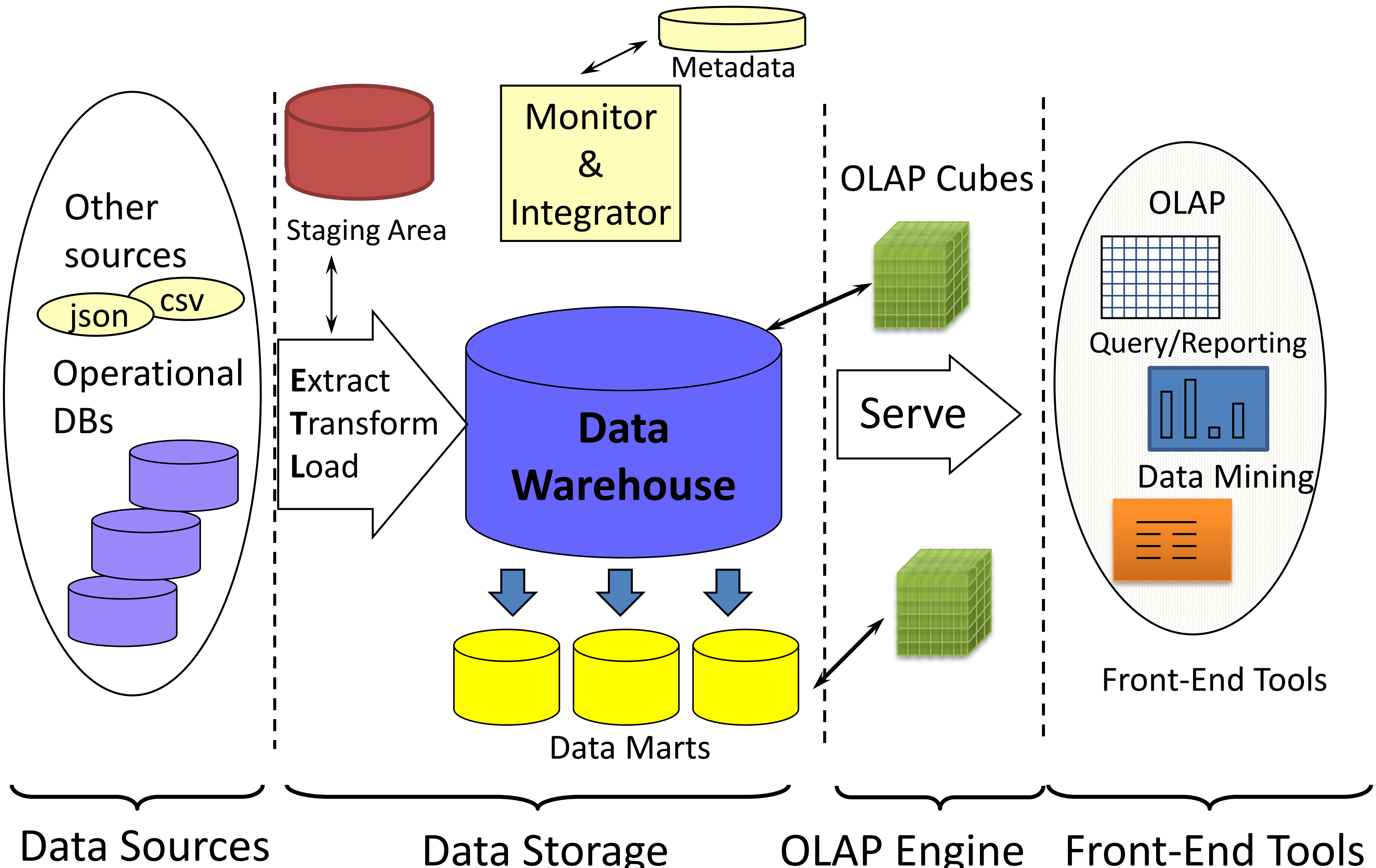
Time-Variant

- The **time horizon** for the data warehouse is significantly longer than that of operational systems
  - Operational database: current data, old values overwritten, deleted or archived
  - Data warehouse: provides data from a historical perspective (e.g., past 5-10 years) for trend analysis

Non-volatile

- A **physically separate** store of data transformed from the operational environment
- Operational update of data does not occur in the data warehouse environment
  - Does not require transaction processing, recovery, and concurrency control mechanisms
  - Requires only **two operations** in data accessing:
    - **loading** of data and **access** to data

Data Warehouse Architecture



Implementation

- **Warehouse database server**
  - Almost always a relational DBMS.
- **OLAP Servers (for computing OLAP Cubes)**
  - ***Relational OLAP (ROLAP):*** extended relational DBMS that maps operations on multidimensional data to standard relational operations.
  - ***Multidimensional OLAP (MOLAP):*** special purpose server that directly implements multidimensional data and operations.
- **Clients**
  - Query and reporting tools.
  - Analysis tools.
  - Data mining tools.

Data Marts

- Smaller warehouses
- Span part of organization
  - e.g., marketing (customers, products, sales)
- Do not require enterprise-wide consensus
  - But may lead to long term integration problems

Basic Query Pattern

- Analyst projects data into a selected subset of dimensions and computes interesting statistics
- In SQL this is expressed by grouping records using the selected attributes and computing aggregate functions (e.g. sum(), average(), count(), max()) over each group
  - "Group by followed by aggregation"
  - Additional filtering may be used to restrict the scope of the query

Example

- “Compute the total revenue (=sum) the minimum and maximum price for each combination of customer and store”

  - Sales Data:

Time	Customer	Store	Product	*Price*
T1	C1	S2	P1	*$90*
T2	C2	S1	P2	*$70*
T3	C1	S1	P2	*$45*
T4	C3	S1	P1	*$40*
T5	C1	S2	P2	*$25*
T6	C1	S2	P2	*$50*
T7	C2	S1	P4	*$45*
T8	C3	S1	P1	*$10*

facts

available dimensions

measure

In SQL: Group By + Aggregation

Select Customer, Store, SUM(Price) as Revenue, MIN(Price) as MinPrice, MAX(Price) as MaxPrice

From Sales Group by Customer, Store

Time	Customer	Store	Product	*Price*
T1	C1	S2	P1	*$90*
T2	C2	S1	P2	*$70*
T3	C1	S1	P2	*$45*
T4	C3	S1	P1	*$40*
T5	C1	S2	P2	*$25*
T6	C1	S2	P2	*$50*
T7	C2	S1	P4	*$45*
T8	C3	S1	P1	*$10*

1. Identify groups:
   - C1,S1
   - C2,S1
   - C3,S1
   - C1,S2

2. Perform aggregation

Customer	Store	*Revenue*	*Min Price*	*Max Price*
C2	S1	$115	*$45*	*$70*
C1	S1	$45	$45	$45
C3	S1	$50	*$10*	*$40*
C1	S2	$165	*$25*	*$90*

Relational Algebra (logical plan)

$$\gamma_{\text{Store, Customer, SUM(Price)->Revenue, MIN(Price)->MinPrice, MAX(Price)->MaxPrice}}$$

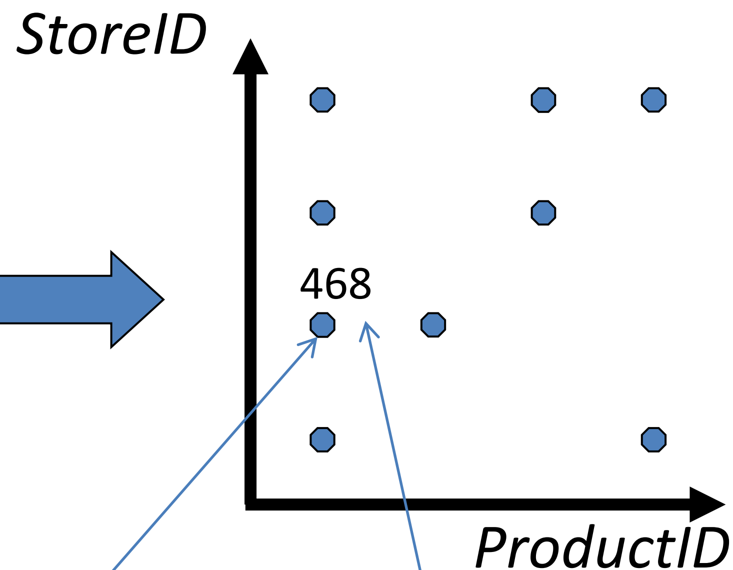
$$|$$

Sales

Map data and aggregates into a high-dimensional space

- Example: compute total *sales* volume per *productID* and *storeID*

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



All sales of ProductID 1 at storeID 2 are accumulated here

This value denotes the result of the aggregation

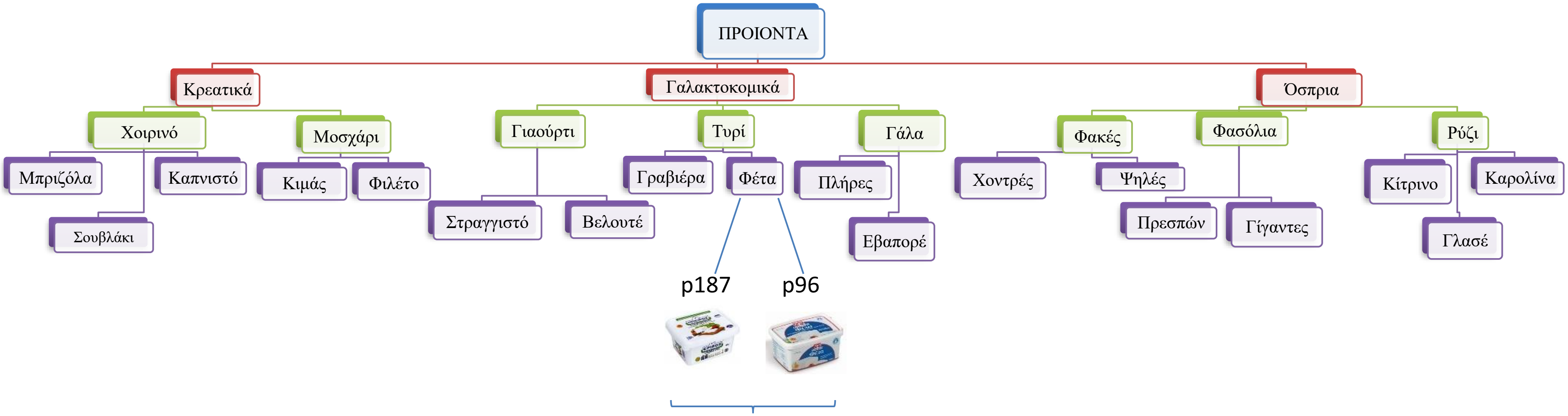
Multidimensional Data Model

- Database is a set of **facts** (points) in a multidimensional space
  - E.g. a sale/an order/a contract
- A fact has
  - A set of **dimensions** with respect to which data is analyzed
    - e.g., store, product, date associated with a sale
  - A set of **measures**
    - quantity that is analyzed, e.g., sale amount, quantity

- Dimensions form a sparsely populated coordinate system
  - Not all combinations exist as facts. E.g. a customer does not visit all stores worldwide

- Each dimension has a set of **attributes**
  - e.g., owner, city and state of store
  - Often attributes are used to encode a **hierarchy**

Product Hierarchy



Κωδικοί για όλα τα τυριά τύπου «φέτα»

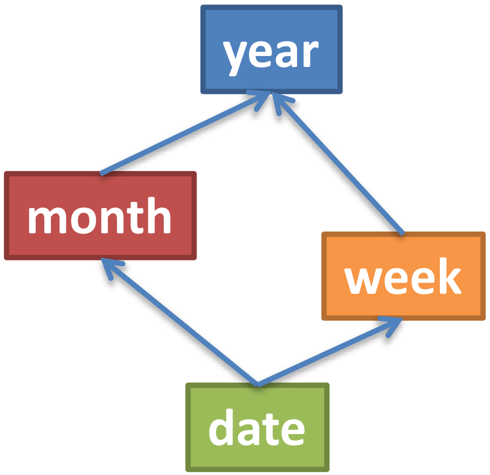
More on Attribute Hierarchies

- Values of a dimension may be related
  - Hierarchies are most common
- Dependency graph may be:
  - Hierarchy (tree): e.g.,

    city → state → country
  - Lattice:

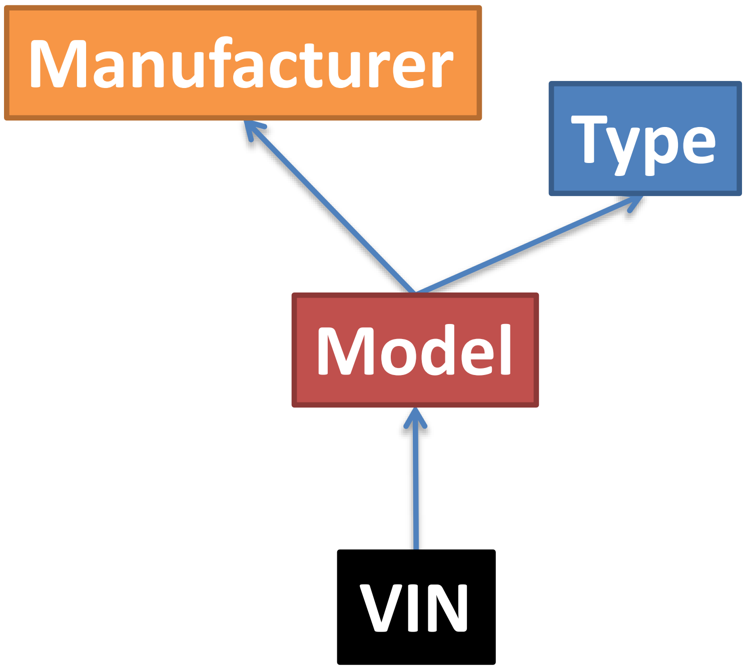
    date → month → year

    date → week (of a year) → year



Another example

- VIN: Vehicle Identification Number (unique key)
- Model: e.g. Fiesta
- Type: e.g. Compact Car
- Manufacturer: e.g. Ford



Using hierarchies

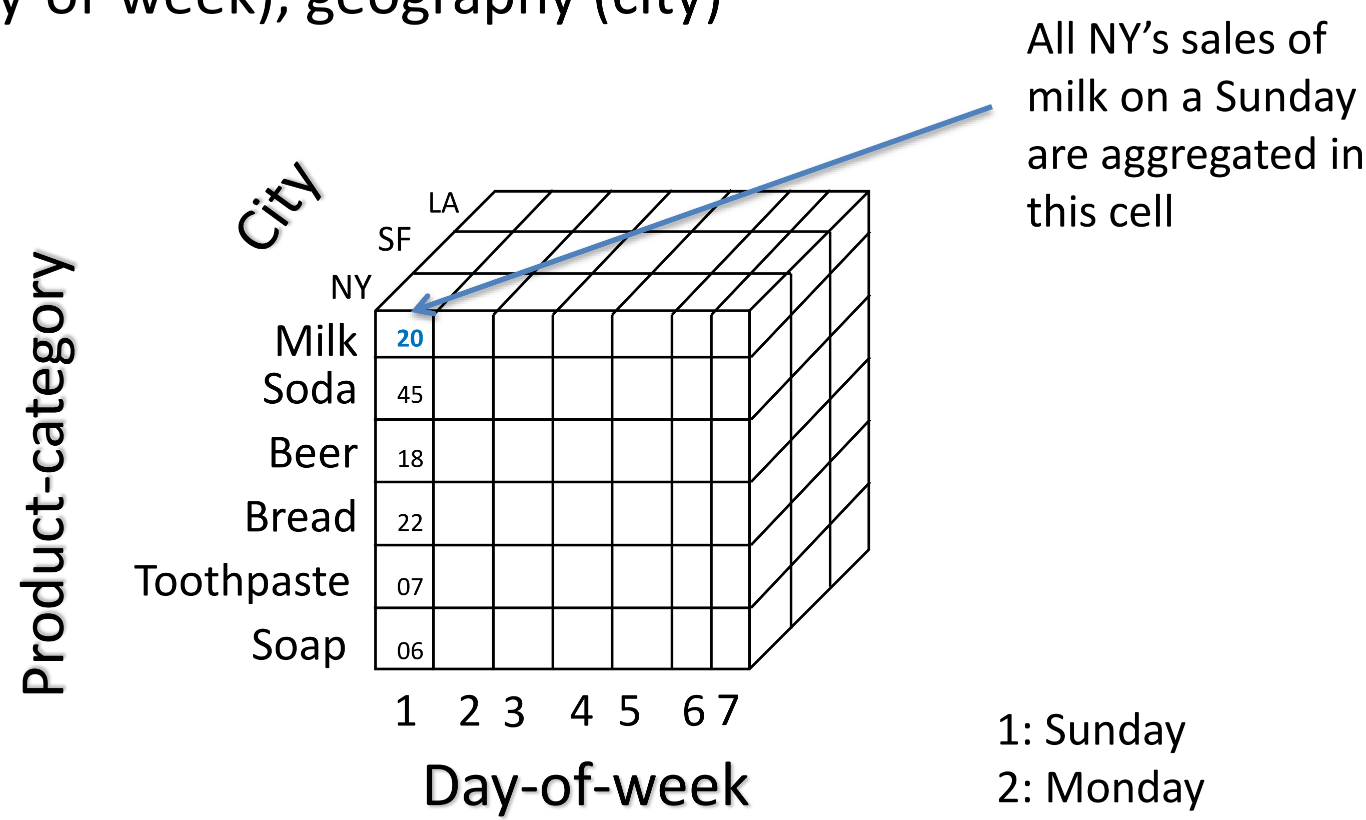
- While projecting the data into a set of dimensions, we may select an appropriate hierarchy level for each dimension
  - "Compute total sales **per productID**"

  Vs
  - "Compute total sales **per product-category**"

- In the second query, sales of different productIDs that all belong to the same category e.g. "Milk" will be accumulated together in the same "coordinate" (value) of the category dimension

Multidimensional View of selected hierarchy levels per dimension

- Aggregate sales volume as a function of product (category), time (day-of-week), geography (city)



1: Sunday
2: Monday
....

Roll-up Operation

- ***Dimension reduction***:
  - e.g., total sales by city by product
  - e.g., total sales by city

		Product			
		1	2	3	4
City	NY	$454	-	-	$925
	SF	$468	$485	-	$315
	LA	$296	-	$340	-
	SE	$652	-	$640	$645

Roll-up

		Total Sales by City
City	NY	$1379
	SF	$1268
	LA	$636
	SE	$1937

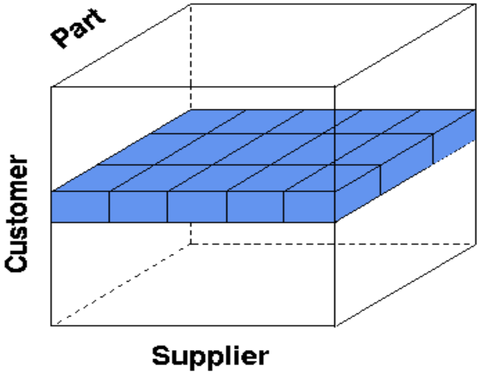
- ***Navigating attribute hierarchy***:
  - e.g., sales by **city**
    - → total sales by **state**
    - → total sales by **country**
  - e.g., total sales by **city** and **year**
    - → total sales by **state** and by **year**
    - → total sales by **country**

Drill-Down

- Drill-down: Inverse operation of roll-up
  - Provides the data set that was aggregated
    - e.g., show “base” data for total sales figure of the state of CA

Other Operations

- Selection (***slice & dice***) defines a subcube
  - Project the cube on fewer dimensions by specifying coordinates of remaining dimensions
  - e.g., sales to customer XXX
- Ranking
  - top 3% of cities by average sales



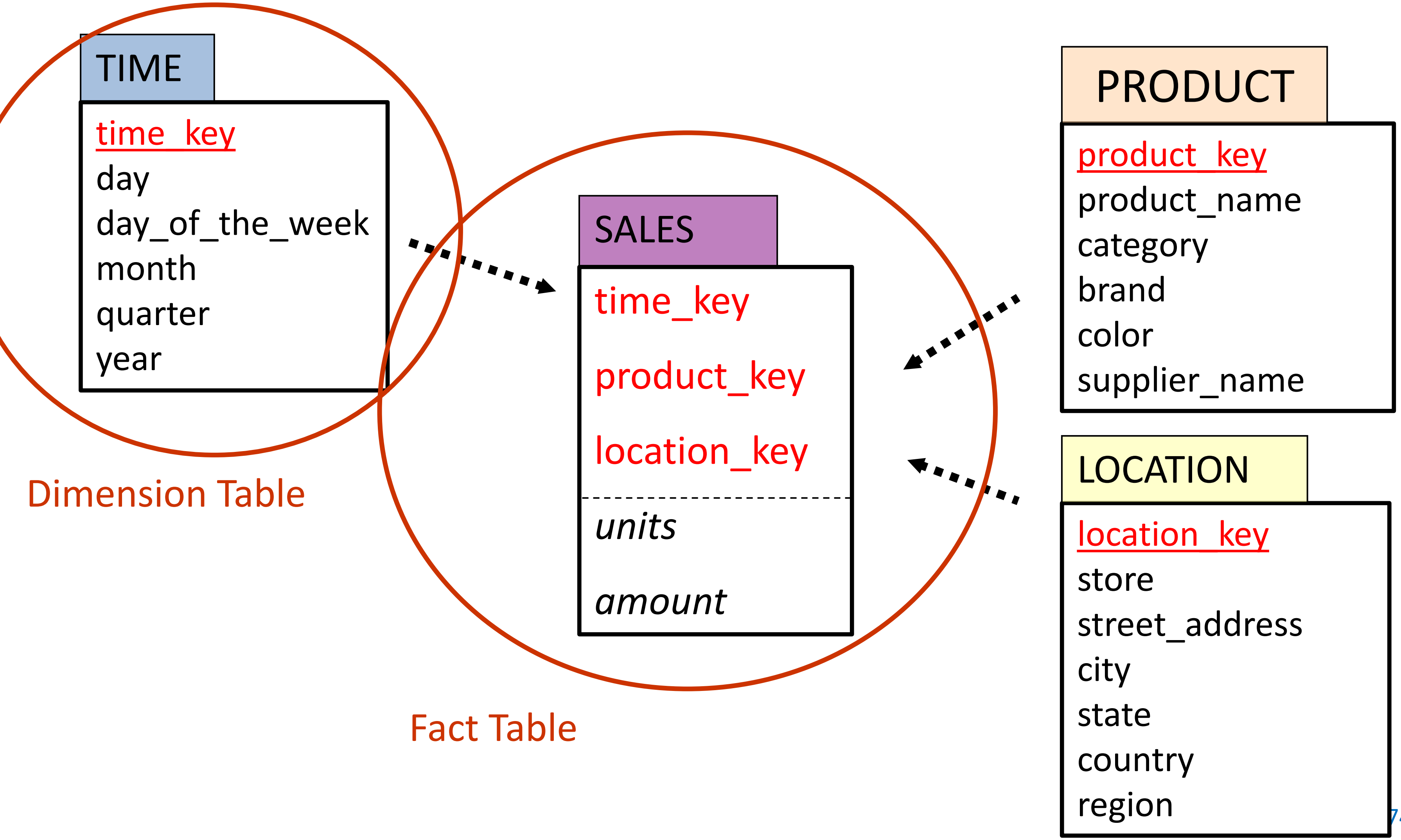
Fact data: Sales sum in $

Warehouse Database Schema

- Relational design should reflect multidimensional view
- Typical schemas:
  - Star Schema
  - Snowflake Schema
  - Fact Constellation Schema

- Data tables (relations) are of two types: **fact tables** and **dimension tables**

The Star Schema (Example 1)



SALES

time_key	product_key	location_key	units	amount
T1	P44	L4	1	12
T2	P157	L4	3	180
T2	P6	L1	14	2560
T3	P25	L3	1	2
T3	P157	L1	1	60

Foreign keys to dimension tables measures

Fact Table

- A table in the data warehouse that contains facts consisting of
  - Numerical performance **measures**
  - **Foreign keys** that tie the fact data to the dimension tables
- Each row records measurements describing a fact
  - Where? When? Who? How much? How many?
- Provides the most detailed view of the data an analyst has access to in the data warehouse
  - this denotes the **grain** of the design

Dimension Tables

product_key	product_name	category	brand	color	supplier name
P1	I7-8600K	CPU	Intel	black	Jim
P2	I5-2400	CPU	Intel	black	Jim
P3	Samsung 830	SSD	Samsung	brown	Ben
P4	Barracuda	HDD	Seagate	silver	Ben
P5	MQ01ABD032	HDD	Toshiba	silver	John

Keys uniquely identify each product

encodes product → category hierarchy

- Dimension Tables contain
  - a key column linked to a foreign key in the fact table
  - textual descriptors such as name of products, addresses etc
  - attributes that encode dependences within the dimension (e.g. hierarchies)
- Dimension tables may be wide
- Dimension tables are usually shallow (e.g. few thousand rows)

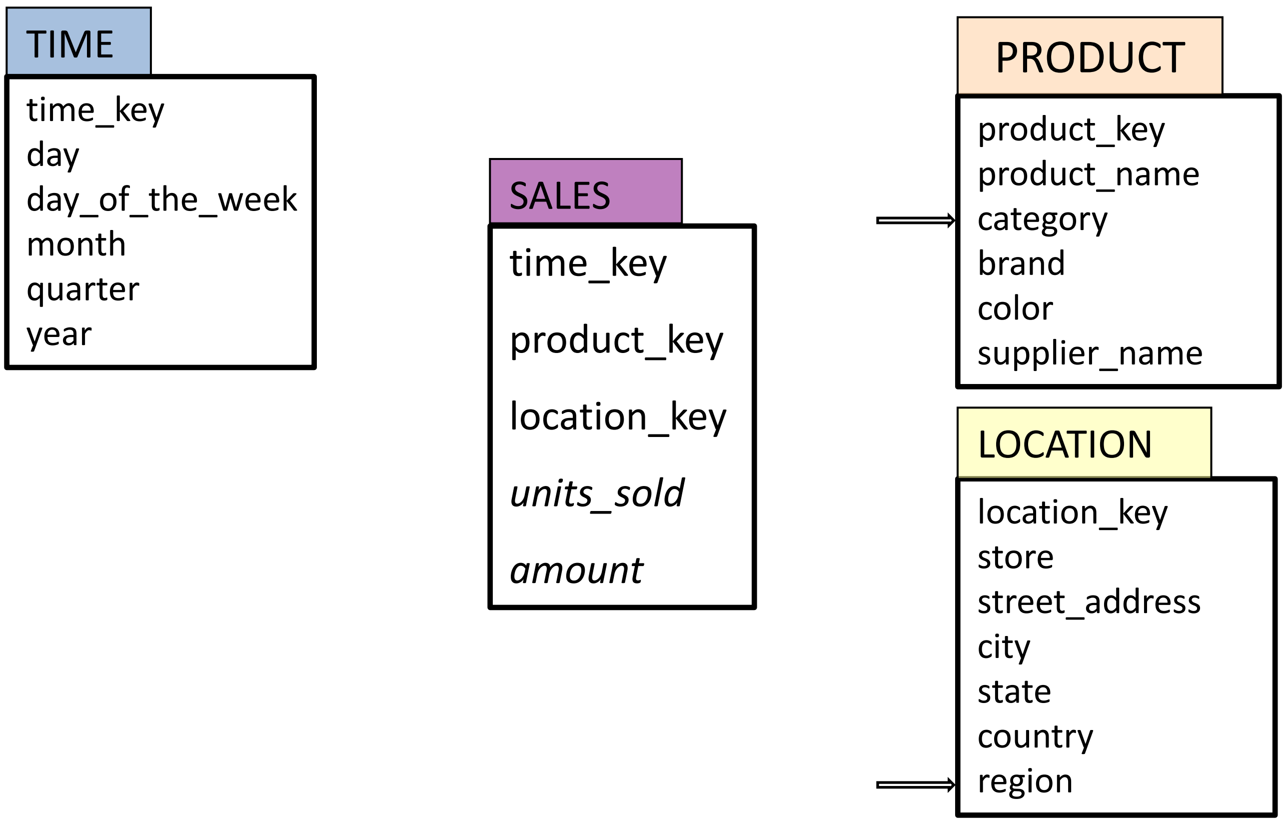
Advantages of Star Schema

- A single fact table where to look for facts to analyze
- One table for each dimension
  - dimensions are clearly depicted in the schema
- Easy to comprehend (and write queries)

- Loading of data
  - dimension tables are relatively static
  - data is loaded (append mostly) into fact table(s)
  - new indexing opportunities

Querying the Star Schema

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



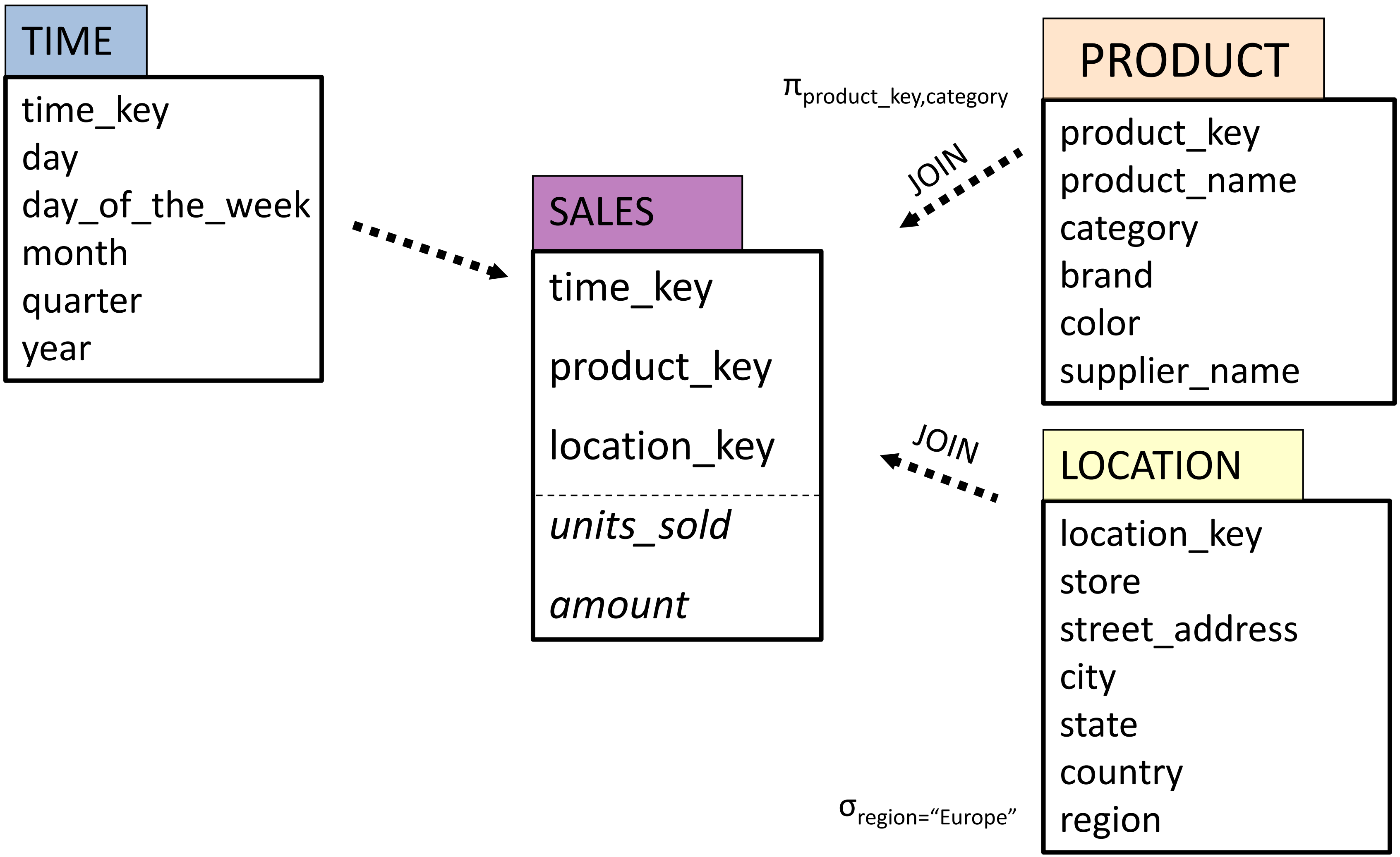
Querying the Star Schema

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

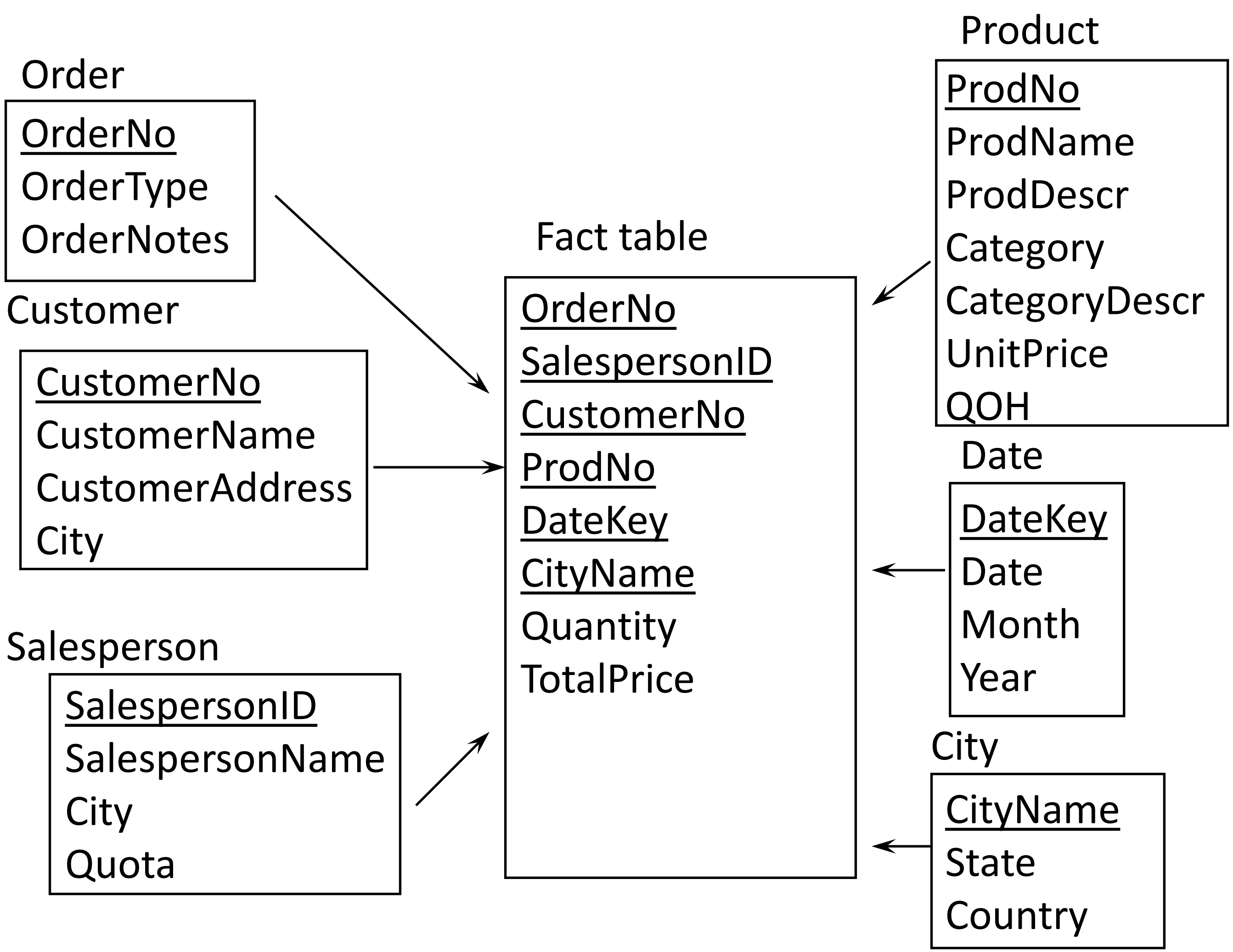
```
SELECT PRODUCT.category, SUM(SALES.amount)
FROM     SALES, PRODUCT,LOCATION
WHERE    SALES.product_key = PRODUCT.product_key
AND      SALES.location_key = LOCATION.location_key
AND      LOCATION.region=“Europe”
GROUP BY PRODUCT.category
```

Join fact table SALES with dimension tables PRODUCT, LOCATION to fetch required attributes (category & region in this example)

Star Schema Query Processing

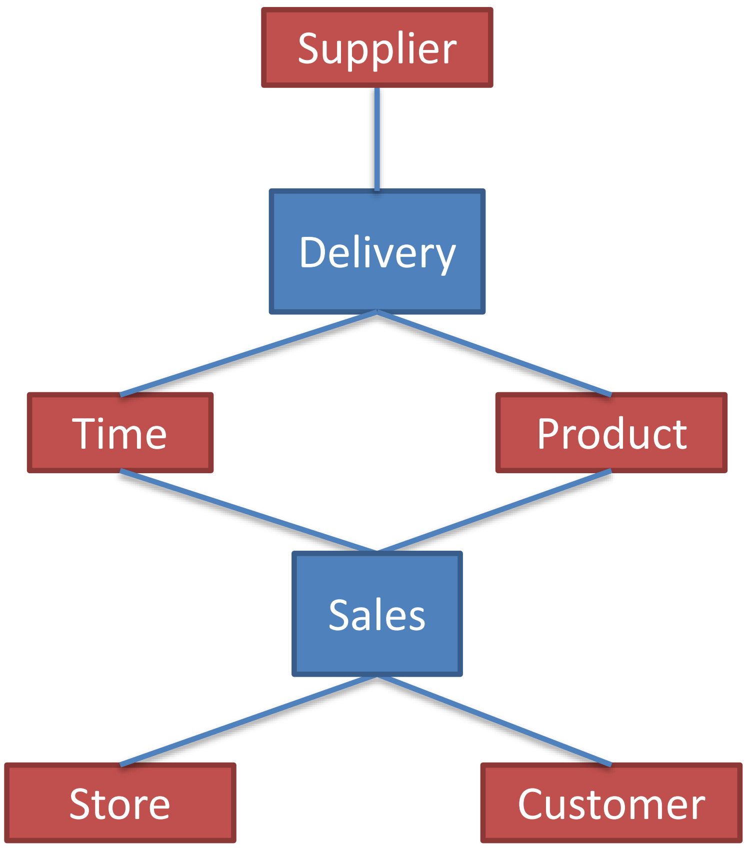


Another Example

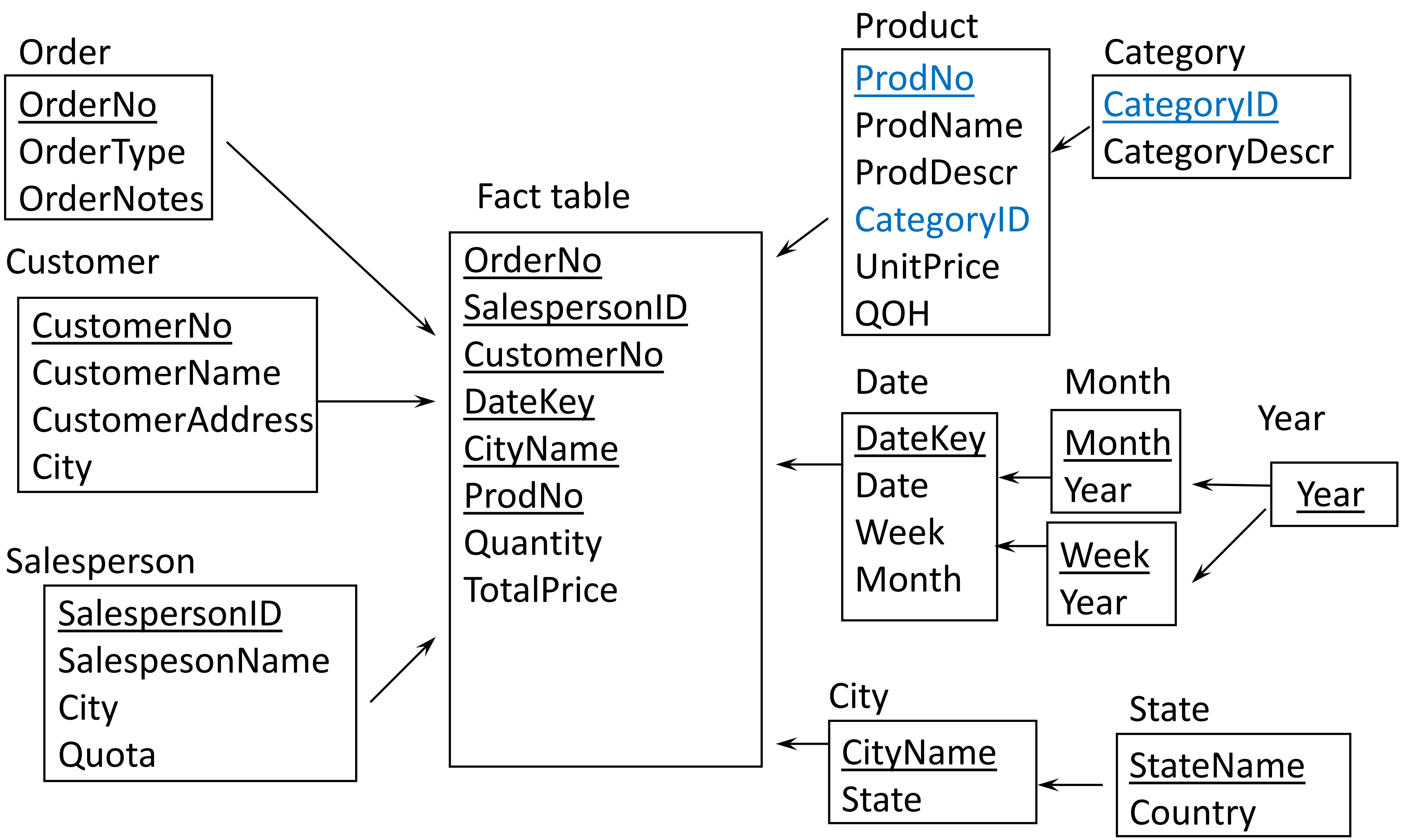


Fact constellation

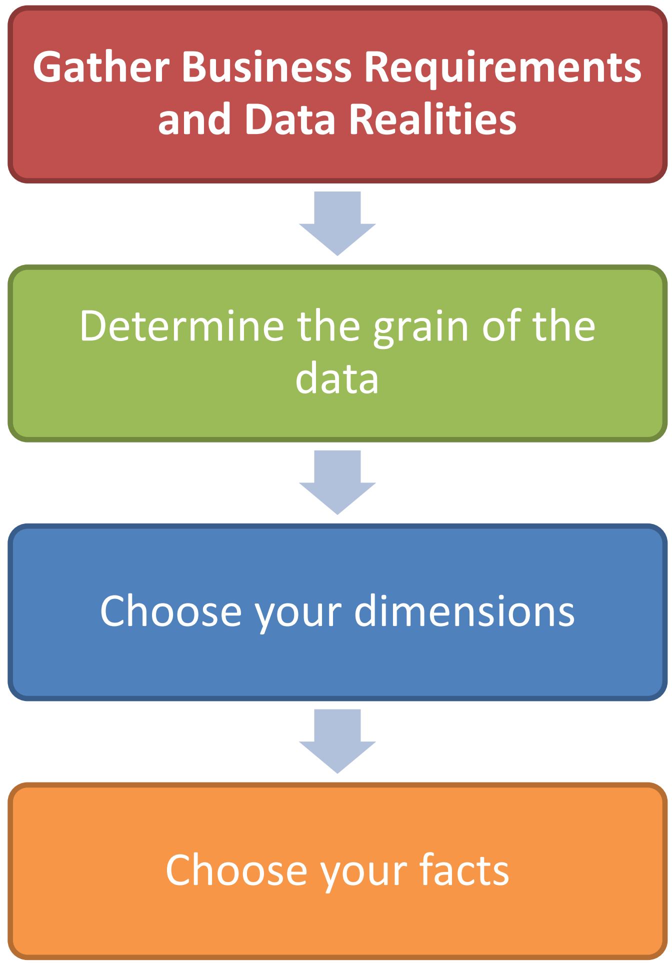
- Multiple fact tables that share common dimension tables
  - Example: Delivery and Sales fact tables share dimension tables Time & Product



Snowflake Schema: represents dimensional hierarchy by normalization



Multidimensional Modeling Stages
(adapted from https://www.kimballgroup.com/)



Gather Business Requirements and Data Realities

- Study the underlying business processes
  - Understand their objectives based on key performance indicators (KPIs), compelling business issues, decision-making processes, and supporting analytic need
- Identify available data sources (internal and external)
  - Assess their quality and completeness

Grain

- Establishes exactly what a single fact table row represents
  - Different grains must not be mixed in the same fact table
- Atomic grain refers to the lowest level at which data is captured by a given business process
  - Safer to start with the atomic grain in order to cope with unpredictable query workload

Identify the dimensions

- *Dimensions* provide the "who, what, where, when, why, and how" context surrounding a business process event.
- Dimension tables contain descriptive attributes used by BI applications for filtering and grouping the facts.

Identify the facts

- A single fact table row has a one-to-one relationship to a measurement event as described by the fact table's grain.

- *Facts* contain measurements that result from a business process event.

- Within a fact table, only facts consistent with the declared grain are allowed.

Indexing Techniques

- Exploiting indexes to reduce scanning of data is of crucial importance
- ROLAP
  - Bitmap Indexes
  - Join Indexes
- MOLAP
  - Array representation

Bitmap Index Example

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

RowID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Bitmap Index Example

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

RowID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Bitmap encodes position of customer records in the base table (rows 1,7) that reside in the North Region

Bitmap Index Example

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

RowID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Rating Index

RowID	H	M	L
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	0	1
7	1	0	0

Bitmap Index Example

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

RowID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Rating Index

RowID	H	M	L
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	0	1
7	1	0	0

Customers *where* *Region = W* *and* *Rating = L*

0011010 AND 0010110=0010010 (rows 3,6)

Bit Map Index Example 2

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

RowID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

How many customers in W region?

Bitmap Index

- An alternative representation of RID-list
- Comparison, join and aggregation operations are reduced to *bit arithmetic*
- Especially advantageous for low-cardinality domains
  - Significant reduction in space and I/O (30:1)
  - Have been adapted for higher cardinality domains
  - Compression (e.g., run-length encoding) exploited
- Products: Model 204, Redbrick, IQ (Sybase), Oracle, etc

Join Index

- Traditional index maps the value in a column to a list of rows with that value
- Join index maintains relationships between attribute value of a dimension and the matching rows in the fact table
- Join index may span multiple dimensions (composite join index)

Example: Join Indexes

- "Combine" SALE, PRODUCT relations

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

Join Indexes

join index

product	id	name	price	jIndex
	p1	bolt	10	r1,r3,r5,r6
	p2	nut	5	r2,r4

sale	rId	prodId	storeId	date	amt
	r1	p1	c1	1	12
	r2	p2	c1	1	11
	r3	p1	c3	1	50
	r4	p2	c2	1	8
	r5	p1	c1	2	44
	r6	p1	c2	2	4

Example: Compute total sales in AFRICA

TIME

time_key
day
day_of_the_week
month
quarter
year

SALES

time_key
product_key
location_key

---

units_sold
amount

PRODUCT

product_key
product_name
category
brand
color
supplier_name

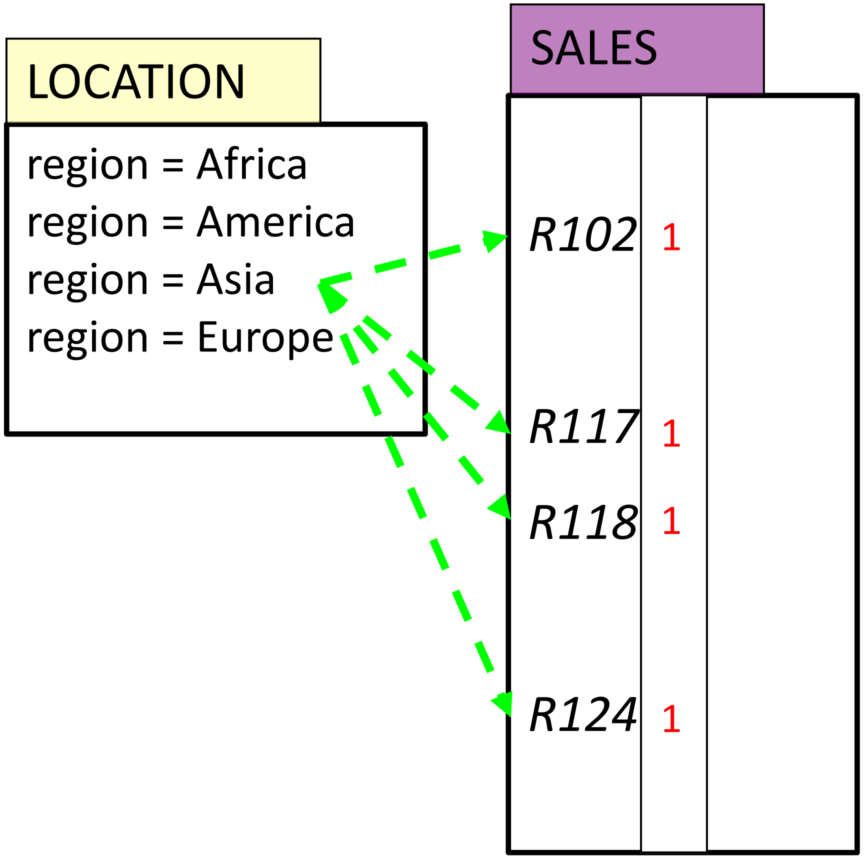
LOCATION

location_key
store
street_address
city
state
country
region

```
SELECT SUM(sales.amount)
FROM sales, location
WHERE sales.location_key=location.location_key
AND location.region=“AFRICA”
```

Join-Index in the Star Schema

- Join index relates the values of the <u>dimensions</u> of a star schema to <u>rows</u> 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 be implemented as bitmap-indexes (next slides)



Join Index on Location.Region implemented as bitmap index

Fact Table Sales

time_key	product_key	location_key	units	amount
T1	P44	L4	1	12
T2	P157	L4	3	180
T2	P6	L1	14	2560
T3	P25	L3	1	2
T3	P157	L1	1	60

Bitmaps for Location.Region

Africa	Asia	Europe	America
0	0	0	1
0	0	0	1
1	0	0	0
0	0	1	0
1	0	0	0

Assuming L1 refers to a store location in Africa, L2 to a store location in Asia etc
This information is stored in the dimension table Location

In SQL

- Join index implemented as bitmap index:
  CREATE BITMAP INDEX loc_sales_bit
  ON sales(location.region)
  FROM sales, location
  WHERE sales.loc_location_key = location.location_key;
- The following query uses the index to avoid computing the join
  SELECT SUM(sales.amount)
  FROM sales,location
  WHERE sales.location_key=location.location_key
  AND location.region="AFRICA"

THE DATA CUBE

Aggregation
(on a single group via filtering)

- Sum up amounts for day 1
- In SQL: SELECT sum(amt)
  FROM SALE
  WHERE day = 1

Assume following fact table:

sale	prodId	storeId	day	amt
	p1	s1	1	12
	p2	s1	1	11
	p1	s3	1	50
	p2	s2	1	8
	p1	s1	2	44
	p1	s2	2	4

81

Group by & Aggregation

- Sum up amounts by day

SELECT day, sum(amt) FROM SALE
GROUP BY day

sale	prodId	storeId	day	amt
	p1	s1	1	12
	p2	s1	1	11
	p1	s3	1	50
	p2	s2	1	8
	p1	s1	2	44
	p1	s2	2	4

ans	day	sum
	1	81
	2	48

Common operations

- Sum up amounts by day, product
- In SQL: SELECT prodid,day,sum(amt) FROM SALE
  GROUP BY prodId, day

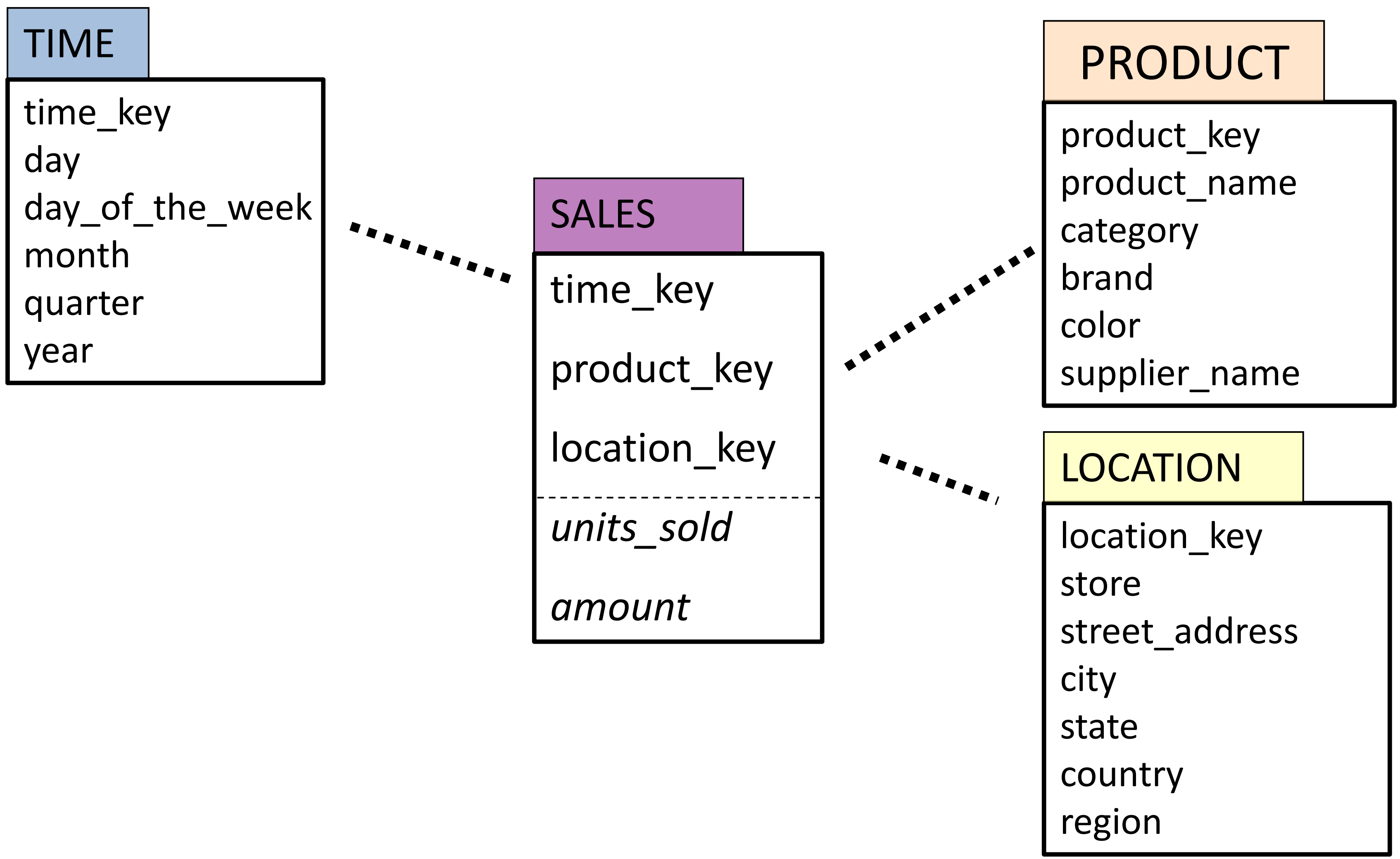
sale	prodId	storeId	day	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

sale	prodId	day	amt
	p1	1	62
	p2	1	19
	p1	2	48

rollup →

← drill-down

Recall: Star Schema Example 1



Compute volume of sales per product_key and store

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

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	652
4	3	540
4	4	745

SQL: SELECT LOCATION.store, SALES.product_key, SUM (amount)

FROM SALES, LOCATION

WHERE SALES.location_key=LOCATION.location_key

GROUP BY SALES.product_key, LOCATION.store

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

How many queries to obtain this result?

Sales		Product_key				
		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

Sub-totals per product_key

Total sales

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

Sales		Product_key				
		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

Aggregate sales
group by (store,product_key)

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

Sales		Product_key				
		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

Aggregate sales group by (store)

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

Sales		Product_key				
		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

Aggregate sales
group by (product_key)

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

Sales		Product_key				
		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

What is this?

Multiple Simultaneous Aggregates

Cross-Tabulation (products/store)

Sales		Product_key				
		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

Sub-totals per product_key

Total sales

<u>4 Group-bys here:</u>
(store,product_key)
(store)
(product_key)
()

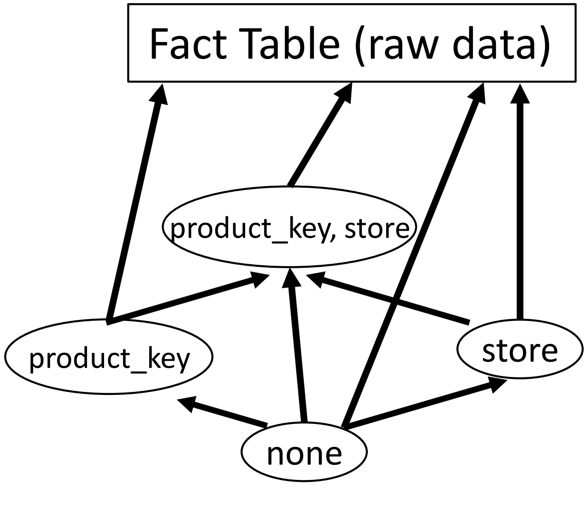
Need to write 4 queries!!!

Multiple Simultaneous Aggregates: Optimizations?

Cross-Tabulation (products/store)

Sales		Product_key				
		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

<u>4 Group-bys here:</u>
(store,product_key)
(store)
(product_key)
()



The Data Cube Operator

(Gray et al)

- All previous aggregates in a single query:

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

Challenge: Optimize Cube Computation

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

GROUP BY SALES.product_key, LOCATION.store
WITH CUBE

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	652
4	3	540
4	4	745
1	ALL	1379
2	ALL	1268
3	ALL	536
4	ALL	1937
ALL	1	1870
ALL	2	800
ALL	3	780
ALL	4	1670
ALL	ALL	5120

Quiz

- SALES(customer,sales_person,store,product,amt)
- Assume the SUM() aggregate function
- What is the meaning of the following data cube records?

(ALL,'JOHN',ALL,ALL,5000)

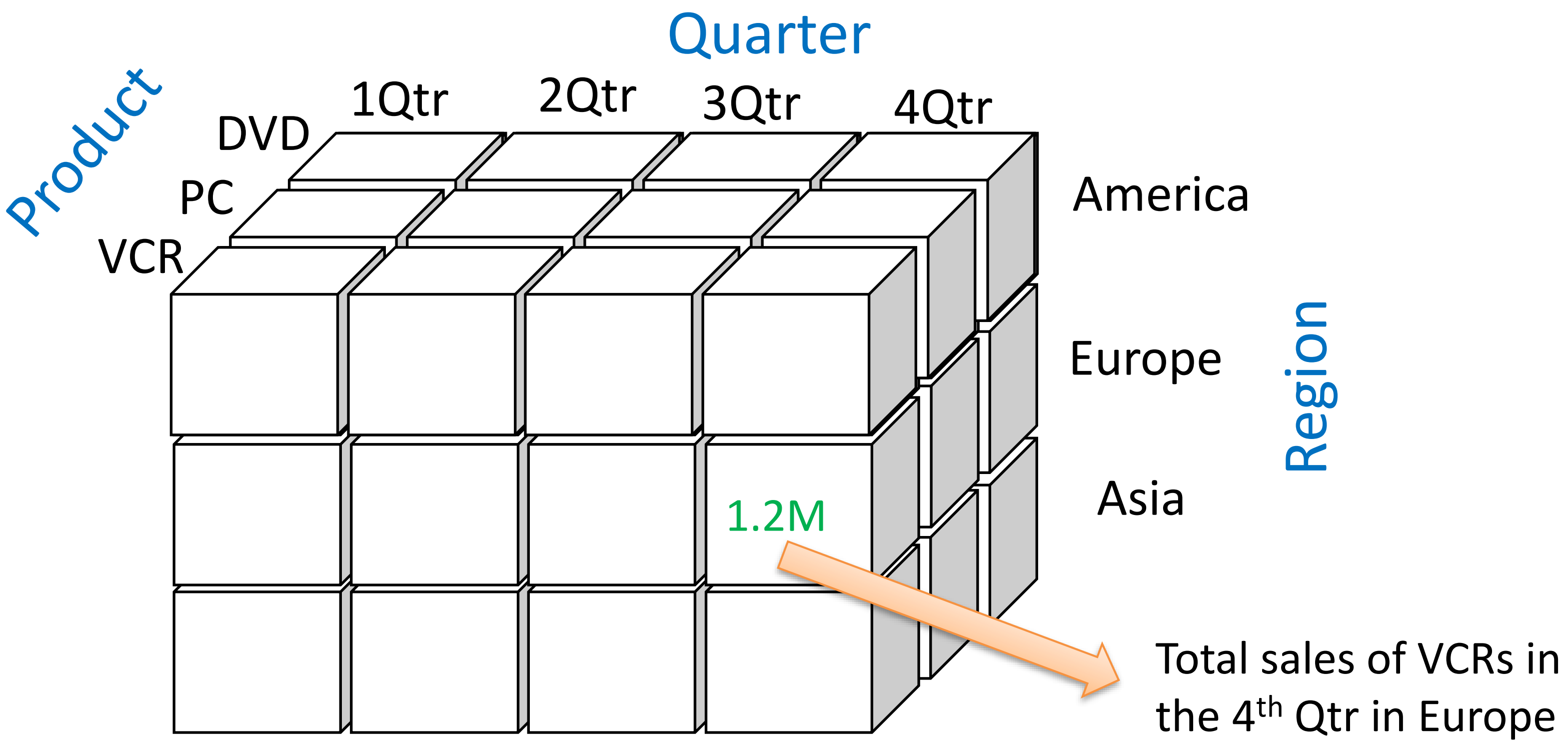
('NICK',ALL,ALL,'BEER',250)

(ALL,ALL,ALL,'MILK',70000)

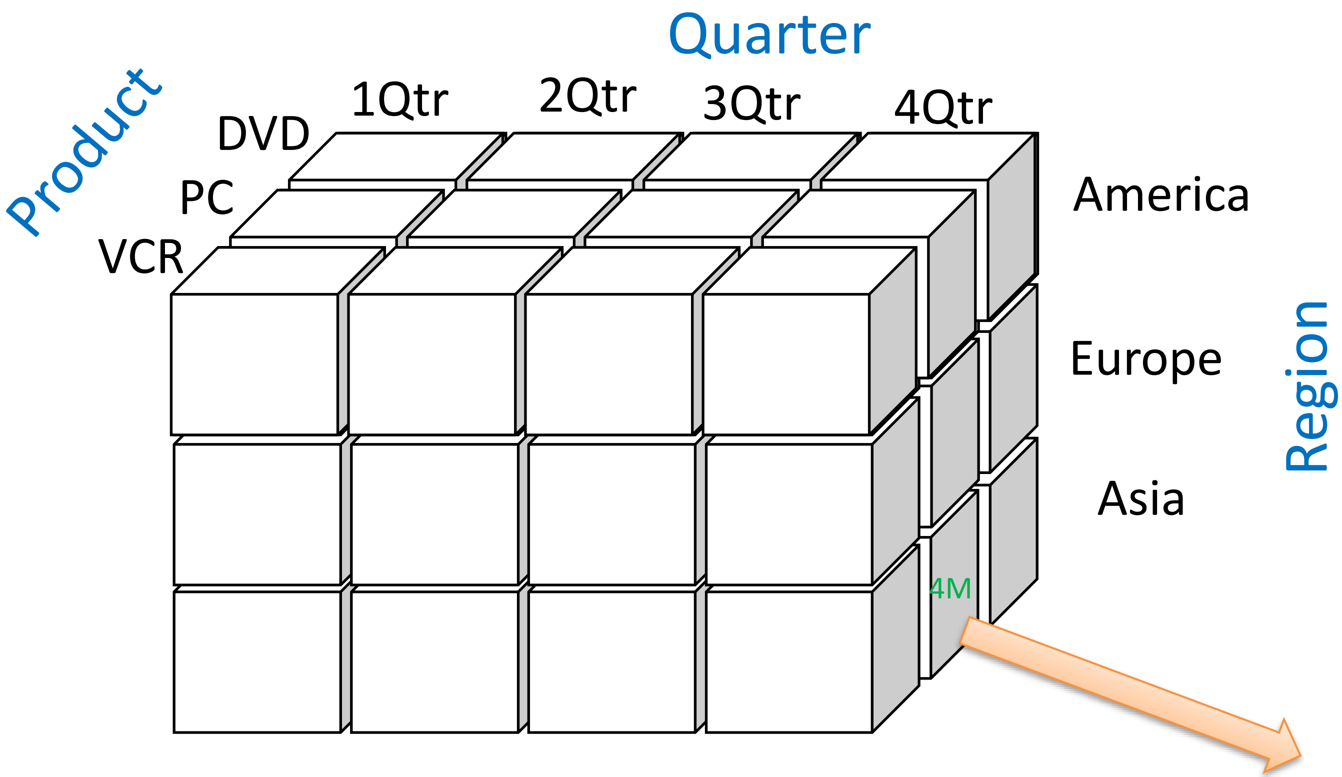
(ALL,ALL,ALL,ALL,250000)

Group by (Product, Quarter, Region)

SUM() aggregate function

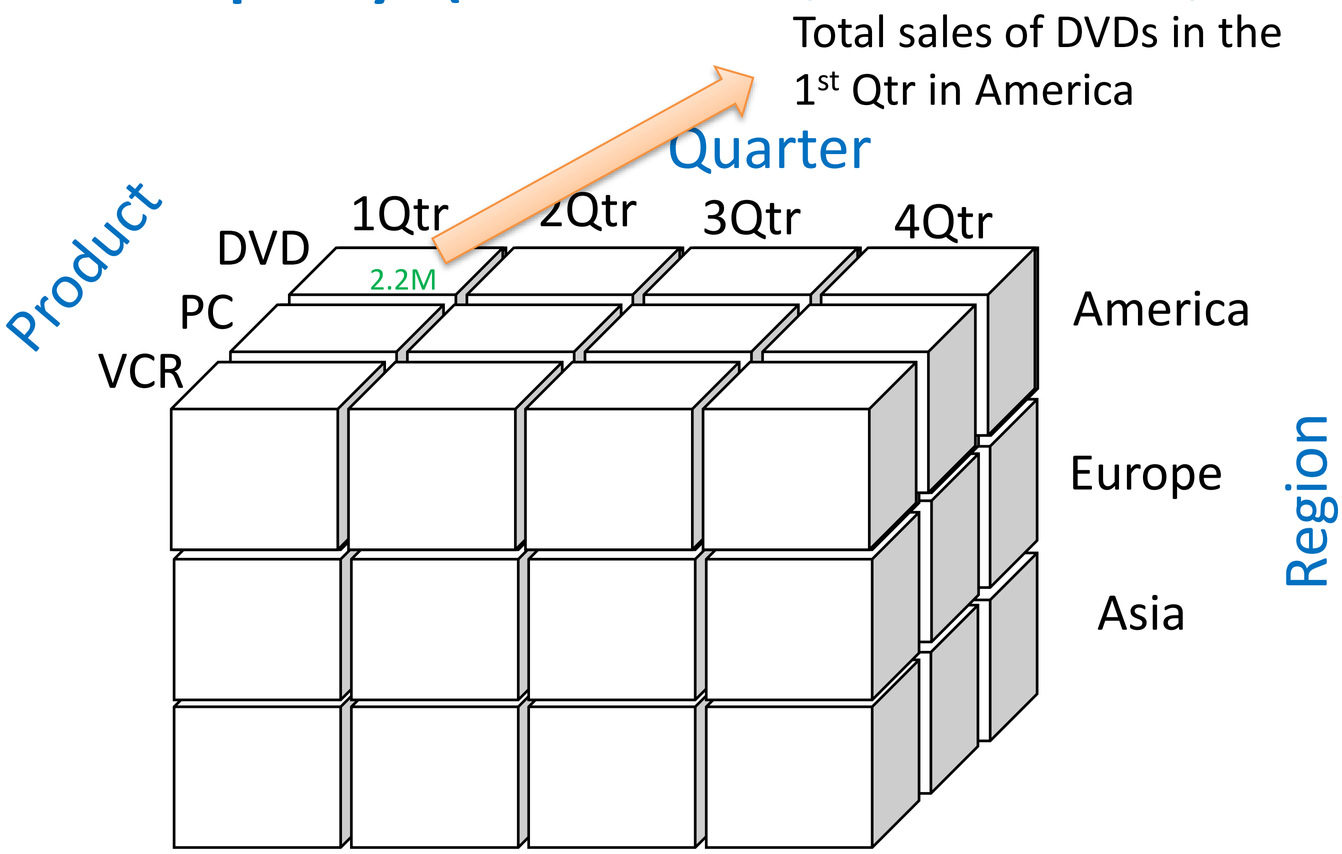


Group by (Product, Quarter, Region)

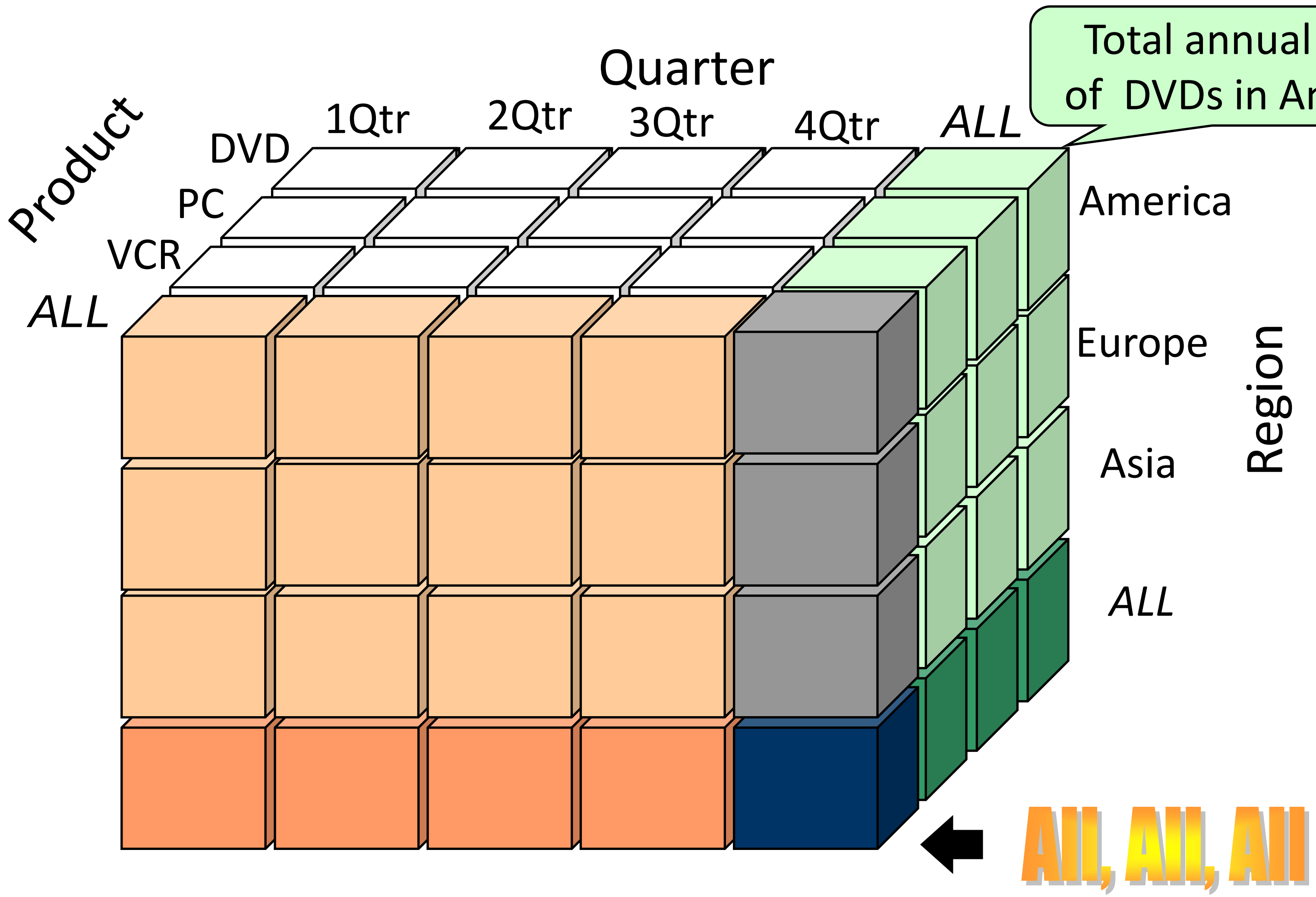


Total sales of PCs in the 4th Qtr in Asia

Group by (Product, Quarter, Region)



Data Cube: Multidimensional View



How are aggregates computed?

1. Bring all records with same values in the groupping attributes together
2. Aggregate their measures

- (1) is done via Hashing / Sorting
- (2) depends on the type of function used
  - Simple calculations for max, sum, count etc
  - Harder for median

Example: Sum sales/prodId ?

Raw data (fact table)

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p2	s1	1	11
	p1	s3	1	50
	p2	s2	1	8
	p1	s1	2	44
	p1	s2	2	4

Step 1: Sort tuples by prodId

Raw data (fact table)

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p2	s1	1	11
	p1	s3	1	50
	p2	s2	1	8
	p1	s1	2	44
	p1	s2	2	4

Sort(prodId)

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p1	s1	2	44
	p1	s2	2	4
	p1	s3	1	50
	p2	s1	1	11
	p2	s2	1	8

Step 2: Aggregate records (sum amt)

Sorted Raw data

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p1	s1	2	44
	p1	s2	2	4
	p1	s3	1	50
	p2	s1	1	11
	p2	s2	1	8

Sales for prodId=1

Aggregate

ans	prodId	sum
	p1	110
	p2	19

More on aggregate

- Assumed SUM() function
- How much space needed?
- How about AVG()?
- How about MEDIAN()?

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p1	s1	2	44
	p1	s2	2	4
	p1	s3	1	50
	p2	s1	1	11
	p2	s2	1	8

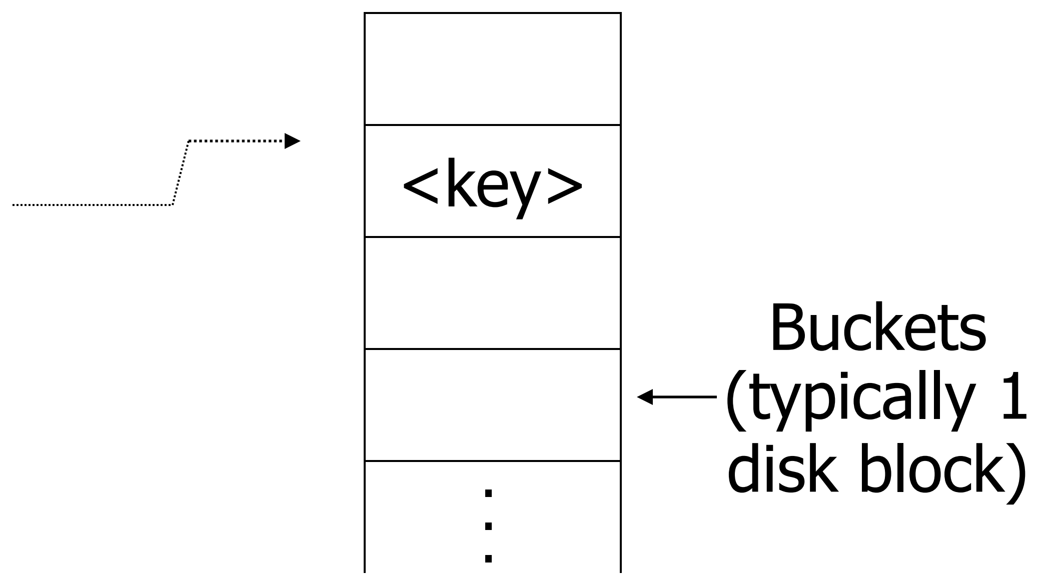
Aggregate Computation

- Certain functions (SUM,MIN,MAX,COUNT,AVERAGE, etc) require small (bounded) space for storing their state and may be computed on the fly, while executing the merging phase of the 2-phase sort algorithm.

- Cost = 3 * B(R) , assuming $M^2 \geq B(R) > M$

Hashing

key → h(key)



Example: 2 records/bucket

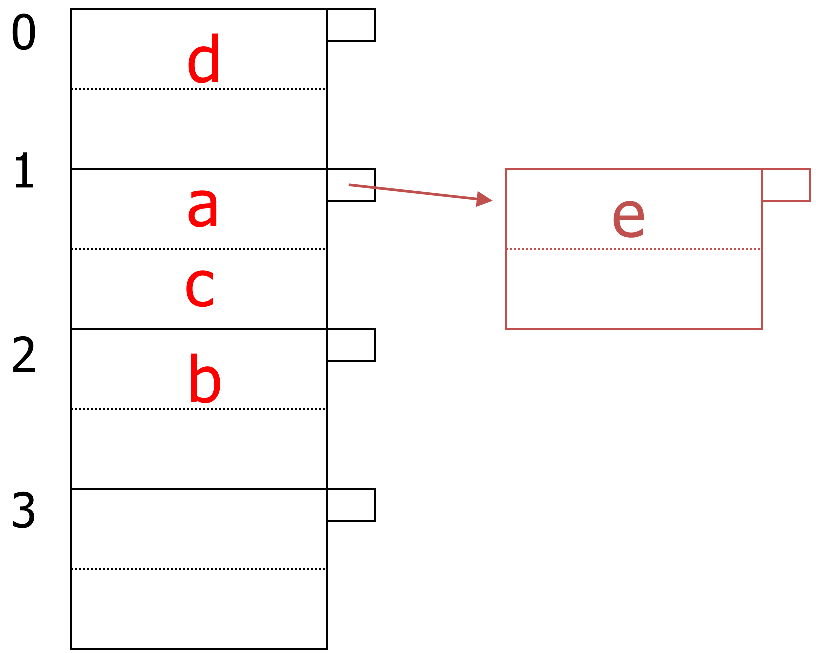
INSERT:

h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0



h(e) = 1

How does this work for aggregates?

Hash on prodId

prodId	storeId	date	amt
p1	s1	1	12
p1	s3	1	50
p1	s1	2	44
p1	s2	2	4
p3	s5	1	7
p7	s2	2	1

Possibly keep records sorted within bucket

Two buckets

$h(\textit{prodId}) =$
$\textit{prodId} \bmod 2$

prodId	storeId	date	amt
p2	s1	1	11
p2	s2	1	8

Not the best hash function

Naïve Data Cube Computation

- Fact table:

sale	prodId	storeId	amt
	p1	s1	12
	p2	s1	11
	p1	s3	50
	p2	s2	8
	p1	s1	44
	p1	s2	4

- Compute: SUM(amt) GROUP BY prodId,storeId WITH CUBE
  - 4 group bys contained in this Data Cube:

prodId	storeId	sum(amt)
p1	s1	56
p1	s2	4
p1	s3	50
p2	s1	11
p2	s2	8

prodId	amt
p1	110
p2	19

storeId	amt
s1	67
s2	12
s3	50

amt
129

Full Data Cube
(from previous example)

prodId	storeId	sum(amt)
p1	s1	56
p1	s2	4
p1	s3	50
p2	s1	11
p2	s2	8
p1	ALL	110
p2	ALL	19
ALL	s1	67
ALL	s2	12
ALL	s3	50
ALL	ALL	129

How much does it cost to compute?

- Assume B(SALES)=1 Million Blocks, larger than main memory
- Our (brute force) strategy: compute each group by indepedently
  - Compute GROUP BY prodId,storeId
  - Compute GROUP BY prodId
  - Compute GROUP BY storeId
  - Compute GROUP BY none (=total amt)

First Group By: prodId,storeId

- In SQL

```
SELECT prodId,storeId,sum(amt)
FROM SALES
GROUP BY prodId,storeId
```

- Use sorting: 3*B(SALES) = 3M I/O

Second Group By: prodId

- In SQL

```
SELECT prodId,sum(amt)
FROM SALES
GROUP BY prodId
```

- Use sorting: 3*B(SALES) = 3M I/O (same)

Third Group By: storeId

- In SQL

SELECT storeId,sum(amt)

FROM SALES

GROUP BY storeId

- Use sorting: 3*B(SALES) = 3M I/O (same)

Group By (none) = sum(amt)

- SQL:

SELECT sum(amt)

FROM SALES

- Cost ?

Recap

- Group By prodId,storeId : 3M I/Os
- Group By prodId : 3M I/Os
- Group By storeId : 3M I/Os
- Group By none : 1M I/Os
  - Compute aggregate function over all records, no sorting necessary

- Total Cost for the Data Cube: 10M I/Os
  - Is this a lot?

Practice Problem

- Rotation speed 7200rpm
- 128 sectors/track
- 4096 bytes/sector
- 4 sectors/block (16KB page size)
- Sequential I/O: ignore SEEKTIME, gaps, etc

Sustained disk speed

- 1 full rotation
  - takes 60/7200=8.33ms
  - retrieves 1 track = 128 sectors = 32 pages (blocks)
- 10 Million blocks in

  8.33/1000 * 10M/32 = 43.5 minutes

- Can we do better?

Share sort orders

If sorted on (prodId,storeId)

prodId	storeId	date	amt
p1	s1	1	12
p1	s1	2	44
p1	s2	2	4
p1	s3	1	50
p2	s1	1	11
p2	s2	1	8

=

Then, also sorted on (prodId)

prodId	storeId	date	amt
p1	s1	1	12
p1	s1	2	44
p1	s2	2	4
p1	s3	1	50
p2	s1	1	11
p2	s2	1	8

Thus, no need to sort SALES twice!

Two group-bys with a single sort on (prodId, storeId)

Output of 2-phase sort algorithm
(one row at a time)

prodId	storeId	date	amt
p1	s1	1	12
p1	s1	2	44
p1	s2	2	4
p1	s3	1	50
p2	s1	1	11
p2	s2	1	8

EOT (End-Of-Table)

+

Maintain 2 variables

SUM1	SUM2
12	12
56	56
4	60
50	110
11	11
8	19

output

p1,s1,56
p1,s2,4
p1,s3,50 p1,110
p2,s1,11
p2,s2,8 p2,19

- SUM1 is used for group-by(prodId,storeId), SUM2 for group-by(prodId)

-Each time we see a new (prodId,storeId) combination we report the previous pair and SUM1 value and initialize SUM1 to the new amt

- Similar logic for SUM2

- Report last combination at EOT

Share sort orders for multiple group bys

- Sort SALES on prodId,storeId
  - **At the merging phase compute both group by prodId and prodId,storeId**
  - **Also compute** group by none
- Then compute group by storeId by sorting SALES on storeId

prodId	storeId	date	amt
p1	s1	1	12
p1	s1	2	44
p1	s2	2	4
p1	s3	1	50
p2	s1	1	11
p2	s2	1	8

- Cost = 3B(SALES) + 3B(SALES) = 6M I/Os
  - Compared to 10M I/Os
  - 40% savings

Can we do better?

- Sort SALES on prodId,storeId
  - At the merging phase compute both group by (prodId,storeId)) and group by (prodId)
  - Also compute group by none at the same time
- **Compute group by (storeId) by sorting the result of group by (prodId,storeId) on storeId**
  - **Notice that by construction B(gb(prodId,storeId)) ≤ B(SALES)**
    - **Each tuple in gb(prodId,storeId) is produced by one or more tuples in SALES**

gb(prodId,storeId)

prodId	storeId	sum(amt)
p1	s1	56
p1	s2	4
p1	s3	50
p2	s1	11
p2	s2	8

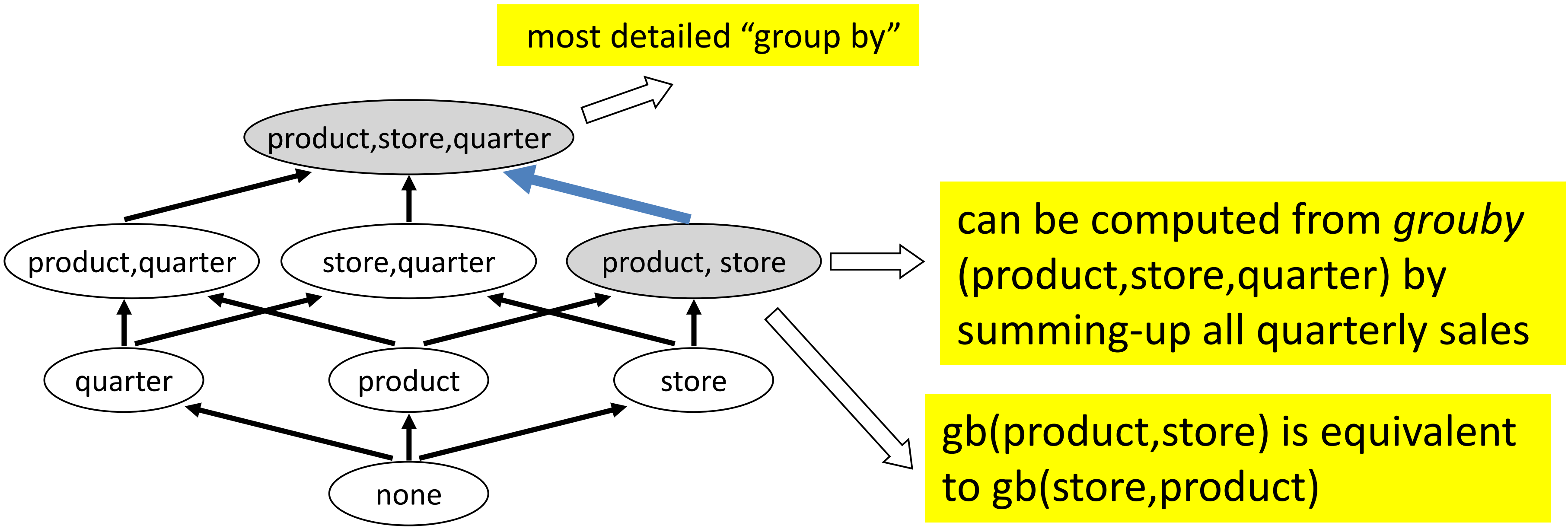
gb(storeId)

storeId	sum(amt)
s1	67
s2	12
s3	50

Cost = 3*B(SALES) + 3*B(gb(prodId,storeId))

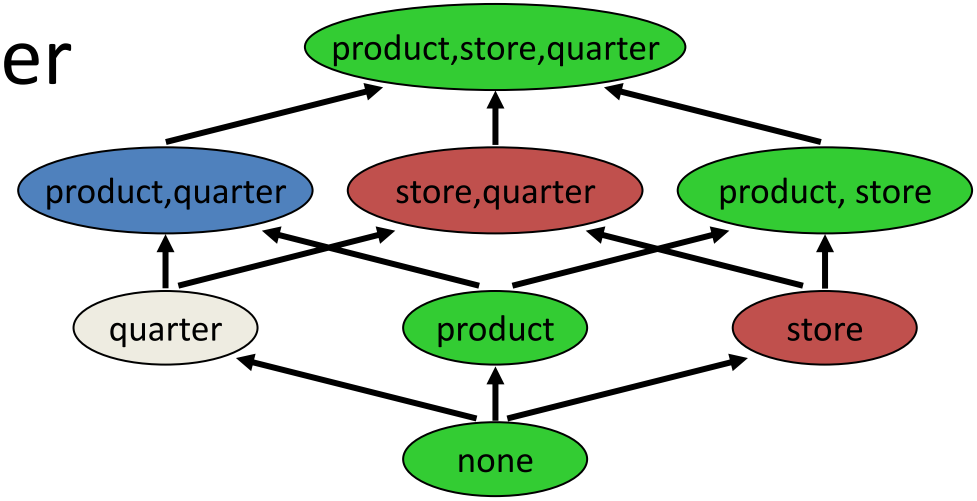
3D Data Cube Lattice

- Model dependencies among the aggregates (independently of the method of computation, e.g. by sorting or otherwise)



Discussed optimization (sharing sort orders) on the 3D Data Cube

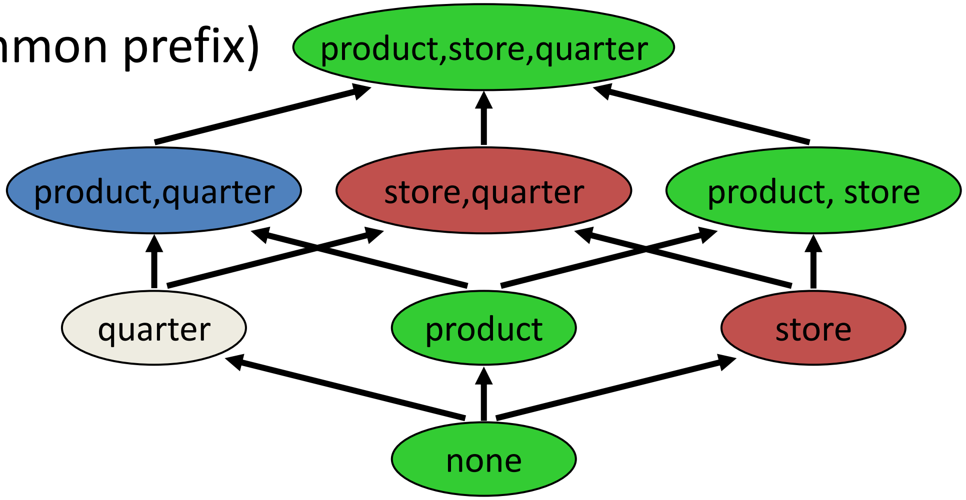
- Sort SALES on product,store,quarter (also get gb product,store, gb product and gb none)
- Sort SALES on product,quarter
- Sort SALES on store,quarter (also get gb store)
- Sort SALES on quarter



Cost of new plan
4*3M=12M I/Os
(45% savings)

Compute from “smallest parent” vs “sharing sort orders”

- Consider computation of gb product, quarter
- Previously: Sort SALES on product,quarter
- Alternative: read and sort previously computed gb product,store,quarter
  - This gb will be smaller than SALES
    - It may even fit in memory (one-pass sort)
  - This gb is *partially* sorted (common prefix)

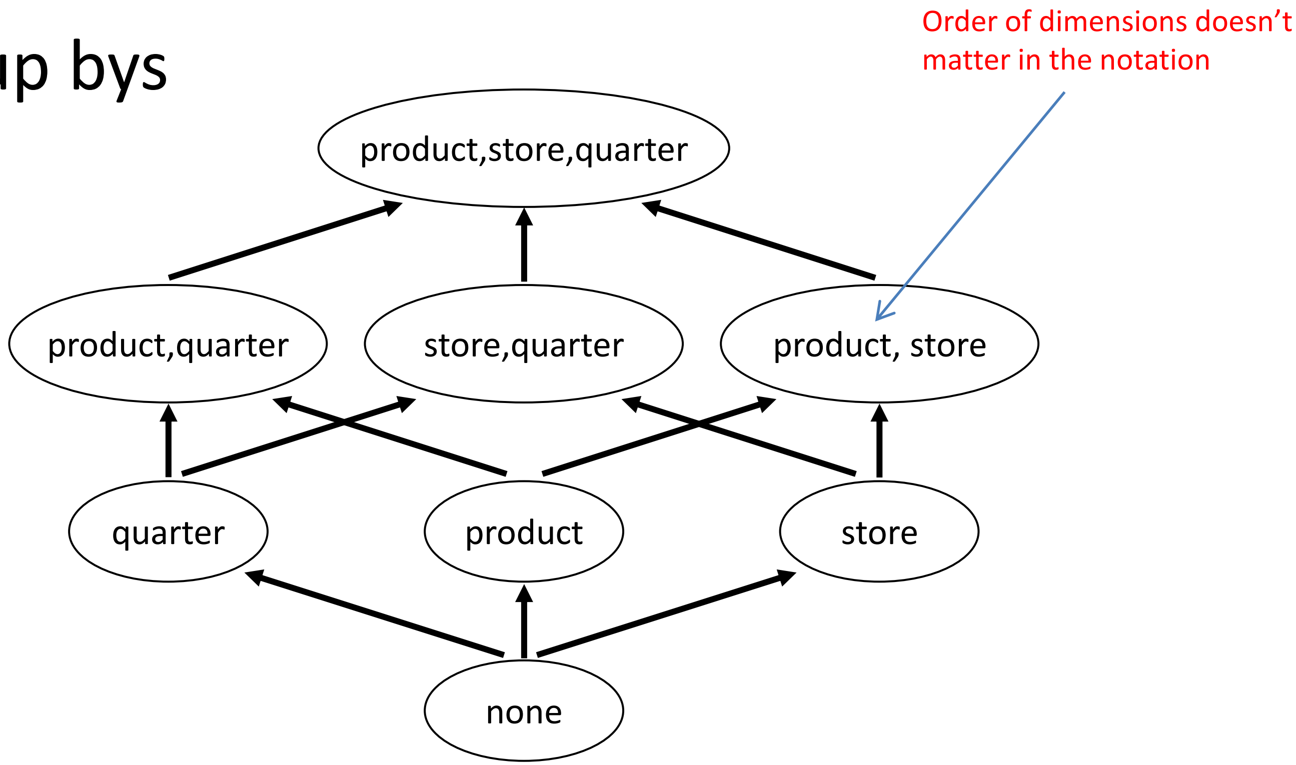


ESTIMATING THE DATA CUBE SIZE

How many group bys in the Data Cube?

- N-dimensional data, no hierarchies

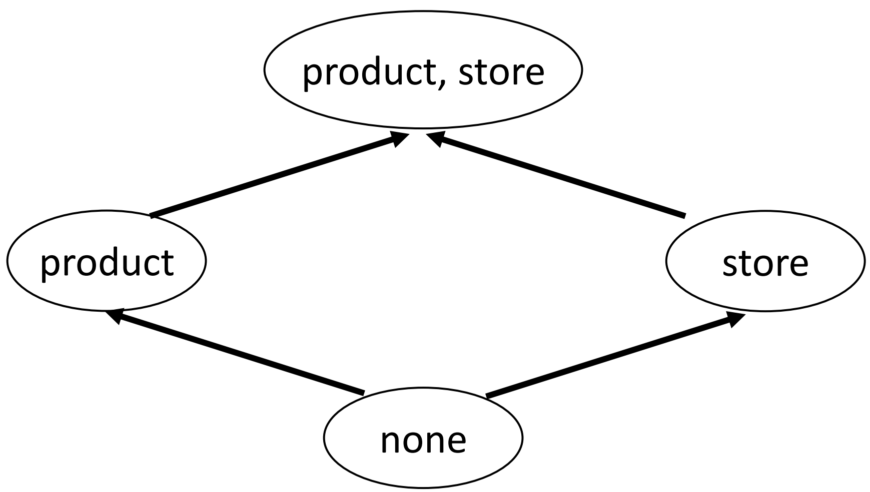
  2^N group bys



2D Data Cube lattice

- 2-dimensional data (product, store)

  $2^2 = 4$ group bys



Let’s add a simple hierarchy

- Assume that products are organized into **categories**
- When we group the sales (facts) we have the option to use this knowledge
  - Aggregate sales per category
  - Aggregate sales per category and store
  - But it does not make sense to aggregate sales per product and category (WHY?)

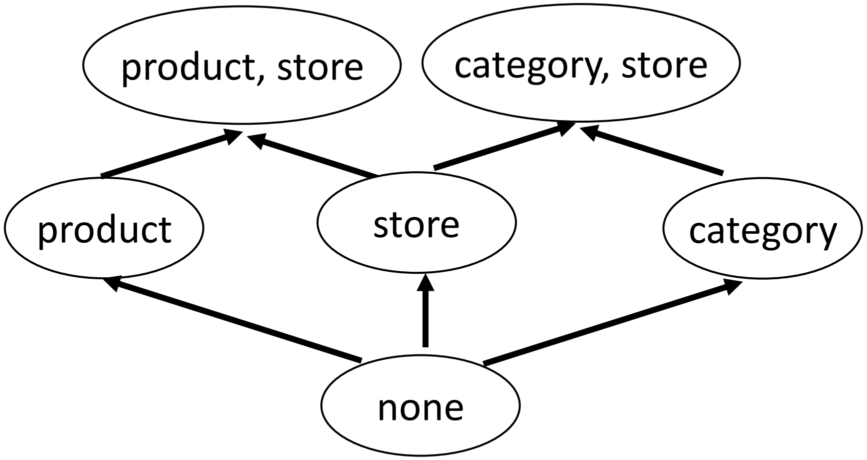
Compare these two results

product	category	sum(amt)
p1	cat1	110
p2	cat1	19
p3	cat3	240
p4	cat2	255
p5	cat1	75

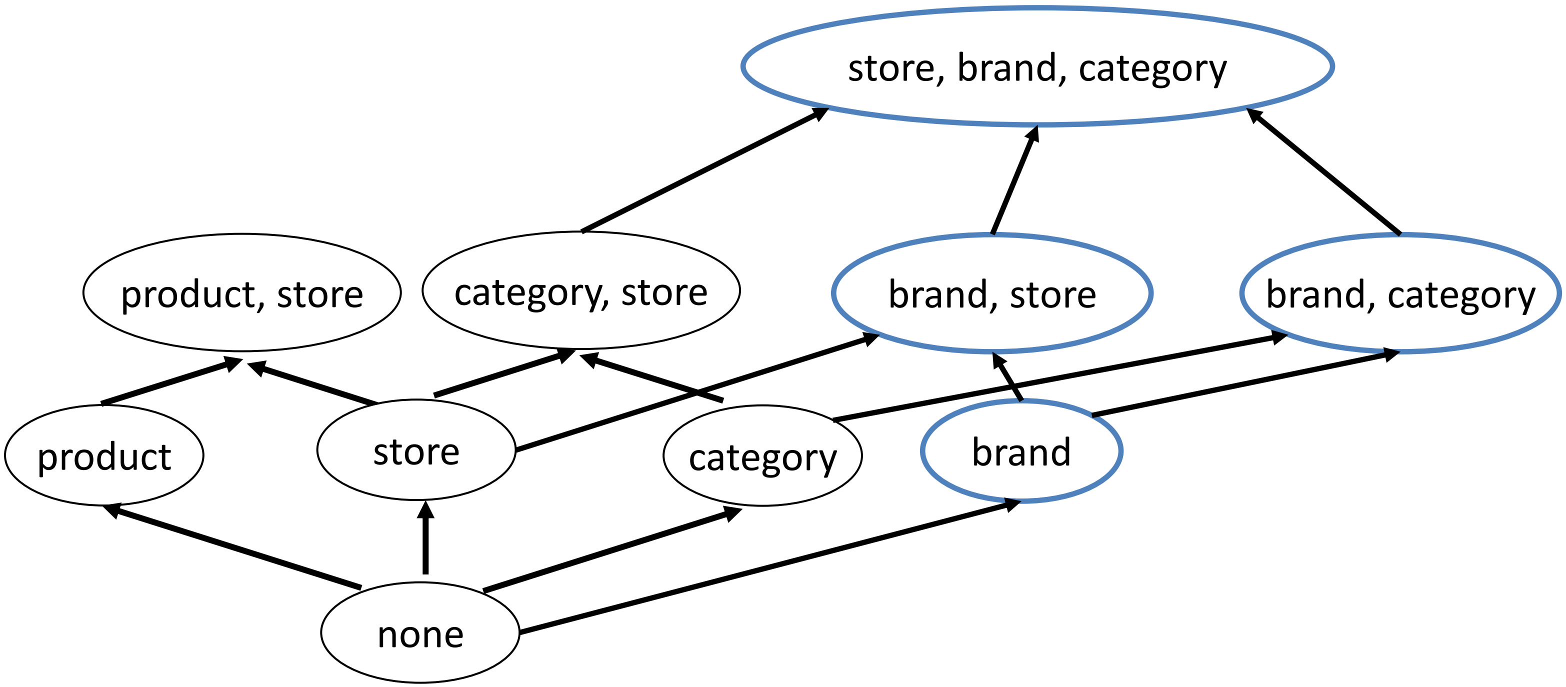
product	sum(amt)
p1	110
p2	19
p3	240
p4	255
p5	75

Notice that there is no difference in the computed aggregates, since prodId→category

2D Data Cube lattice with simple hierarchy



2D Data Cube lattice with 2 separate hierarchies on the product dimension



Notice lack of gb on (product,store,brand,category) ....

#of group bys when there is a single hierarchy per dimension

- N dimensions
- Dimension d_i has a hierachy of length L_i
- Location: store→city→country

  $L_{Location}$ =3

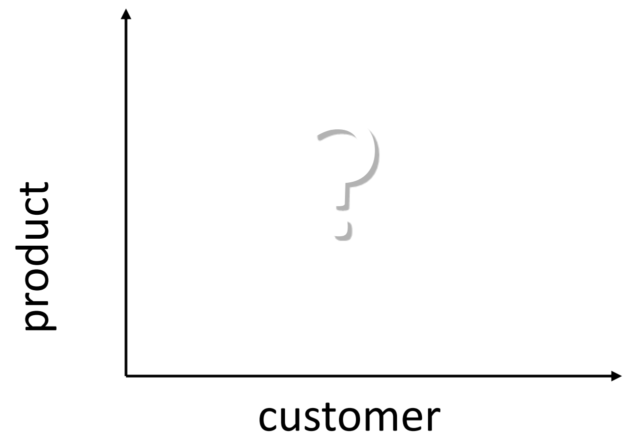
  – If no hierarchy, then L_i =1
- Number of group bys = $(1+L_1)\ (1+L_2)\ldots\ (1+L_N)$
  – No need to memorize formulas! Seek to understand their derivation instead (next slide)

How is the formula derived

- Consider Location dimension with hierarchy
  - store→city→country (i.e. $\mathbf{L_{Location} = 3}$)
- In a group by (aggregate) query I may
  - Not consider location at all (e.g. total sales per product)
    - Another way to think about this is that +1 stands for ALL
  - Consider location information at the store-level
    - (e.g. total sales per customer, store)
  - Consider location information at the city-level
    - (e.g. total sales per product, city)
  - Consider location information at the country-level
    - (e.g. total sales per sales_person, country)
- There are (1+3) choices regarding that dimension independently on what other dimensions I select in a gb
  - Thus, $(1+L_1)$ $(1+L_2)$… $(1+L_N)$ possible combinations of dimensions in a query

Example

- 8 dimensions (typical)
- 3-level hierarchy/dimension
- Number of group bys = 4^8=65536 group bys!
- BUT, how many tuples in the cube?
  - Depends on data distribution
  - Worst case is uniform



Upper bound on the size of each group by

- Assume relation R (fact table) has T(R) tuples
- Each dimension has cardinality t_i

- Size of group by (d_1, d_2,… d_k) is upper bounded by both
  - t_1* t_2 *..* t_k
  - T(R) (since records in the group by are produced by combination of attribute values that appear in existing facts)

Example gb(customer,product)

- Assume I have 1000 customers and 50 products
- Assume uniform distribution (customers buy products with same probability)
  - There can be 1000 x 50 combinations of pairs (customer, product) in the fact table (sales)
  - Thus, 50000 records in gb(customer,product) (at most)
- Each record in this gb is derived from a real sale
  - There can not be an aggregated record if there are not base records in the fact table to support it
- Thus, there can not be more records in the gb than the number of actual sales in the fact table

Example

- Consider R(product,store,quarter,amt) with 1M records
- 10,000 products, 30 stores, 4 quarters
  - Let G(x,y) denote the maximum number of records in group by x,y
  - G(product,store,quarter)=min(1M,10000*30*4)=1,000,000
  - G(product,store)=min(1M,10000*30)=300,000
  - G(product,quarter)=min(1M,10000*4)=40,000
  - G(store,quarter)=min(1M,30*4)=120
  - G(product)=min(1M,10000)=10,000
  - G(store)=min(1M,30)=30
  - G(quarter)=min(1M,4)=4
  - G(none)=1

  - Maximum cube size = 1,350,155 records

Quick and Dirty Upper Bound

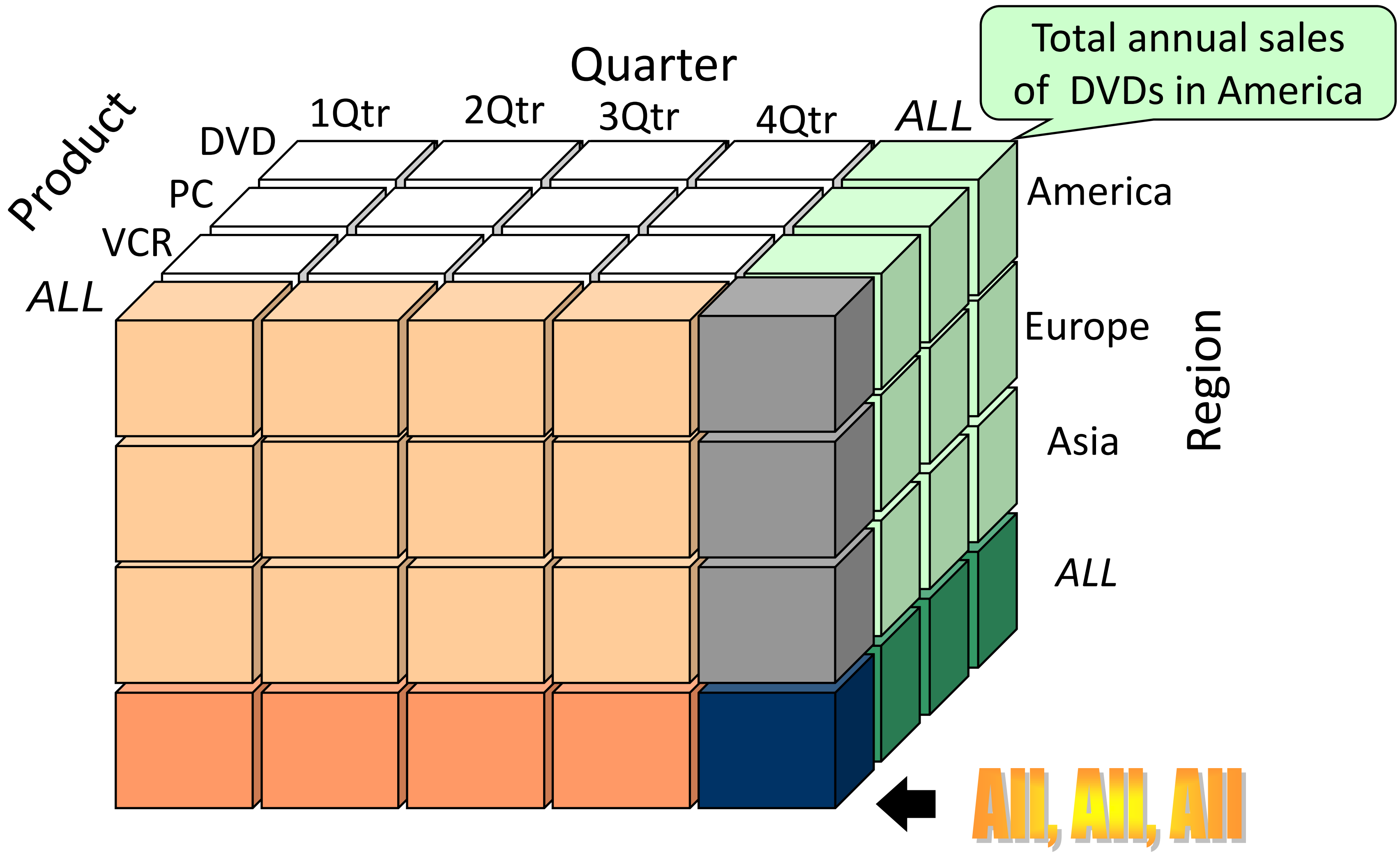
MAX-SIZE<=10001*31*5 = 1550155

$(1+t_1)*(1+t_2)*(1+t_3)$

(compare with 1350155)

This upper bound ignores size of fact table
WHY ??

Data Cube: Multidimensional View



Extended Cube with Hierarchies

- Products are organized in 50 categories
- Additional group bys in extended cube
  - +G(category,store,quarter)=min(1M,50*30*4)=6,000
  - +G(category,store)=min(1M,50*30)=1,500
  - +G(category,quarter)=min(1M,50*4)=200
  - +G(category)=min(1M,50)=50

  - Maximum ext-cube size = 1,357,905 records

Correlated Attributes

- In practice there is some correlation between different dimensions
- Example 1: each store sells up to 1,000 products (specialized stores)
- Example 2: some products are not sold through-out the year
  - Ice cream, watermelon, snow-chains

Solve Example-1

- R(product,store,customer) with 1M records
- 1,000 products, 20 stores, 100 customers
- Each customer buys from one store (closest)

FD: customer → store

G(store,customer)=min(1M,1*100)=100

G(product,store,customer)=min(1M,1000*1*100)
=100,000

More realistic example

- 100,000 parts
- 20,000 customers
- 2,000 suppliers
- 5 years (=365 *5 days)
- 100 stores
- 1,000 sales persons

- Max-cube size = 738,855,253,876,896,582,426 (tuples)

Catch With Data Cube

- …. too0000 many aggregates
- So Data Cube is large!
  - And takes time to compute…

What to Materialize?

- Data Cube extremely large for many applications
- Store in warehouse results useful for common queries
- Example:
  - Total sales per product, store
  - Max sales per product
  - Avg sales per store,day
  - …

Materialization Factors

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

MATERIALIZED VIEWS

Preliminaries

- We will consider solutions that selectively materialize some of the groups by in the Data Cube
- We will be referring to the group bys as "views"
- When a group by is materialized we will call it "materialized view"

Views in OLTP databases

Employee(ename, age, dept, address, telno, salary)

- Views are **derived** tables
  - Instance of view is generated **on demand** by executing the view query:

    create view V as

    select ename,age, address,telno

    from employee

    where employee.dept = “Sales”
- Views have many uses
  - Shortcuts for complex queries
  - Logical-physical independence
  - Hide details from the end-user
  - Integration systems

Materialized Views (OLAP)

- Sometimes, we may want to compute and store the content of the view in the database
  - Such Views are called materialized
  - Queries on the materialized view instance will be much faster
  - Materialized views are now supported by some vendors
    - Otherwise we will be storing their data in regular tables
- This is our extended architecture:

Data Warehouse=

$\textbf{detailed records (star schema)} + \underbrace{\textbf{aggregates (materialized views)}}_{\text{Used to speed up certain queries of interest}}$

Materialized views in OLAP

- Contain derived data
  - Can be computed from the star schema
- Populated while updating the data warehouse
  - Usually they contain results of complex aggregate queries
- Several interesting problems:
  - How to select which views to materialize?
  - How to compute/refresh these views?
  - How to store these views in the relational schema?
  - How to use these views at query time?

View selection problem

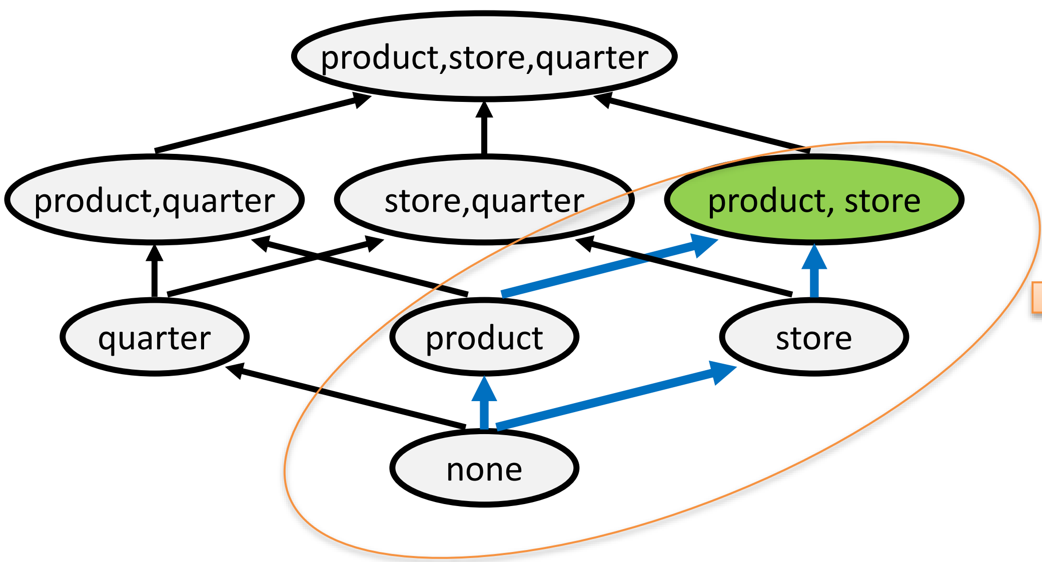
- Set up as an optimization problem
  - V_{DC} = set of all group bys (=views) in the Data Cube
  - Give a constraint
    - Usually space bound B, e.g. materialize up to 100GB from the CUBE
    - What else?
  - Give an objective
    - Minimize cost of answering set of (frequent/interesting) queries Q
- View selection problem (with space constraint):

$$\boxed{\begin{array}{l} \underset{V \subseteq V_{DC}}{minimize} \quad Cost(Q) \\ \\ \text{such that } \mathrm{Size}(V) \leq B \end{array}}$$

- Problem is NP-hard

View Selection Problem: Heuristic

- Use some notion of *benefit* per view based on the dependencies depicted in the Data Cube lattice



group by(product,store)

product	store	sum(amt)
p1	s1	56
p1	s2	4
p1	s3	50
p2	s1	11
p2	s2	8

Queries related to these group bys can be computed from a materialized view on group by (product,store), independently of the method of computation (sort, hash, etc)

A simple greedy algorithm

- Utilize a benefit criterion
  - Assume V is the views we have chosen so far
  - Let v be a candidate view not in V
  - Benefit(v) = cost of answering queries using V – cost of answering queries using V U {v}
    - Measures the reduction in query answering cost if this view is materialized
    - Benefit(v) ≥ 0
- Greedy algorithm
  - At each step, pick the view that has the maximum benefit
    - Re-compute benefits of remaining views
    - Update B=B-sizeof(v)
    - Remove views that do not fit in new B
  - Stop if no more space available or no view fits in the remaining space

Simple Example

- Star schema with three dimensions
  - Product (p), Store location (s), Quarter (q)
- Assume the following queries Q = {(p,s),(s,q), (p,q), (p),(s)}
  - Notation: (s,q) is a query on group by (store,quarter)

(s,q): SELECT store, quarter, sum(amt)
FROM SALES
GROUP BY store, quarter

Query computation cost

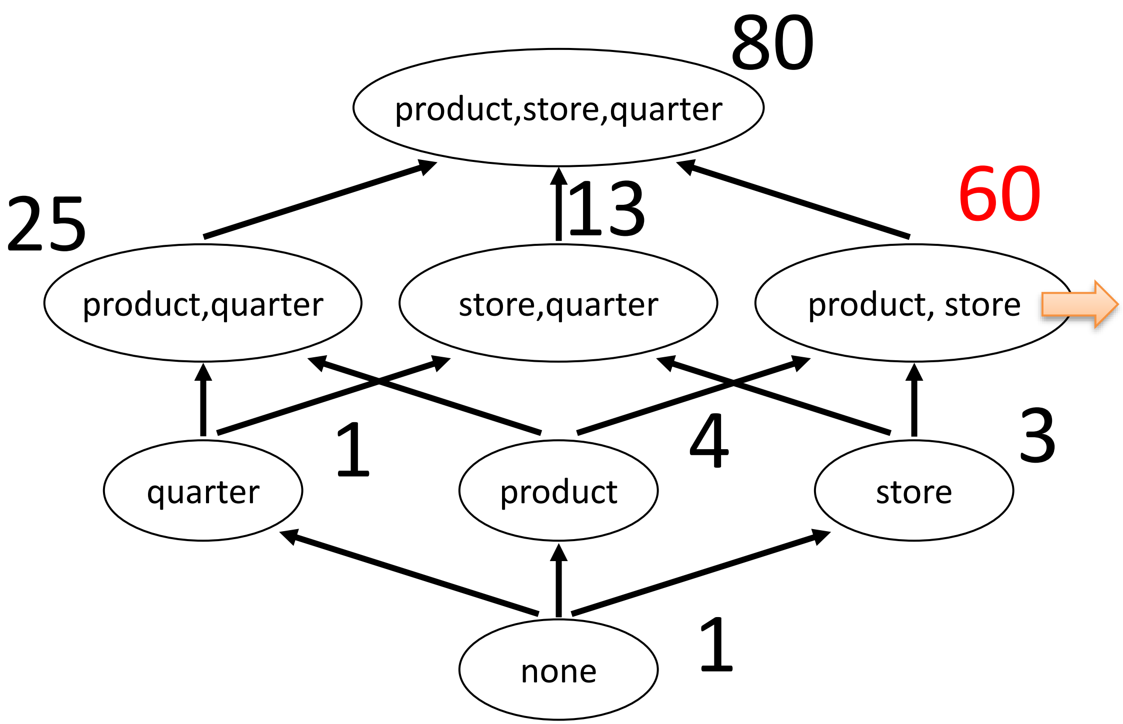
- For ease of presentation, let us assume that each query can be computed from the fact table SALES with the same cost 100 I/O

(s,q): SELECT store, quarter, sum(amt)
FROM SALES
GROUP BY store, quarter

Cost = 100 I/O

Data Cube sizes

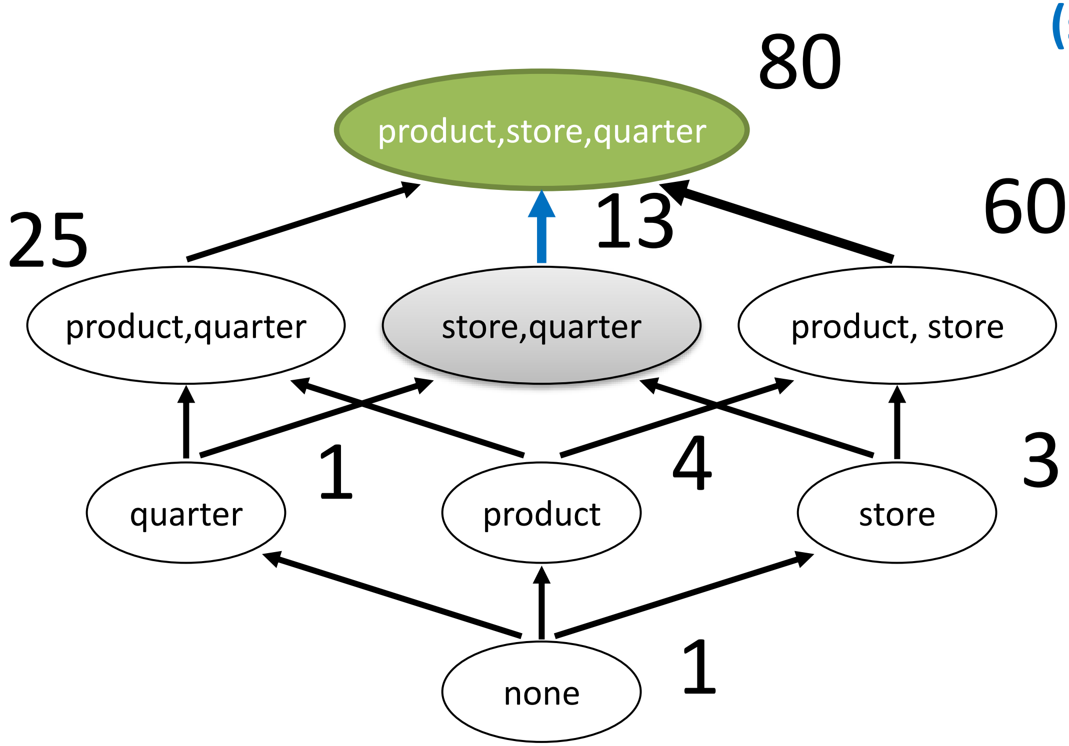
- Assume each group by in the Data Cube requires the depicted number of blocks, when stored as a materialized view



Size $V_{product,store}$=60 blocks

Assumption (for this simple example)

- A group by query is computable from an ancestor materialized view V with Cost=size(V)



Computation for (s,q) from Sales:

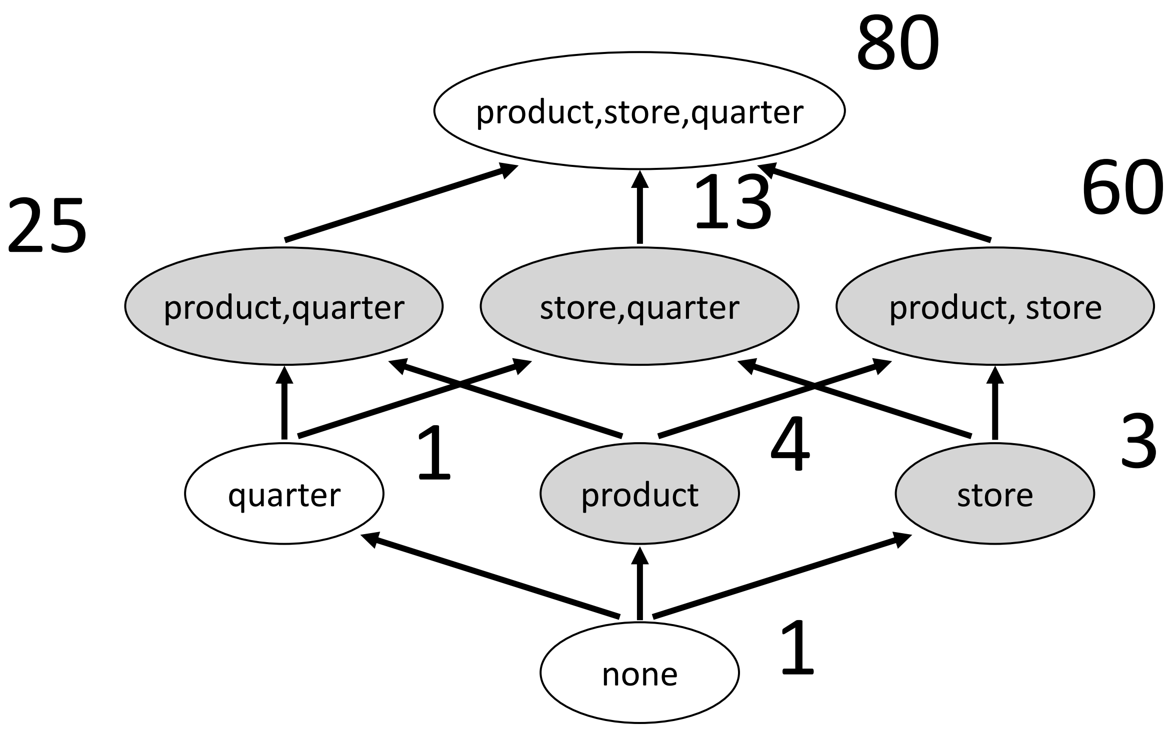
(s,q): SELECT store, quarter, sum(amt)
FROM SALES
GROUP BY store, quarter
Cost = 100 I/O

Alternative computation for (s,q):

SELECT store, quarter, sum(amt)
FROM $V_{product,store,quarter}$
GROUP BY store, quarter
Cost = 80 I/O

View Selection Problem

- Minimize the cost of answering the depicted queries when available space B=100



Initial Benefits
(no view is materialized yet)

Group By (Materialized View)	Benefit for Q={(p,s),(s,q), (p,q), (p),(s)}
p,s,q	(100-80)+(100-80)+(100-80)+(100-80)+(100-80)=100
p,q	2*(100-25)=150
s,q	2*(100-13)=174
p,s	3*(100-60)=120
p	100-4=96
s	100-3=97
q	0
None	0

Step-1

- Materialize view $V_{s,q}$
- Update space budget B = 100-13 = 87
- Recompute benefits (next slide)

Space=87

V={(s,q)}

Updated Benefits

Group By (Materialized View)	Benefit for Q={(p,s),(s,q), (p,q), (p),(s)}
p,s,q	3*(100-80)=60
p,q	(100-25)+(100-25)=150
s,q	MATERIALIZED
p,s	2*(100-60)+0=80 (careful)
p	100-4=96
s	13-3=10 (careful)
q	0
None	0

Step-2

- Materialize view $V_{p,q}$
- Update space budget B = 87-25 = 62
- Update benefits (next slide)

Updated Benefits

Space=62

V={(s,q),(p,q)}

Group By (Materialized View)	Benefit for Q={(p,s),(s,q), (p,q), (p),(s)}
p,s,q	Not-enough-space-left
p,q	MATERIALIZED
s,q	MATERIALIZED
p,s	(100-60)=40 (careful)
p	25-4 =21 (careful)
s	13-3=10 (careful)
q	0
None	0

Step-3

- Materialize view $V_{p,s}$
- Update space budget B = 62-60 = 2
- Update benefits

Updated Benefits

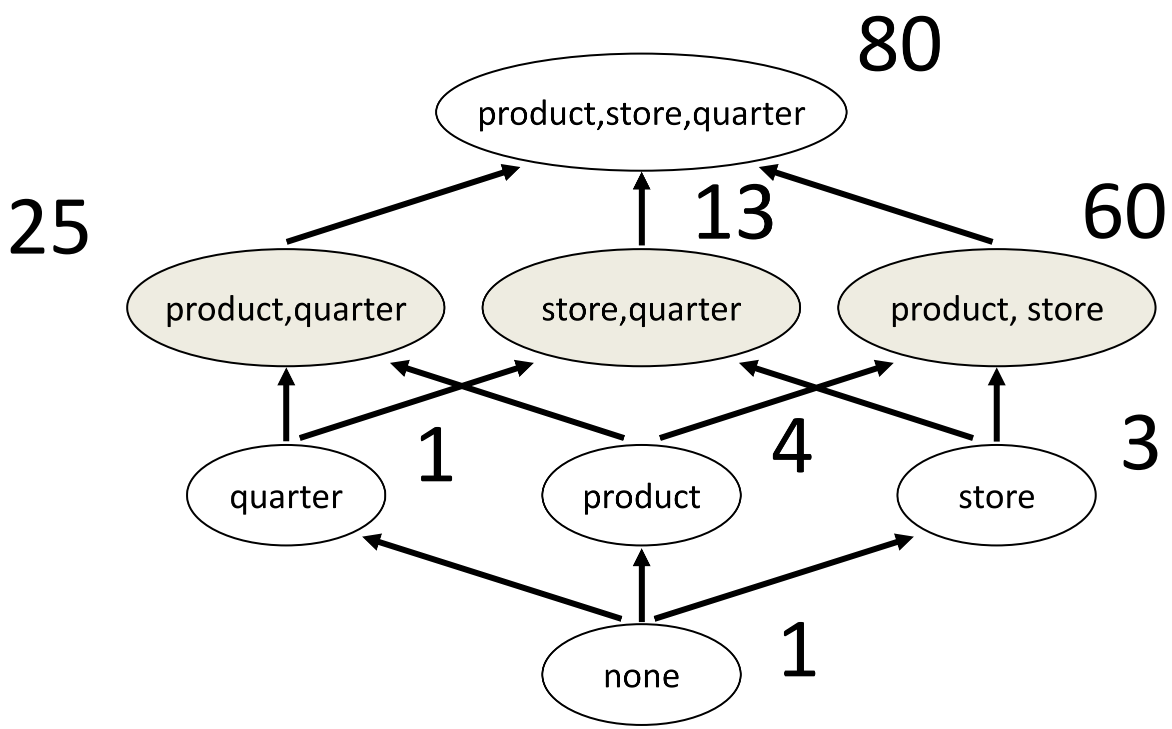
Space=2

V={(s,q),(p,q),(p,s)}

Group By (Materialized View)	Benefit for Q={(p,s),(s,q), (p,q), (p),(s)}
p,s,q	Not-enough-space-left
p,q	MATERIALIZED
s,q	MATERIALIZED
p,s	MATERIALIZED
p	Not-enough-space-left
s	Not-enough-space-left
q	0
None	0

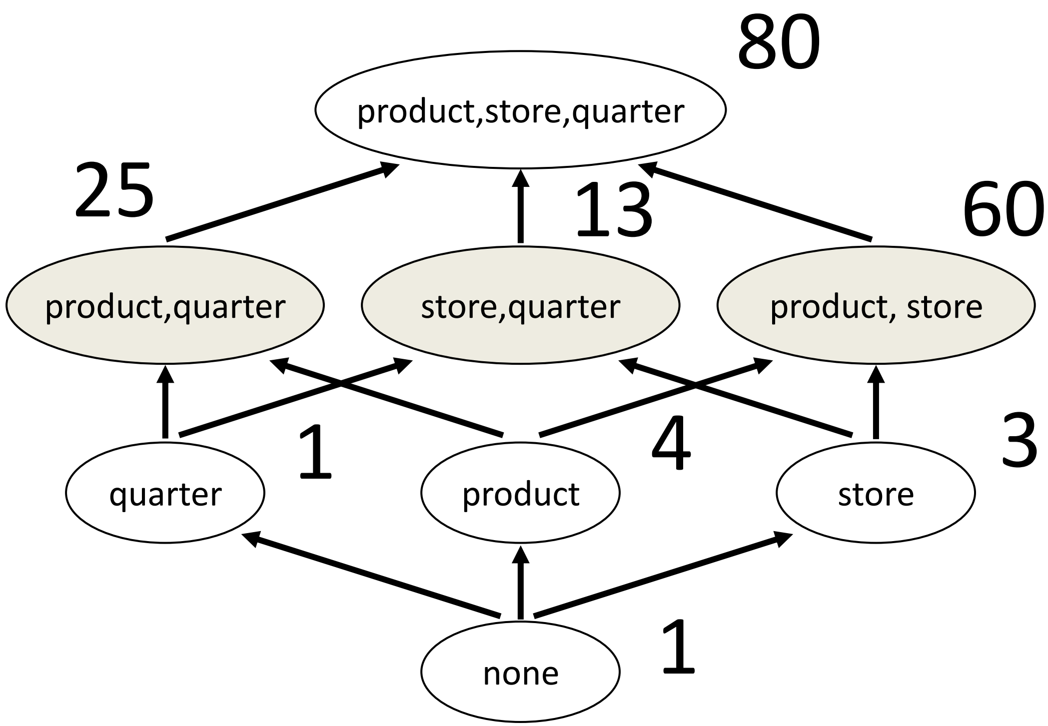
Greedy algorithm selection

- Final choice V={(s,q),(p,q),(p,s)}
  - Utilize 25+13+69=98 blocks out of 100 available



Query costs for this selection

- Q = {(p,s),(s,q), (p,q), (p),(s)}
  - Cost(p,s) = 60
  - Cost(s,q) = 13
  - Cost(p,q) = 25
  - Cost(p) = 25
  - Cost(s) = ?



Benefit of using Materialized Views

Q = {(p,s),(s,q), (p,q), (p),(s)}

Using the suggested Materialized Views

Cost(p,s) = 60

Cost(s,q) = 13

Cost(p,q) = 25

Cost(p) = 25

Cost(s) = 13

Total Query Cost = 136

Querying the Fact Table

Cost(p,s) = 100

Cost(s,q) = 100

Cost(p,q) = 100

Cost(p) = 100

Cost(s) = 100

Total QueryCost = 500

The View Update problem

Materialized View: Vsc

Store	Customer	*Price*
S1	C2	$700
S1	C3	$240
S2	C1	$190
S2	C3	$450

How to update this view?

Table Deltas:
(new records to be appended in the fact table)

New sale:

Store	Customer	Product	*Price*
S1	C2	P2	$55
S1	C2	P3	$15
S1	C1	P1	$50
S2	C1	P3	$20

Choice 1:
Re-compute from fact table

- First update fact table (append new facts)
- Then re-execute SQL query to obtain view

In SQL:

```
//load new records
insert into Fact select * from Delta
//drop and recreate View
drop Vsc;
create table Vsc(store,customer,price);
//recompute View from scratch
insert into Vsc
  select store,customer,sum(price)
  from Fact
  group by store,customer;
```

Choice-2: Incremental Updates

- Adding delta tuples means
  - Step 1: Update sum() from combinations already in the view
  - Step 2: Insert sum() with new coordinates for rest

Store	**Customer**	***Price***
S1	C2	$700
S1	C3	$240
S2	C1	$190
S2	C3	$450

Store	**Customer**	**Product**	***Price***
S1	C2	P2	$55
S1	C2	P3	$15
S1	C1	P1	$50
S2	C1	P3	$20

Step 1: Increment existing combinations

```
update Vsc
set Vsc.m=Vsc.m+(select sum(price) from Delta
   where Vsc.store=Delta.store and
   Vsc.customer=Delta.customer)
where (Vsc.store,Vsc.customer)
        in
        (select store,customer from Delta);
```

Step 2: Add new combinations

```
insert into Vsc
   select store,customer,sum(price)
   from Delta where (store,customer) not in
         (select store,customer from Vsc)
   group by store,customer;
```

Choice-2: Alternative

- Idea: add delta records to the view, create a new table to hold updated records, then rename

```
insert into Vsc
  select store,customer,sum(price) from Delta
  group by store,customer;
create table Vnew(store,customer,price);
insert into Vnew
  select store,customer,sum(price) from Vsc
  group by store,customer
drop table Vsc;
rename table Vnew to Vsc;
```

Simple Example

After insertion of deltas

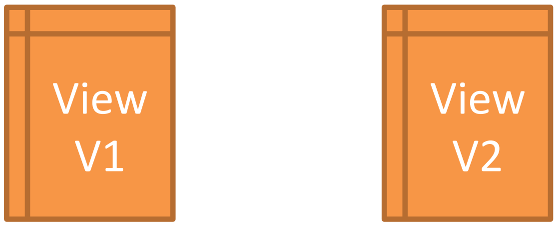
Store	Customer	*Price*
S1	C2	$700
S1	C3	$240
S2	C1	$190
S2	C3	$450
S1	C1	$50
S1	C2	$70
S2	C1	$20

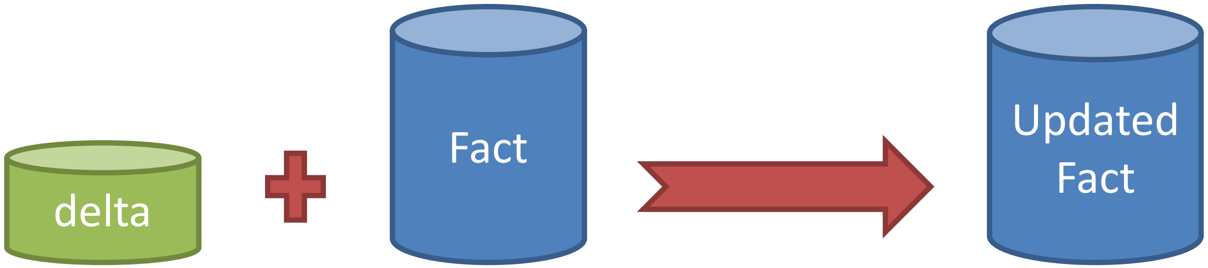
Final View

Store	Customer	*Price*
S1	C1	$50
S1	C2	$770
S1	C3	$240
S2	C1	$210
S2	C3	$450

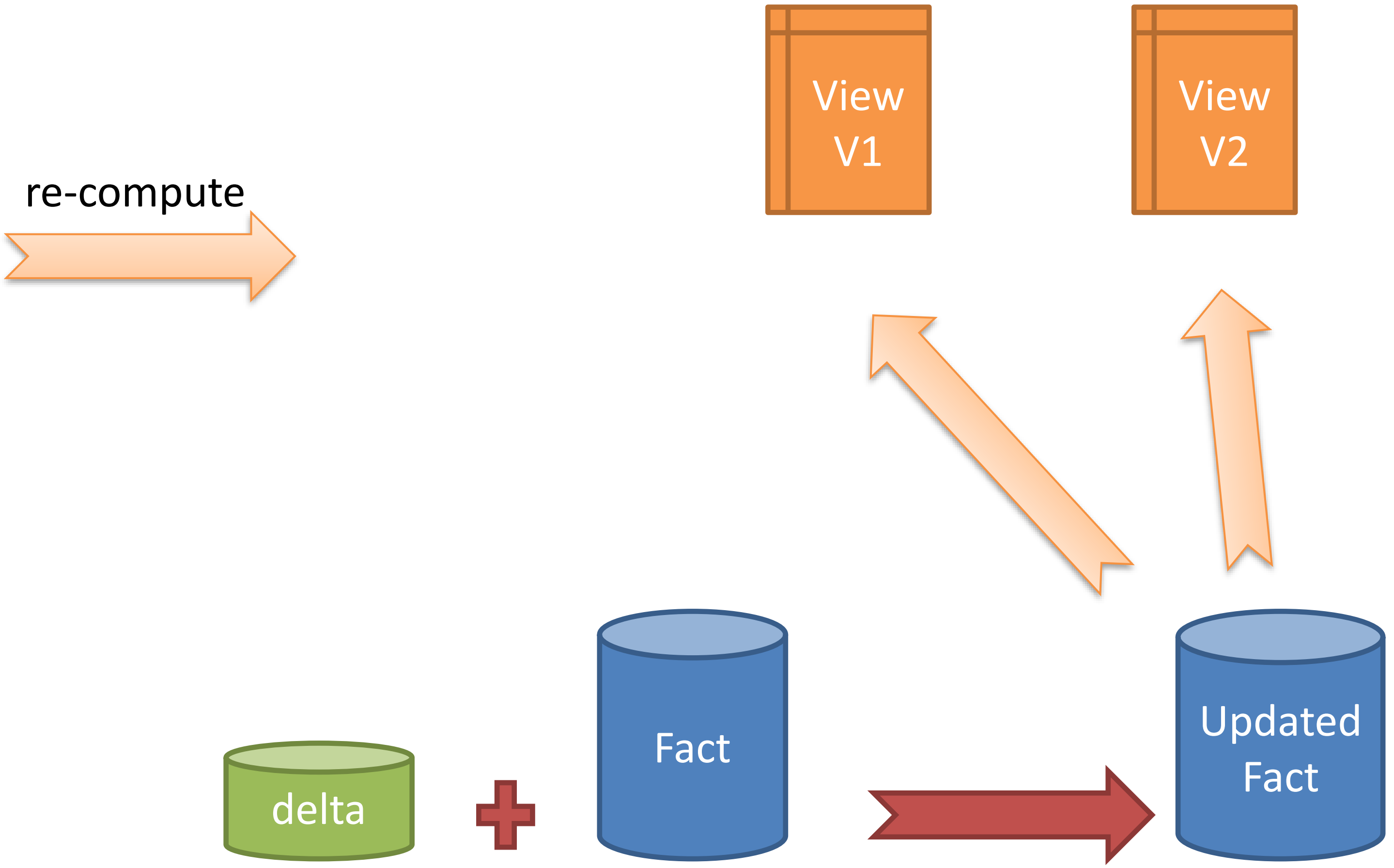
Multiple View Update

Assume V2 descendant of V1 in the Data Cube Lattice (e.g. V1 can be used to compute V2)

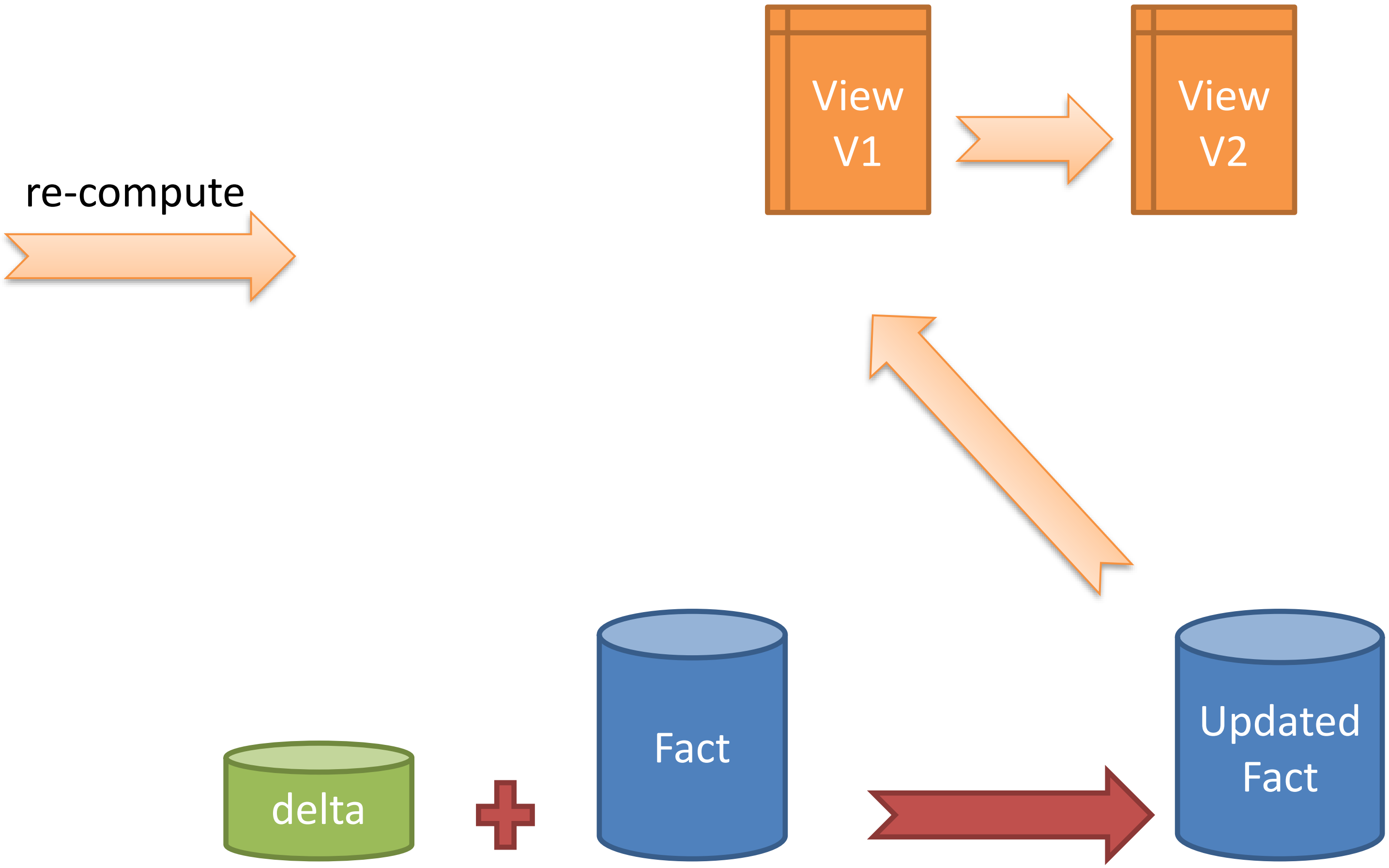




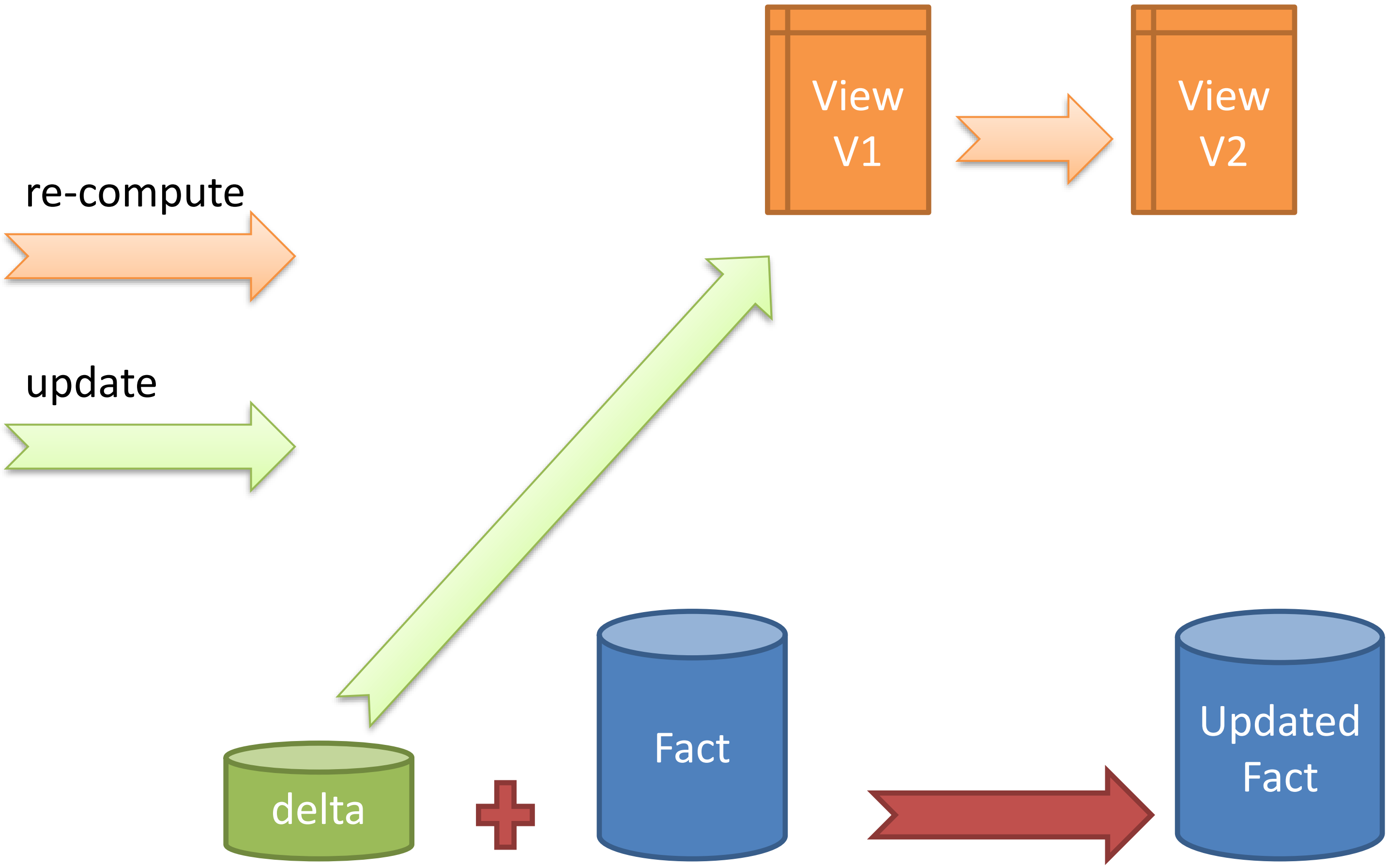
Scenario 1: Re-compute views after finishing updating the Fact table



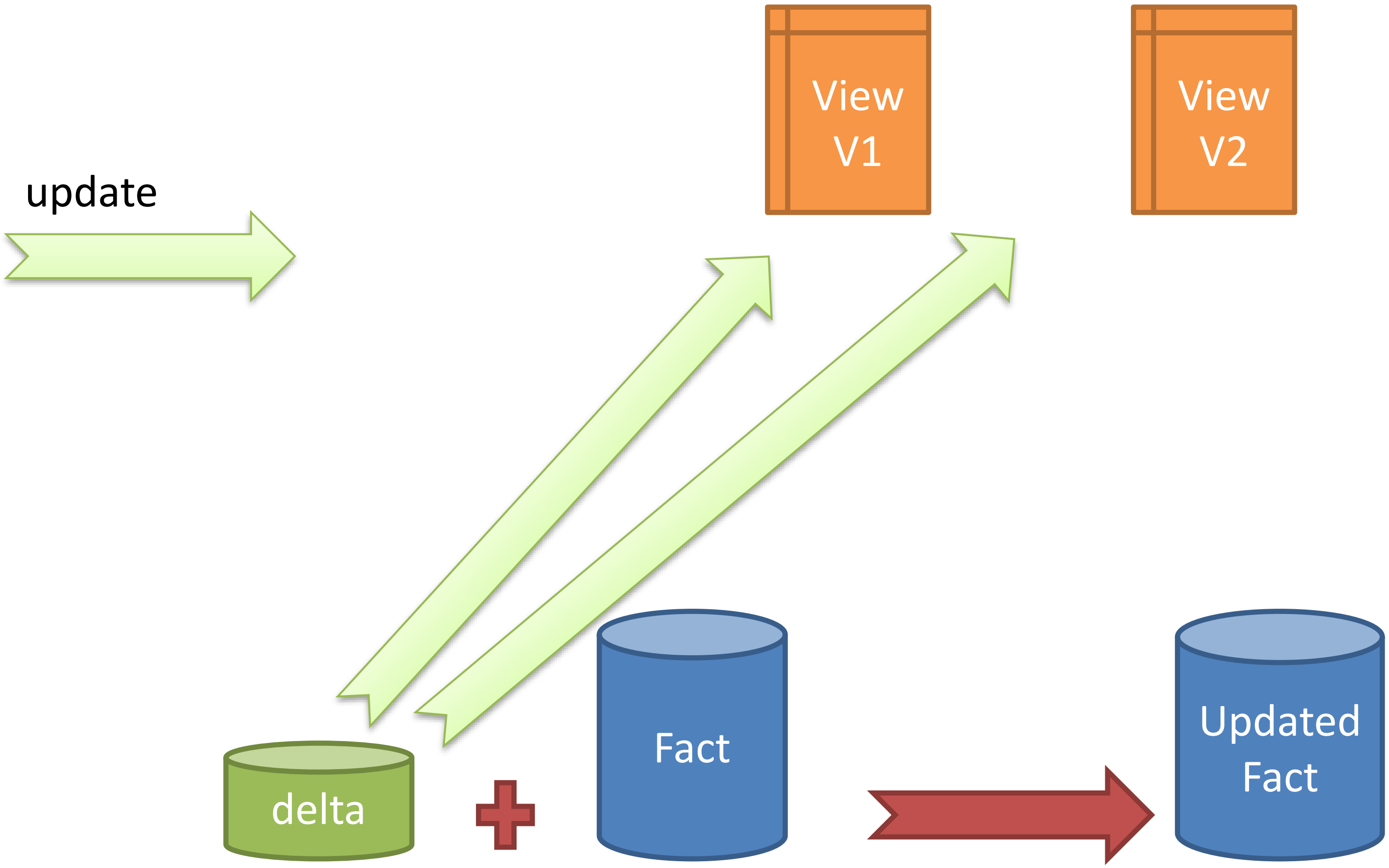
Scenario 2: Re-compute V1 from Fact, V2 from V1



Scenario 3: Incrementally update V1 from delta then recompute V2 from V1



Scenario 4: Incrementally update both V1 and V2 from delta



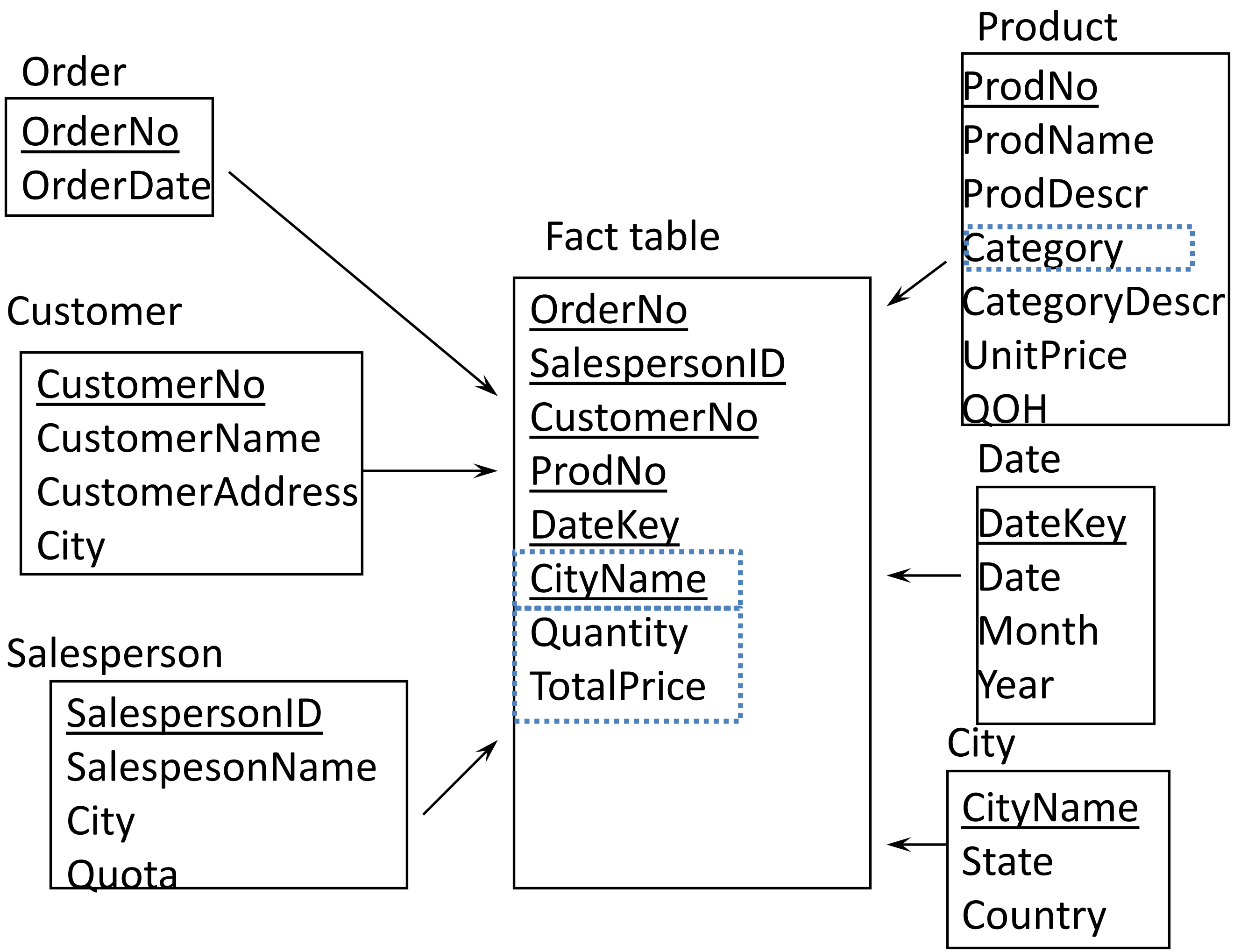
Consider

- More scenarios?

- Now consider the case of 100 views

PHYSICAL REPRESENTATION OF MATERIALIZED VIEWS IN THE STAR SCHEMA

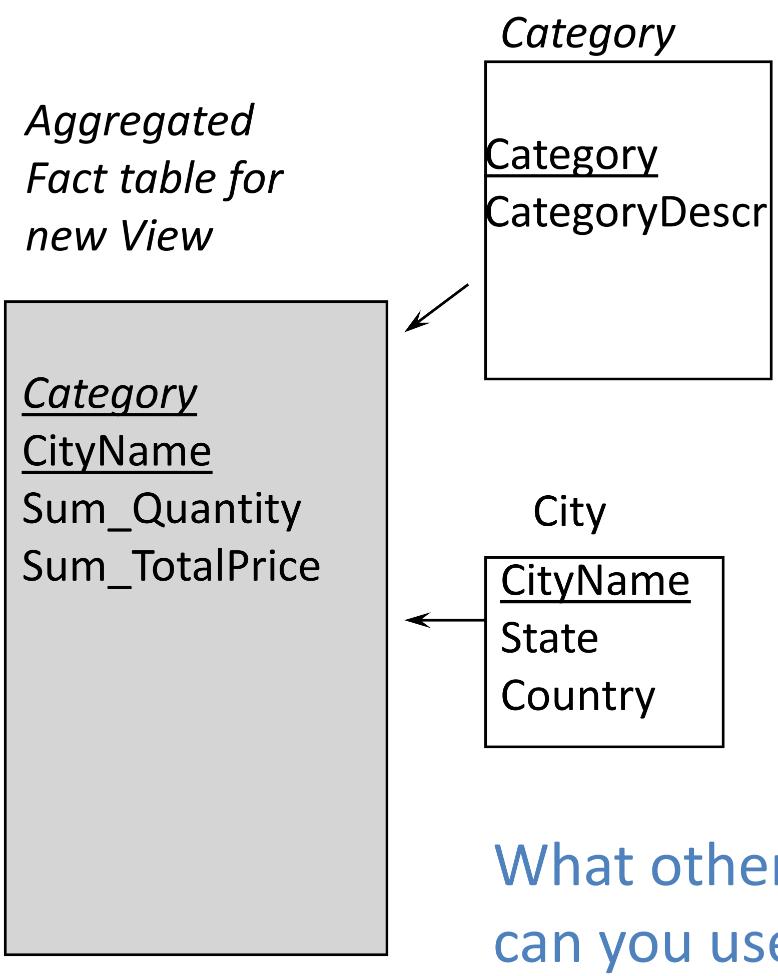
Want to create View:
SUM(Quantity), SUM(TotalPrice) per Category, CityName



SQL Επερώτηση

```
Select Category,CityName,SUM(TotalPrice) as Sum_TotalPrice,SUM(Quantity) as
Sum_Quantity
From Fact,Product
Where Fact.ProdNo=Product.ProdNo
Group by Category,CityName
```

Create New Fact Table (= this view)



What other queries
can you use this view for?

Using Materialized Views through Selection

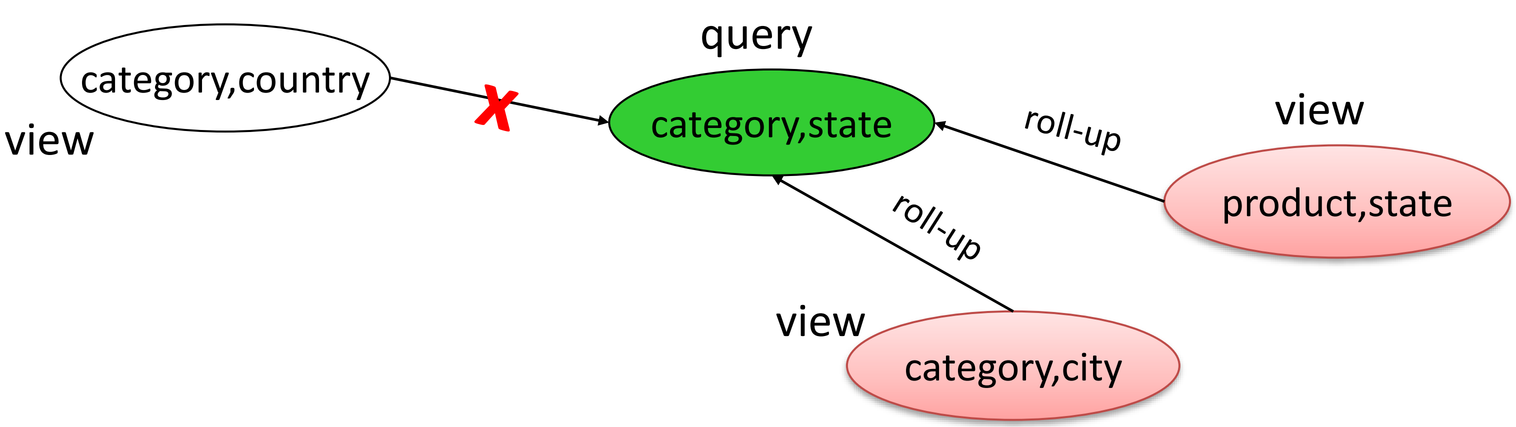
- A query can use a view through a selection if
  - Each selection condition C on each dimension d in the query logically implies a condition C' on dimension d in the view
- Example: A view has sum(sales) by product and by year for products introduced after 1991
  - OK to use for sum(sales) by product for products introduced after 1992
  - CANNOT use for sum(sales) for products introduced after 1989

Using Materialized Views through Group By (Roll Up)

- The view V may be applicable via roll-up if for every grouping attribute g of the query Q:
  - Q has *Group By a1,..,g, an*
  - V has *Group By a1,..,h, an*
  - Attribute *g is higher than h* in the attribute hierarchy
  - Aggregation functions are distributive (sum, count, max, etc)
- Example: Compute "sum(sales) by category" from the view "sum(sales) by product"

Using Views

- Need cost-based optimization to decide which view(s) to use for answering a query
  - Consider a query on (category, state) and three materialized aggregate views on
    1. (product, state)
    2. (category, city)
    3. (category, country)
  - (product, state) and (category, city) are *candidate materialized views* to answer the query



Σημείωση

- Τα παρακάτω slides είναι εκτός ύλης για το μάθημα του Σχεδιασμού Βάσεων Δεδομένων

Data Cube Storage and Indexing

- Several approaches within the relational world
  - Cubetrees, QC-trees, Dwarf, CURE
- Main idea: exploit inherent redundancy of multidimensional aggregates

The Dwarf (sigmod 2002)

- Data-Driven DAG
  - Factors out inter-view redundancies
  - 100% accurate (no approximation)
  - All views are included
  - Indexes for free
  - Partial materialization possible
- Look at the Data Cube Records
  - Common Prefixes
    - high in dense areas
  - Common Suffixes
    - extremely high in sparse areas

Redundancy in the Cube (1)

- Common Prefixes

S2,C1,P1,90
S2,C1,P2,50
S2,C1,ALL,140

Mostly in dense areas:

- customer C1 buys a lot of products at store S2
- all these records have the same prefix: S2,C1

Store	**Customer**	**Product**	***Price***
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Redundancy in the Cube (2)

- Common Suffices

S2,C1,P1,90

S2,ALL,P1,90

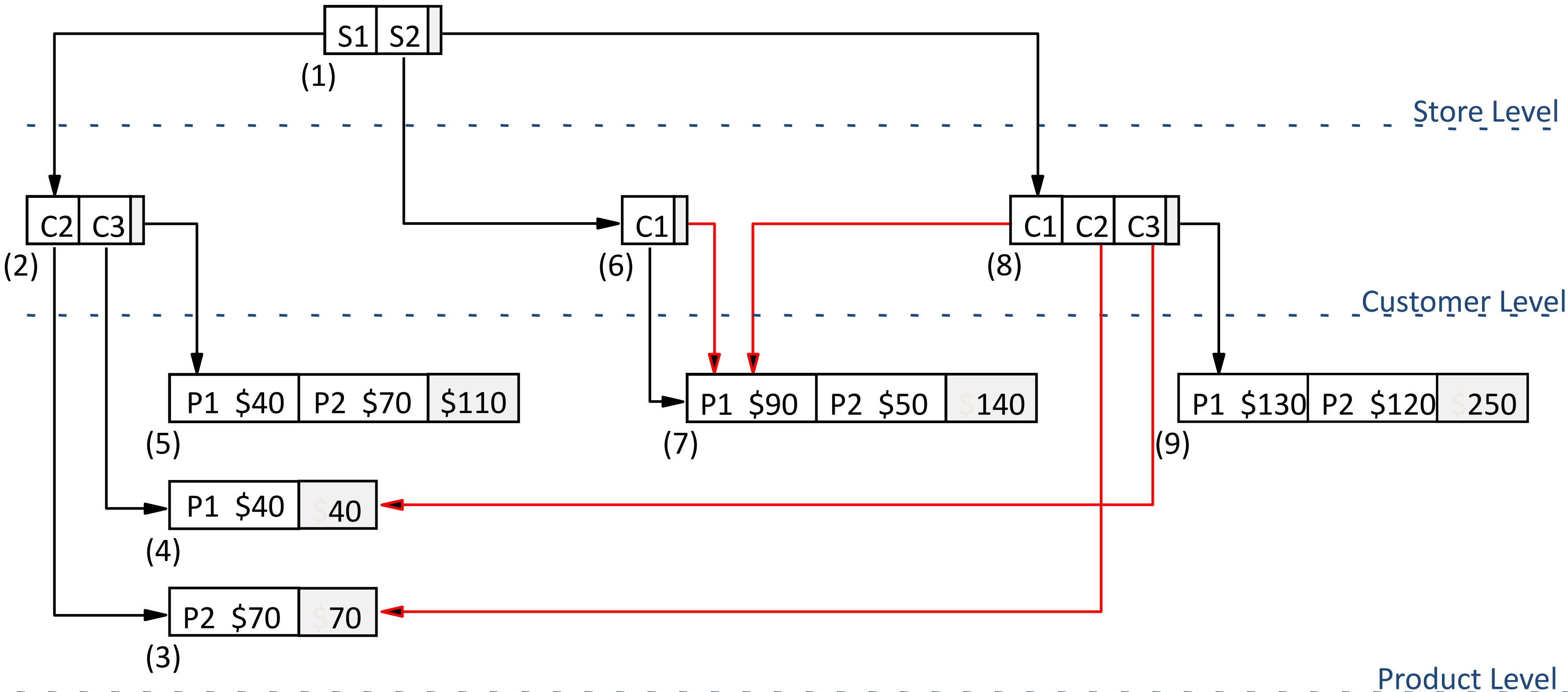
ALL,C1,P1,90

Mostly in sparse areas

C1 only visits S2 and is the only customer that buys P1,P2

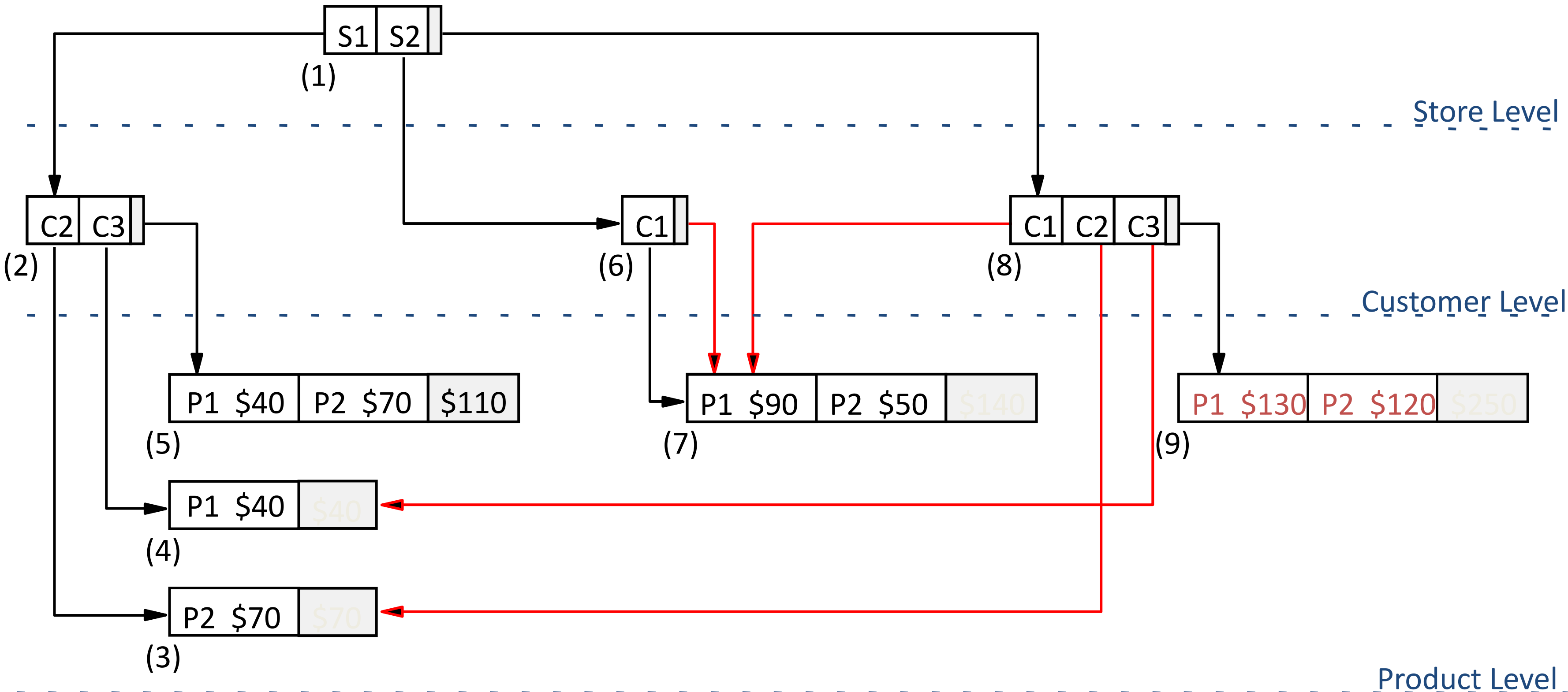
Store	**Customer**	**Product**	***Price***
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Dwarf Example



Store	Customer	Product	*Price*
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Dwarf Example

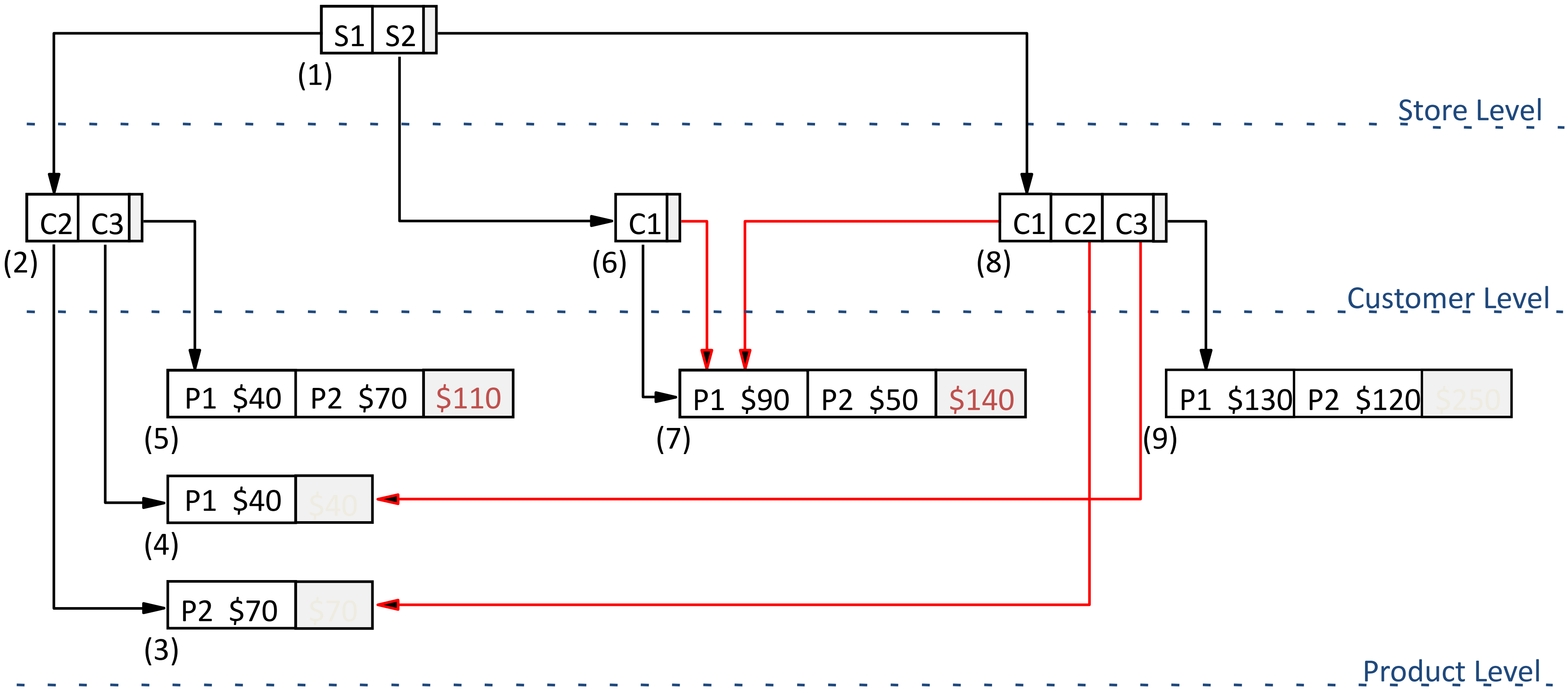


Store	Customer	Product	*Price*
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Group-by Product:

Store	Customer	Product	*Sum(Price)*
ALL	ALL	P1	$130
ALL	ALL	P2	$120

Dwarf Example



Store	Customer	Product	*Price*
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Group-by Store:

Store	Customer	Product	*Sum(Price)*
S1	ALL	ALL	$110
S2	ALL	ALL	$140