Some Reminders for a Seamless Online Class…

- Please turn on your video
- Mute yourself (press and hold spacebar when you'd like to talk)
- Don't do anything you wouldn't do in an in-person class
- I will occasionally check the chat for messages if you'd like to share there instead
- Please say your name before you speak

Recap

- Data-savviness is the future!
- "Classical" relational databases
  - Notion of a DBMS
  - The relational data model and algebra: bags and sets
  - SQL Queries, Modifications, DDL
  - Database Design
  - Views, constraints, triggers, and indexes
  - Query processing & optimization
  - Transactions
- Non-classical data systems
  - Semi-structured data and document stores
  - Unstructured data and search engines
  - Cell-structured data and spreadsheets
  - Dataframes and dataframe systems
  - **OLAP, summarization, and visual analytics**

Let's start with OLAP

- OLAP = OnLine Analytical Processing
- Also known as *decision support* or *business intelligence* (BI)
    - But now BI has grown to include more (e.g., ML)

- What is OLAP?
    - A specialization of relational databases that prioritizes the reading and summarizing large volumes (TB, PB) of relational data to understand high-level trends and patterns
        - E.g., the total sales figures of each type of Honda car over time for each county
    - "Read-only" queries
- Contrast this to OLTP: OnLine Transaction Processing
    - "Read-write" queries
    - Usually touch a small amount of data, e.g., append a new car sale into the sales table

Where and How is OLAP Performed?

- Usually, OLAP is performed on a separate *data warehouse* database away from the critical path of OLTP transactions (a *live or transactional* database).
- This data warehouse is periodically updated from data from various sources (e.g., once an hour or once a day)
  - For example, sales data from various regional sales databases gets periodically consolidated in a central data warehouse
  - This is through a process of ETL
    - Extract, Transform, Load
    - Extract useful business that needs to be summarized, transform it (e.g., canonicalize values, clean it up), load it in the data warehouse
  - By doing it periodically, this data warehouse can become stale

Where and How is OLAP Performed?

- Usually, OLAP is performed on a separate *data warehouse* database away from the critical path of OLTP transactions (a *live or transactional* database).
- Why?
  - Because OLAP queries end up reading most of the data, and will prevent OLTP queries from taking precedence
    - it is more important to ensure that sales are not prevented than to make sure that a report for a manager is generated promptly
    - the latter will anyway take a long time, so might as well have them wait a bit longer
  - It's OK if the warehouse data is a bit stale.

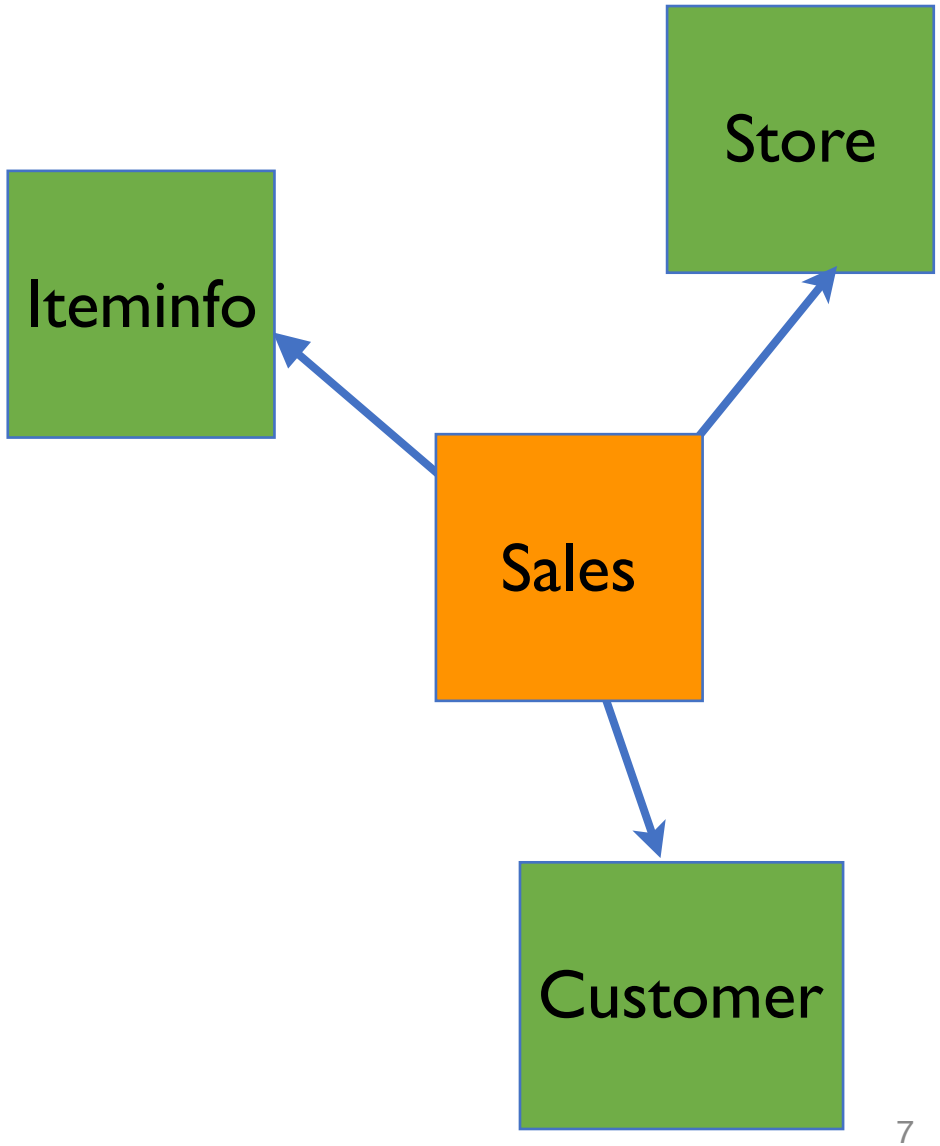
Schemas in Data Warehouses

- Known as a *star* (or sometimes *snowflake*) schema
- One *fact table* and many *dimension tables*
- Fact tables contain *dimension attributes* and *measure attributes*

- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)

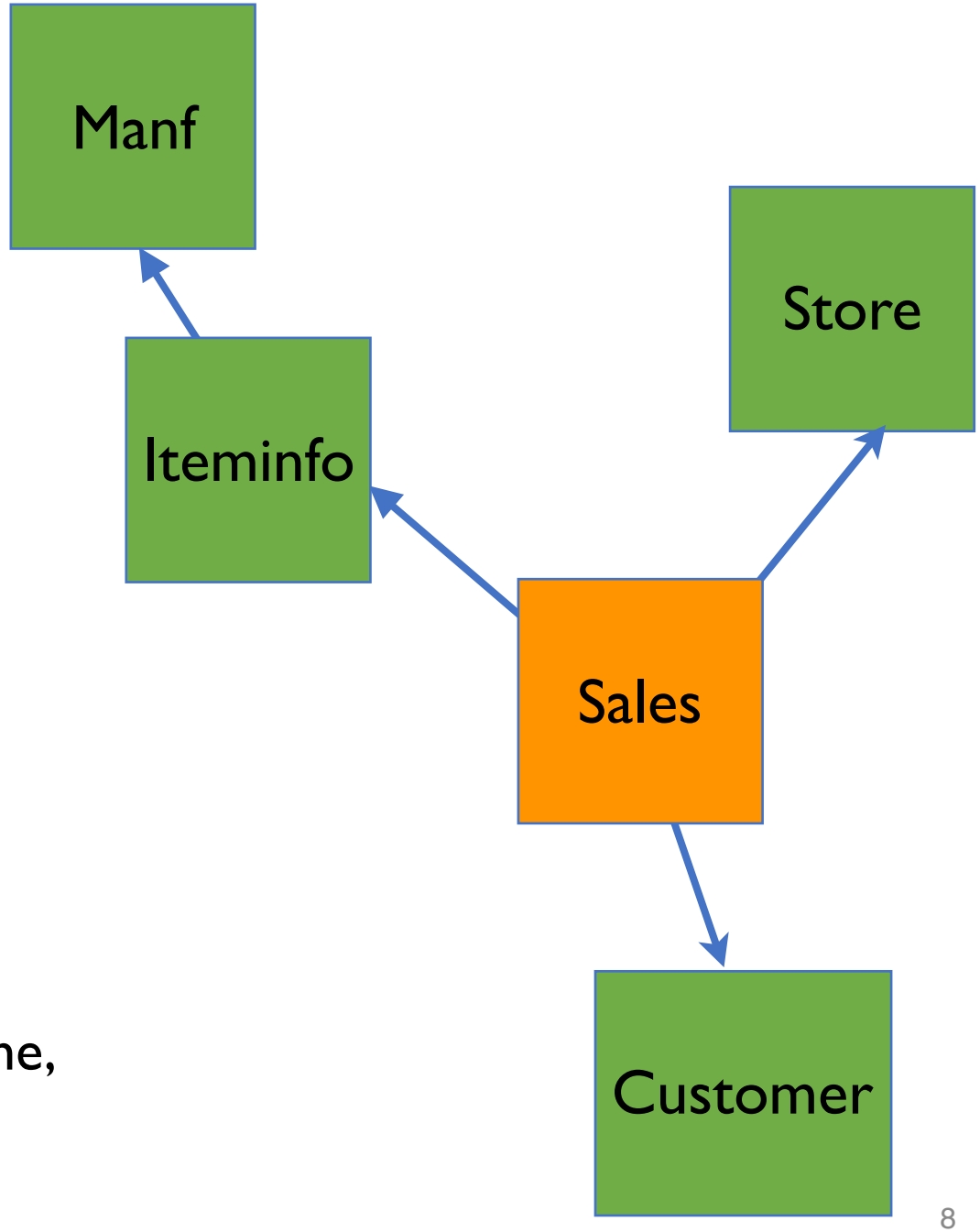
Star Schema

- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)

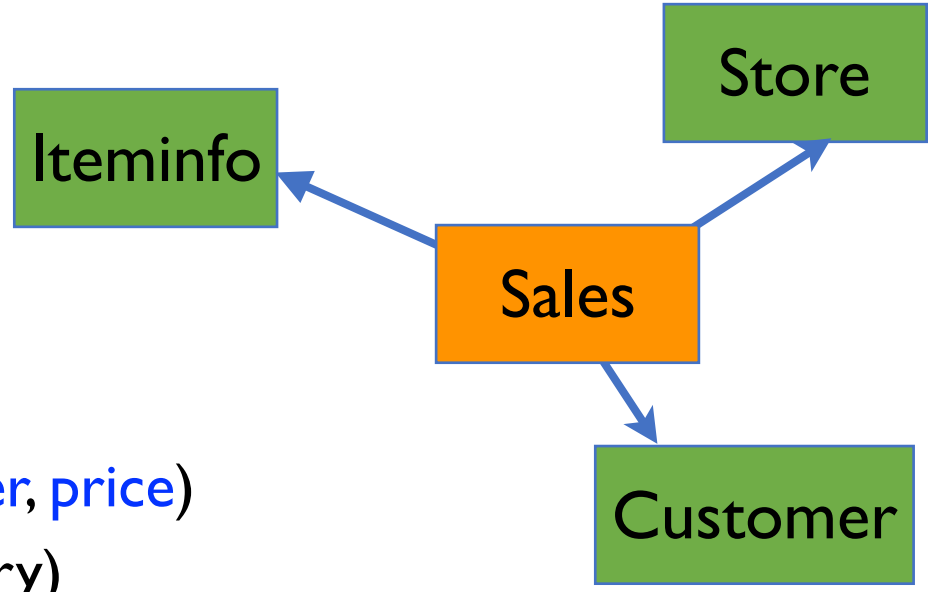


Star Schema

- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category, manfname)
    - Dim table: Manf (name, address, owner)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)



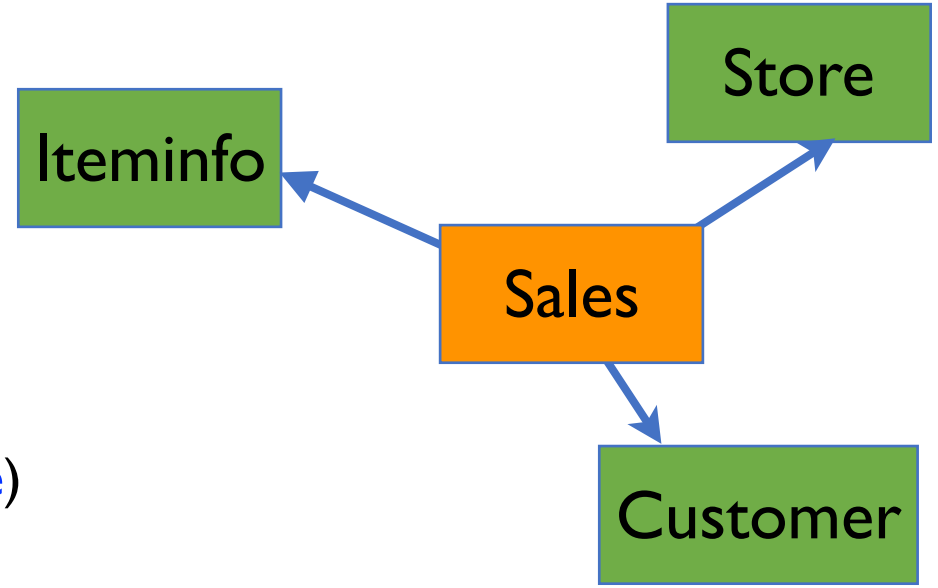
Star Schema



- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)

- Typical "report" queries GROUP BY some dimension attribs, aggregate some measure attributes
  - SELECT category, country, COUNT(number)
  - FROM Sales NATURAL JOIN Iteminfo NATURAL JOIN Store
  - GROUP BY category, country
- Q: What does this query return?

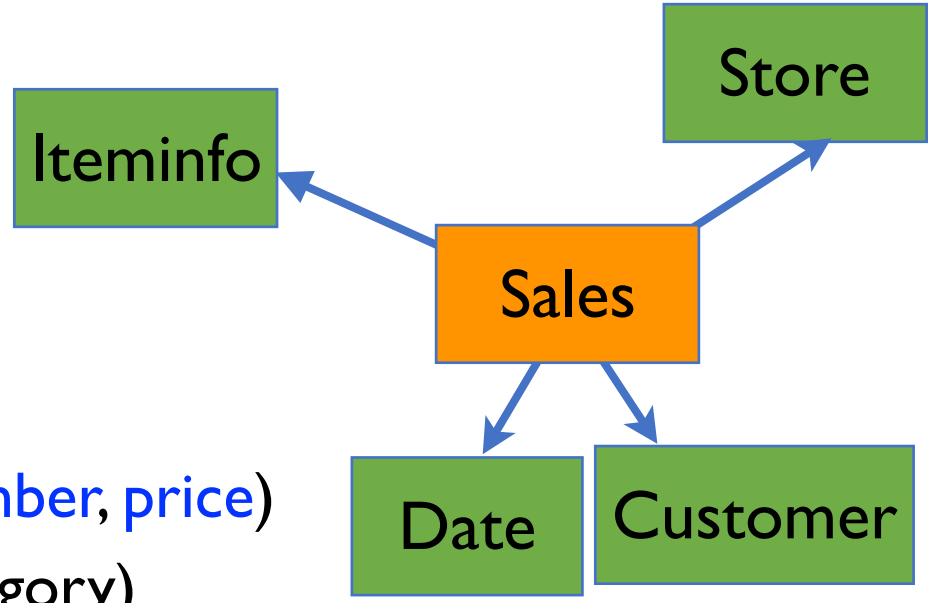
Star Schema



- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)

- Typical "report" queries GROUP BY some dimension attribs, aggregate some measure attributes
- Not very useful to "aggregate" date; better to treat date as an implicit dimension!
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Implicit Dim table: Dateinfo (date, month, quarter, year)
- Allows us to look at trends across dates

Star Schema



- Let's take a canonical example for a data warehouse:
  - Fact table: Sales (itemid, storeid, customerid, date, number, price)
  - Dim table: Iteminfo (itemid, itemname, color, size, category)
  - Dim table: Store (storeid, city, state, country)
  - Dim table: Customer (customerid, name, street, city, state)
  - Dim table: Dateinfo (date, month, quarter, year)
- Query:
  - SELECT category, country, month, COUNT(number)
  - FROM Sales NATURAL JOIN Iteminfo NATURAL JOIN Store
  - GROUP BY category, country, month

Introducing Data Cubes

- More convenient to describe these concepts on a "denormalized view", where all the fact and dimension tables are joined together, but same considerations apply across the star or snowflake schema
- Consider a simpler relation: (itemname, color, size, number)

Itemname	Color	Size	Number
Jacket	Blue	Small	1
Jacket	Red	Medium	1
Jeans	Black	Large	2
…	…	…	…

Vanilla GROUP BY across all groups

- Q: If there are n item names, m colors, and k sizes, what are the number of rows?

Itemname	Color	Size	Number
Jacket	Blue	Small	23
Jacket	Blue	Medium	17
Jacket	Blue	Large	34
Jacket	Red	Small	18
…	…	…	…
Jeans	Blue	Small	14
…	…	…	13

Say I was interested in only the color and itemnames…

- I might want to see what is known as a *cross-tabulation* of the aggregates corresponding to Itemname and Color

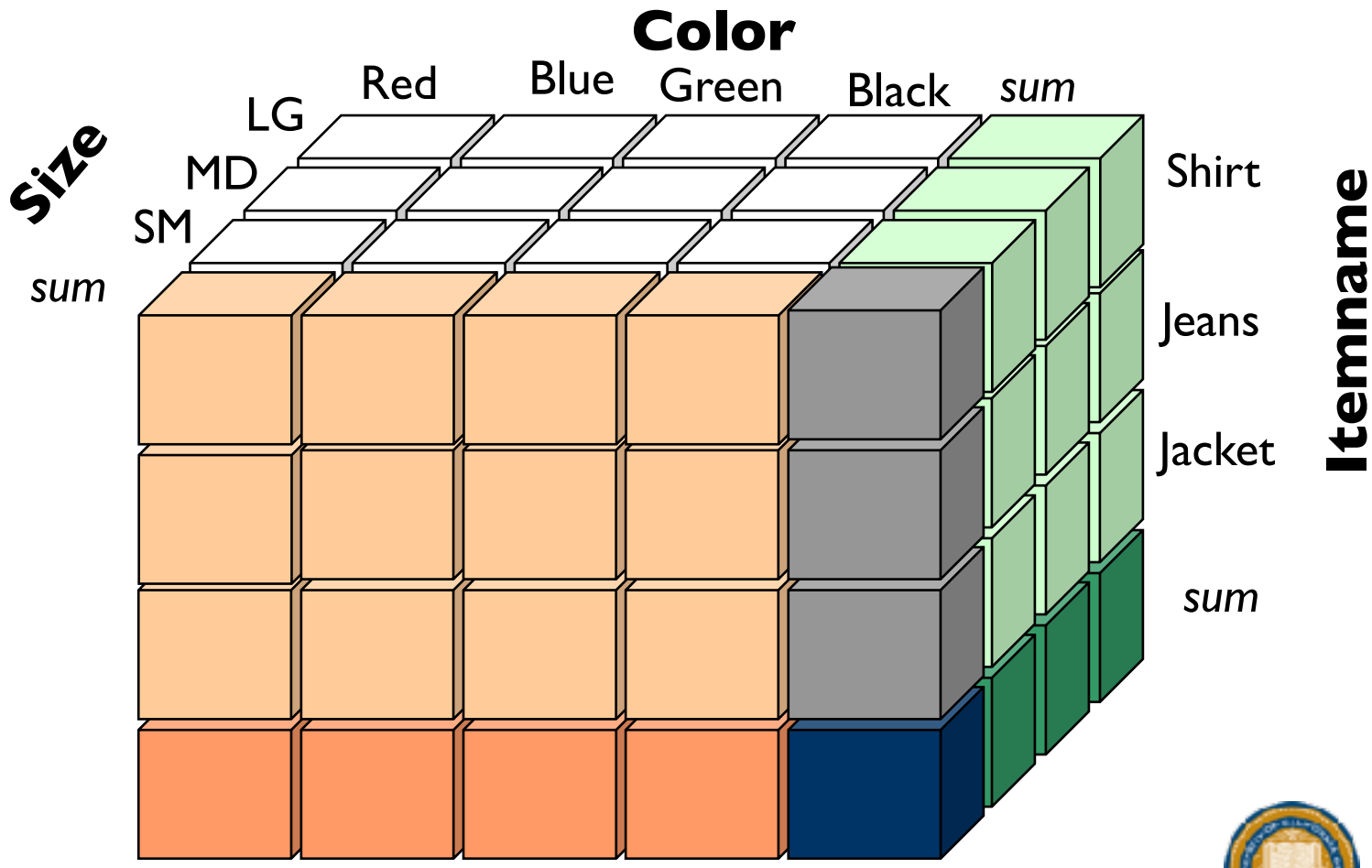
- Q: What does this remind you of?
- Q: How could you get this via a GROUP BY?

	Blue	Red	…	Total
Jacket	23	45	…	234
Jeans	24	28	…	462
…	…	…	…	…
Total	**89**	**132**	**…**	**2384**

Another way to view this…

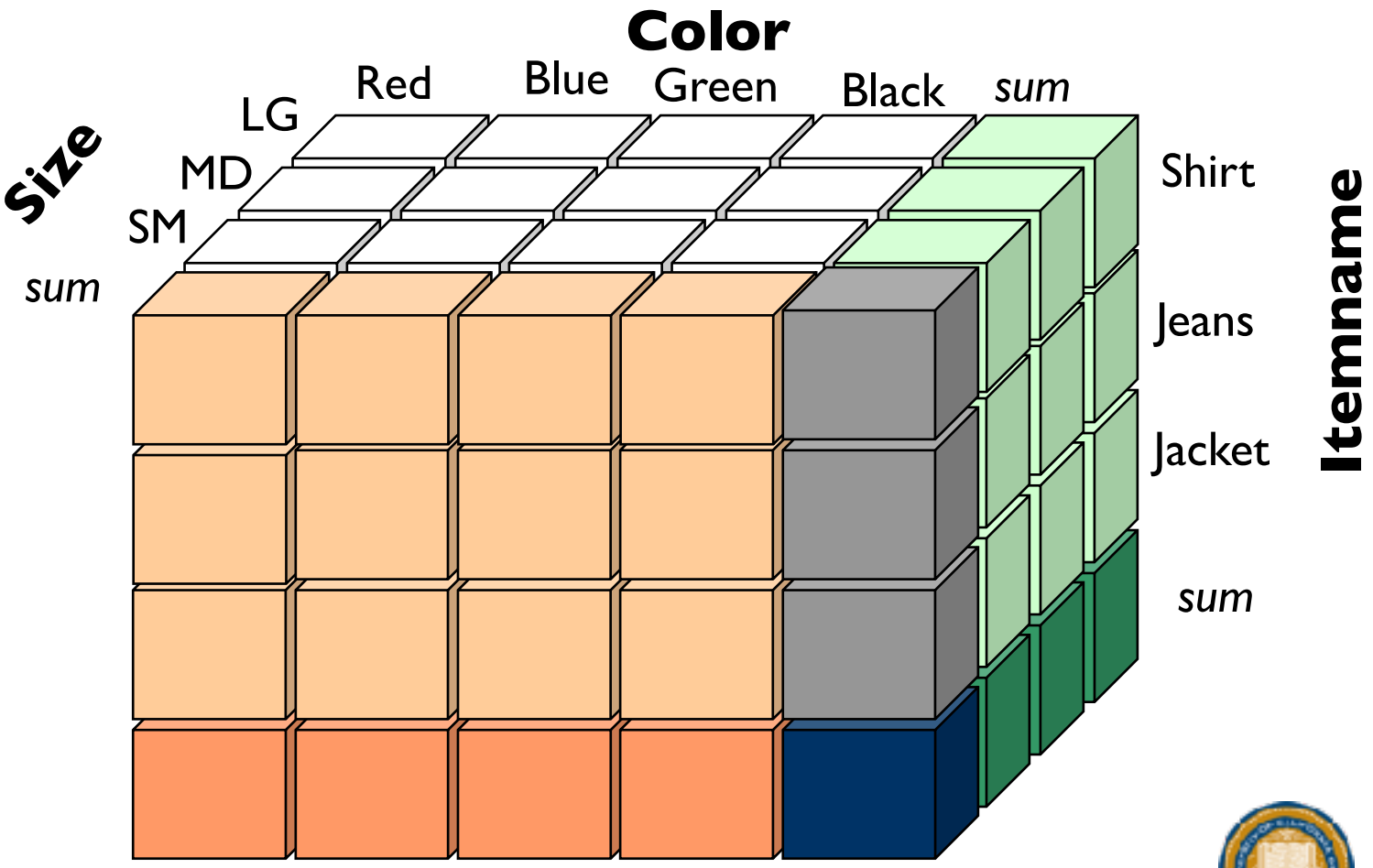
- Crosstab of item name and color = vertical plane closest to us

	Blue	Red	…	Total
Jacket	23	45	…	234
Jeans	24	28	…	462
…	…	…	…	…
Total	**89**	**132**	**…**	**2384**



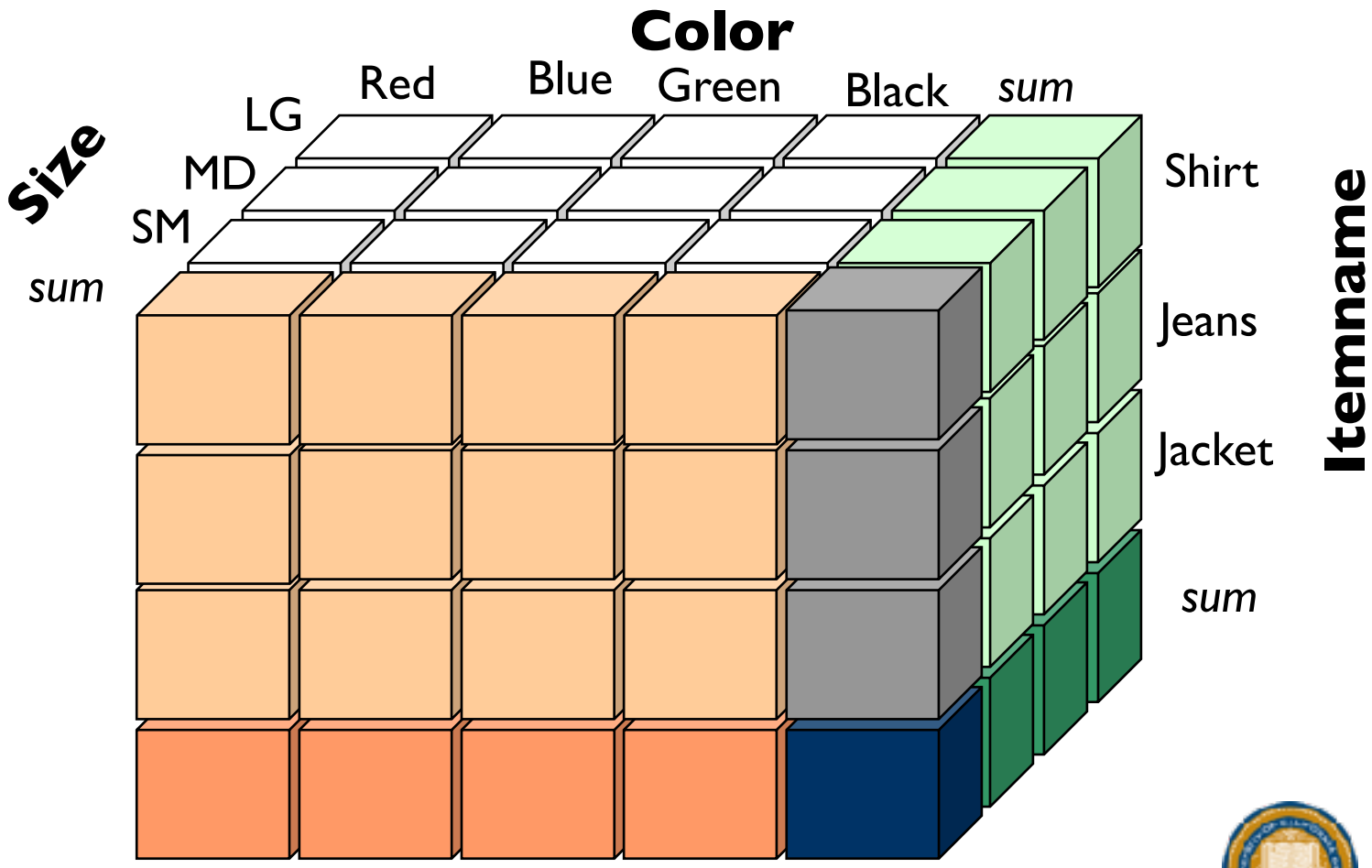
Another way to view this…

- This is a *data cube*
- This data cube has 3 dimensions (hence the name dimensions for those attributes!)
- This cube can be *sliced* and *diced* in various ways



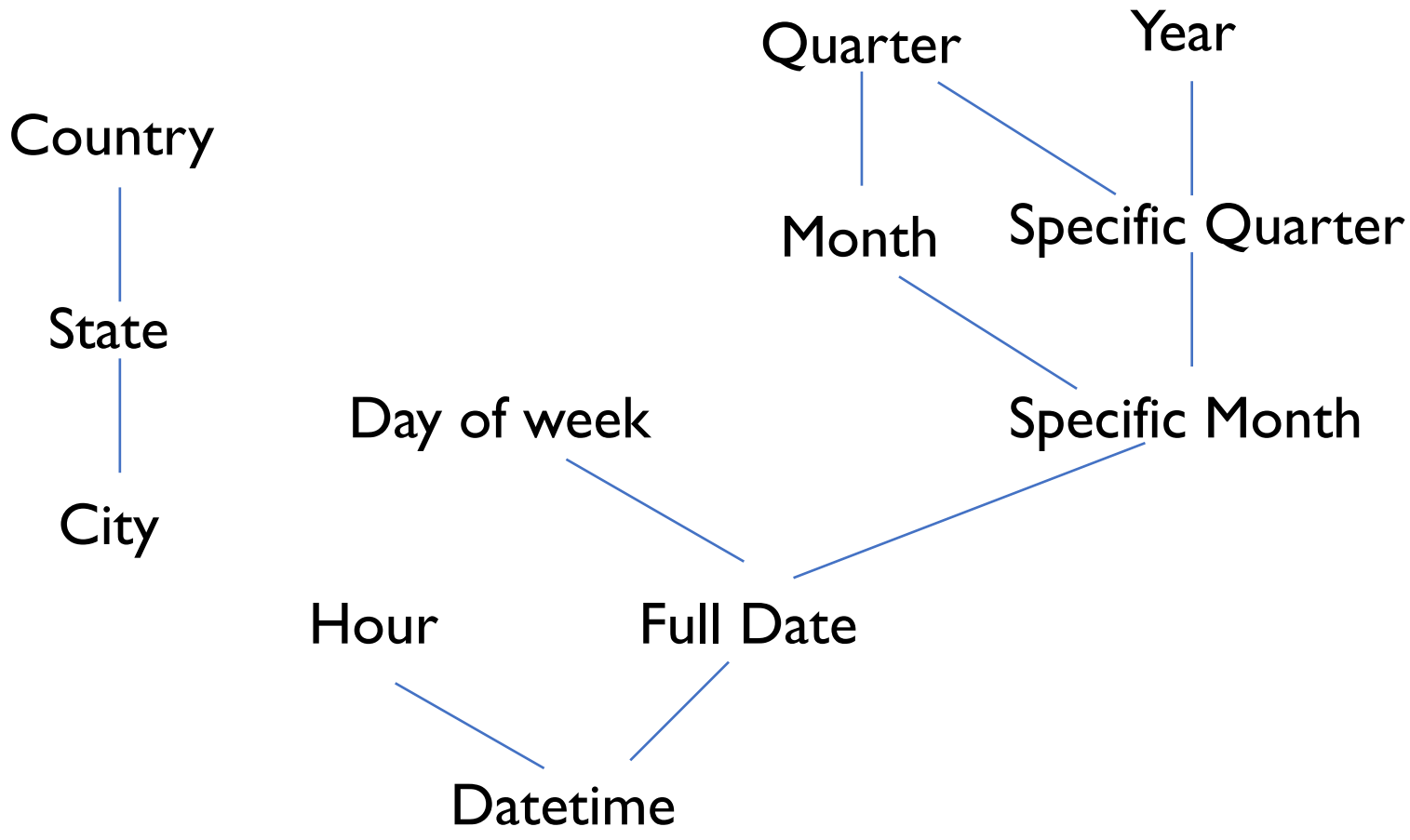
Another way to view this…

- Slicing is equivalent to adding a conditions to one/more of the dimensions
  - e.g., green color
- Dicing is a partitioning of the dimension
  - Here, we partitioned based on the distinct values, but we can partition in more coarse grained ways
  - e.g., lights and darks for color
- This is especially useful for the date dimension:
  - Can group by months, days, years, weeks, etc.



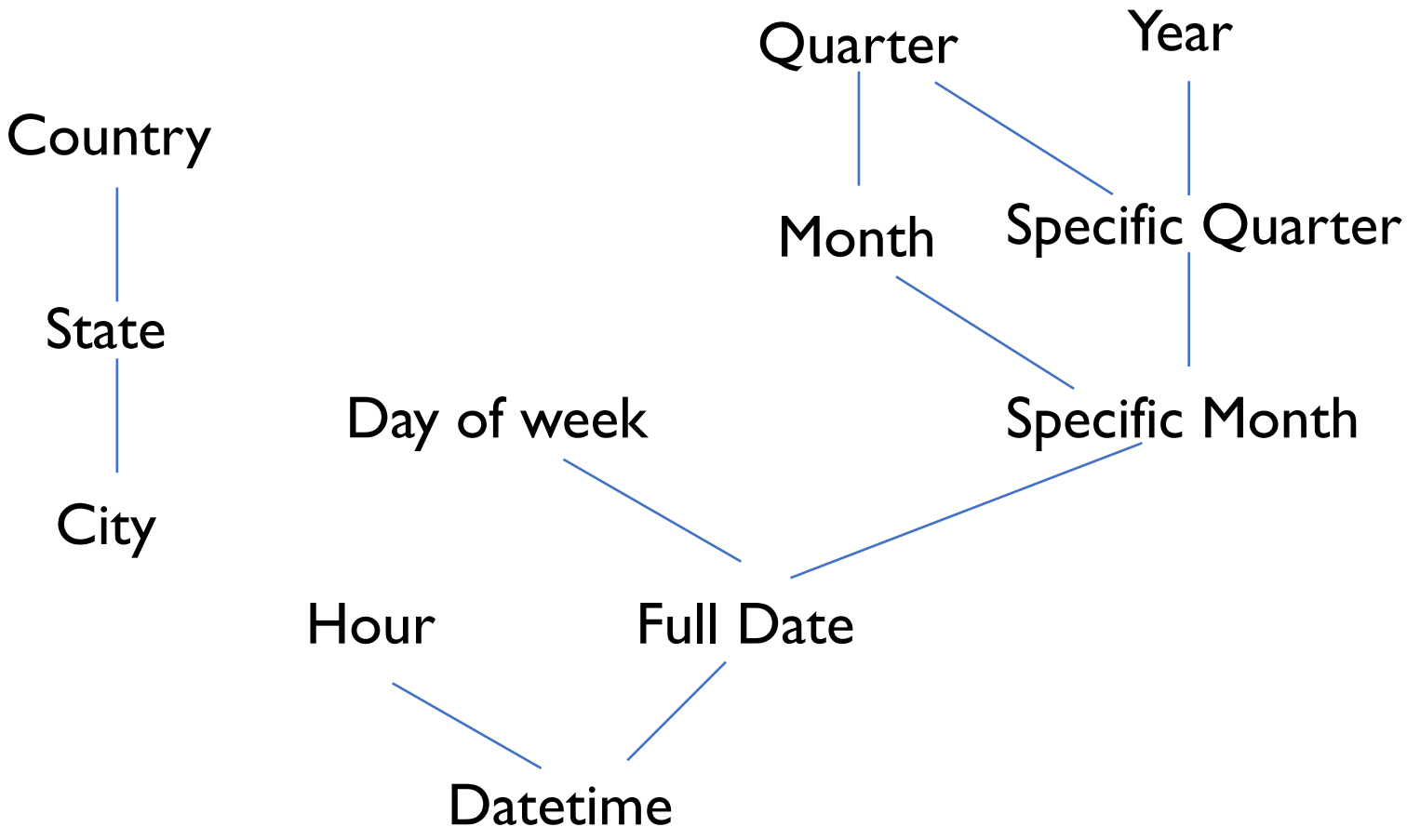
Hierarchies for Setting the Partitioning Granularity

- The partitioning granularity can be set based on user needs…
- Different partitioning may be useful for different applications



Hierarchies for Setting the Partitioning Granularity

- Different partitioning may be useful for different applications
- Q: I want the aggregates per month. What if I set my partitioning based on Full Date and computed the data cube? Can I avoid recomputing the cube (or cross-tab)
- Q: What about if I had set my partitioning based on Year?



Moves in the Hierarchy and Corresponding Cube

- Moving from a finer granularity to a coarser granularity is called a *rollup*
- Moving from a coarser granularity to a finer granularity is called a *drill-down*

- Two unit steps from coarse to fine or vice versa:
  - Move down (or up) the hierarchy for one of the dimensions, OR
  - Move from
    - the origin to an edge, or
    - an edge to a plane, or
    - a plane to the cube

OLAP in SQL: CUBE

- "NULL" is used to indicate "ALL"

- SELECT itemname, color, SUM(number)
- FROM Sales
- GROUP BY CUBE (itemname, color)

- Any attribute may be replaced with an ALL.

Itemname	Color	Number
Jacket	Blue	23
…	…	…
Jacket	Green	34
Jeans	Blue	28
…	…	…
Jeans	Green	17
Jacket	NULL	185
Jeans	NULL	200
…	…	…
NULL	Blue	94
NULL	Red	74
…	…	…
NULL	NULL	984

ROLLUP is an alternative to CUBE

- SELECT itemname, color, SUM(number)
- FROM Sales
- GROUP BY ROLLUP (itemname), ROLLUP (color, size)

- Every combination of:
    - specific itemname or ALL for the first rollup
    - for the second rollup
        - specific color and specific size
        - specific color and ALL sizes
        - ALL colors and ALL sizes
- Thus, combinations include {(itemname, color, size), (itemname, color), (itemname), (color, size), (color), ()}

How do we pick the right CUBE/ROLLUP query?

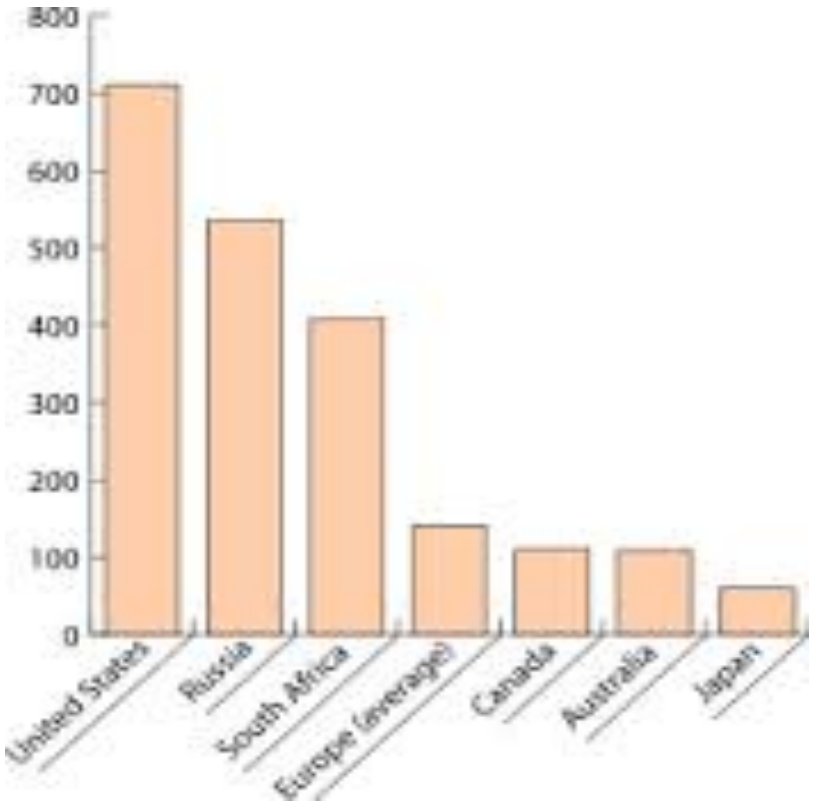
- First off, why does this matter?
  - If this query is being run once on a petabyte sized warehouse, it is important to get it right!
- Think about all the ways you want to slice and dice your data
- Pick as fine a granularity of partitioning so that you can recreate all the aggregates, but not so fine as to blow up the query result
  - Remember, the query result size grows exponentially in the attributes; this can be quite bad in large snowflake schemas
  - This is known as the *curse of dimensionality*

How does this relate to visualizations?

- Most visualizations are group-by queries

- SELECT AGG(M), D
- FROM R
- WHERE …
- GROUP BY D

- SELECT COUNT(*), Country
- FROM R
- GROUP BY Country



Why Visualizations?

- Analyze
  - Discover trends
    - Stock price is going up/down
  - Develop hypotheses
    - House prices are down due to the downturn
  - Check hypotheses
  - Detect errors
    - Null values in a column
- Share, record & communicate
- Our focus will primarily be on basic visualization types, how they relate to the data types, and the data processing aspects: Marti Hearst's classes are highly recommended for a deep-dive into visualization.

Types of Data: A Visualization-Oriented Perspective

- Nominal
  - =, ≠
- Ordinal
  - =, ≠, <, >
- Quantitative Interval
  - =, ≠, <, >, –
  - Arbitrary zero
- Quantitative Ratio
  - =, ≠, <, >, –, %
  - Physical quantities

Airlines, Genre

Film ratings, Batteries

Year, Location

Sales, Profit, Temperature

Types of Data: A Visualization-Oriented Perspective

- Nominal
  - =, ≠
- Ordinal
  - =, ≠, <, >
- Quantitative Interval
  - =, ≠, <, >, –
  - Arbitrary zero
- Quantitative Ratio
  - =, ≠, <, >, –, %
  - Physical quantities

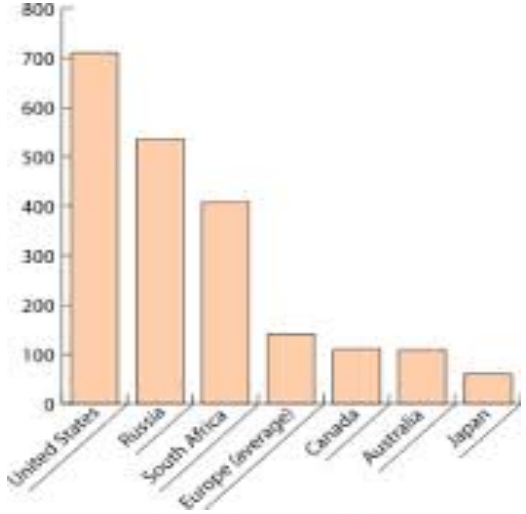
Hot, cold	Good, OK, Bad
	Grade
Temperature	Score

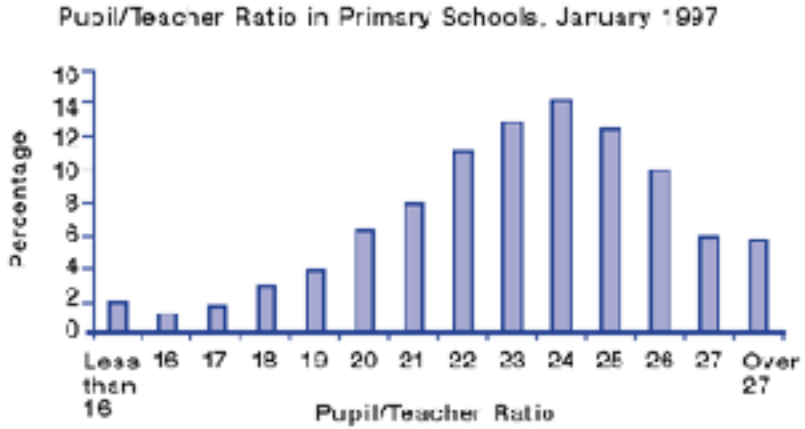
A Very Quick Primer on Visualization Types

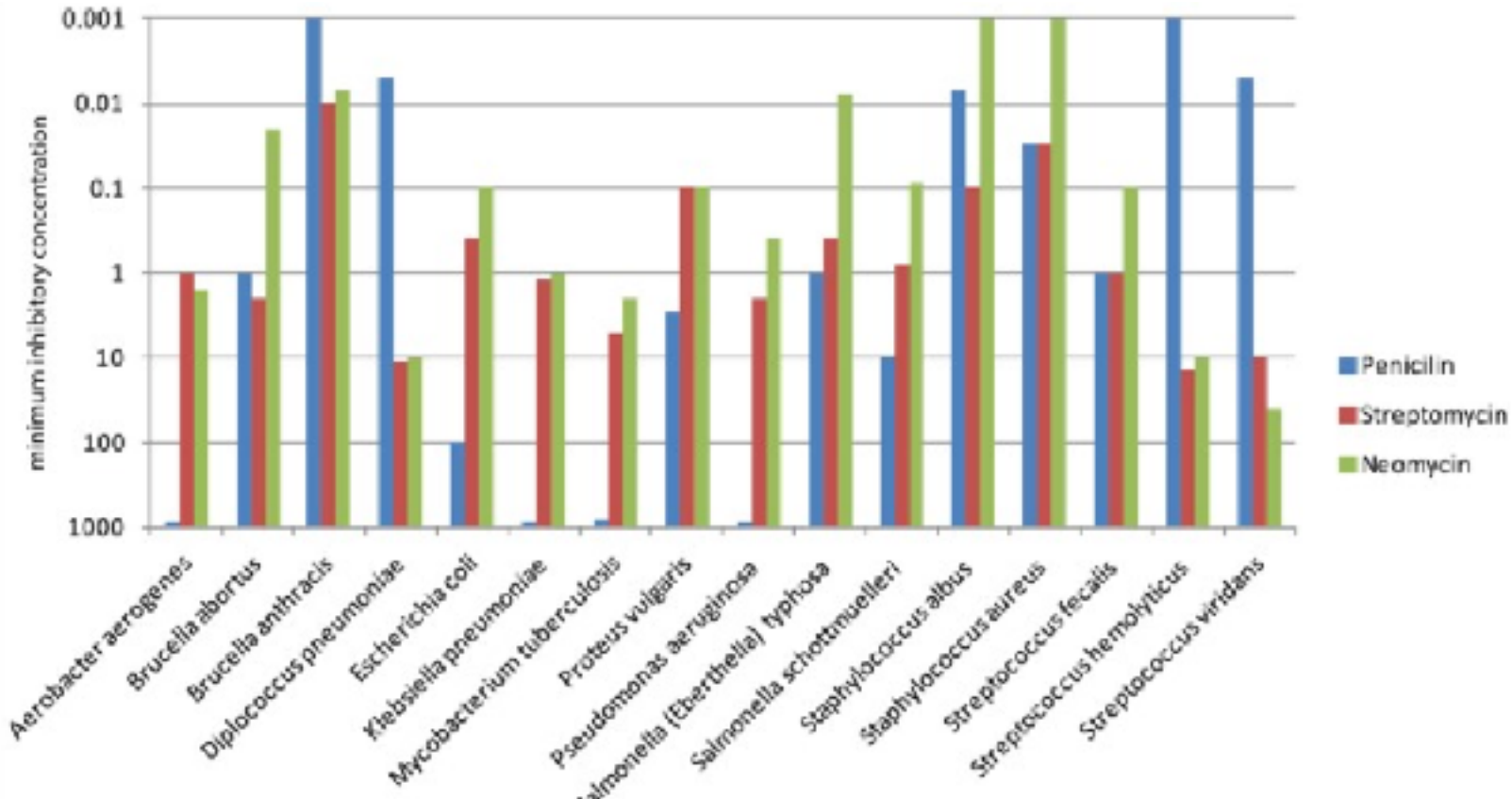
- The most basic visualization is a table!

- Bar Charts
- Line Charts
- Scatter Plot
- Choropleth

Bar Charts





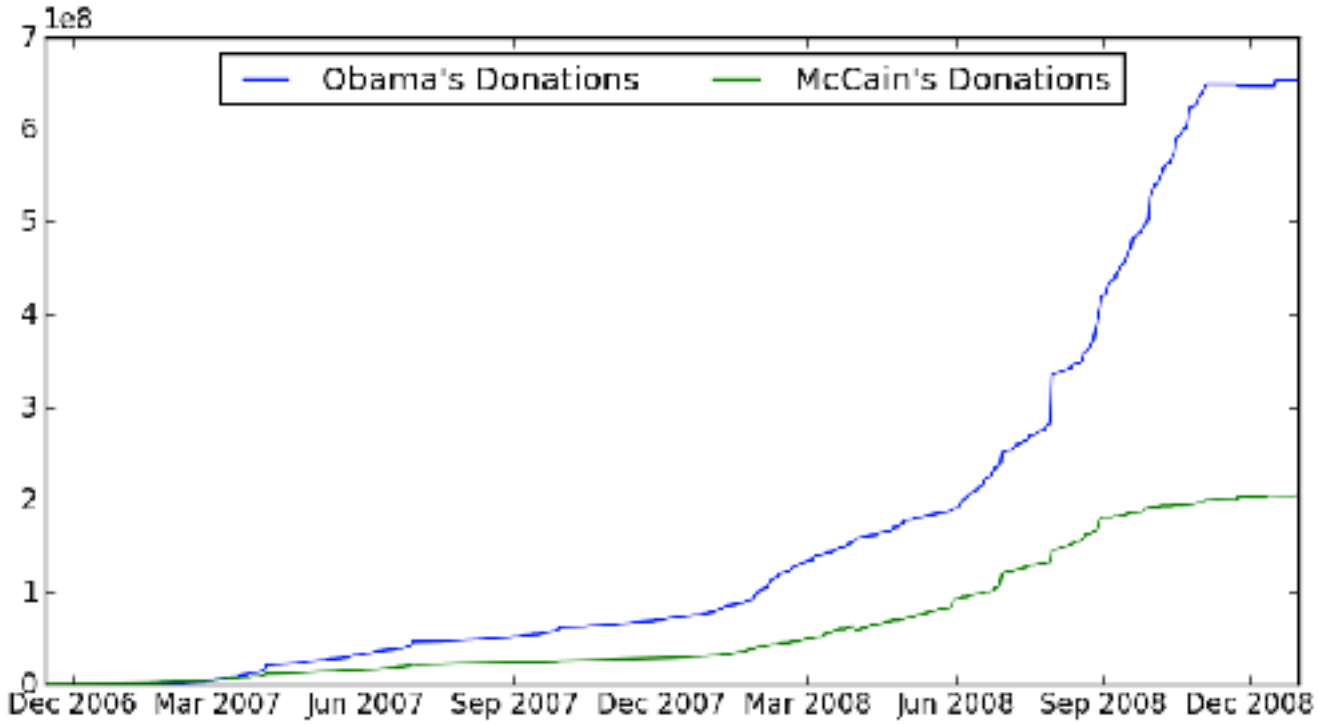


Q: What SQL query generates a multiple bar chart?

When are bar charts appropriate?

- When plotting a Q-R vs. either an N, O, Q-I, or Q-R
- Emphasizes the differences in height than differences in X axis
- Most fundamental chart
- From a SQL standpoint, simple aggregation of some Y axis measure, grouped by one or more dimensions
  - can generate results in the appropriate order in the X axis by doing an ORDER BY following the GROUP BY

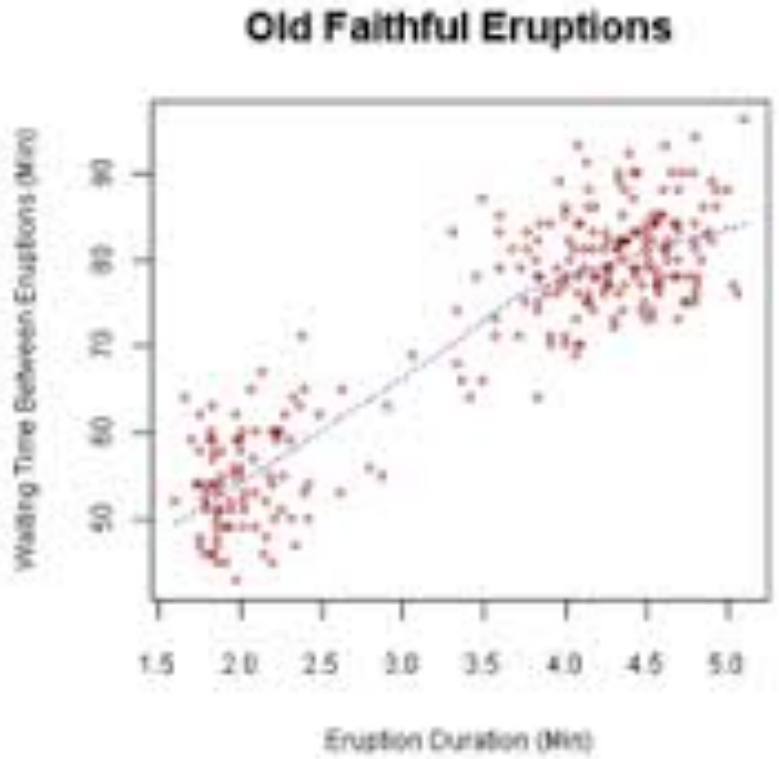
Line Charts

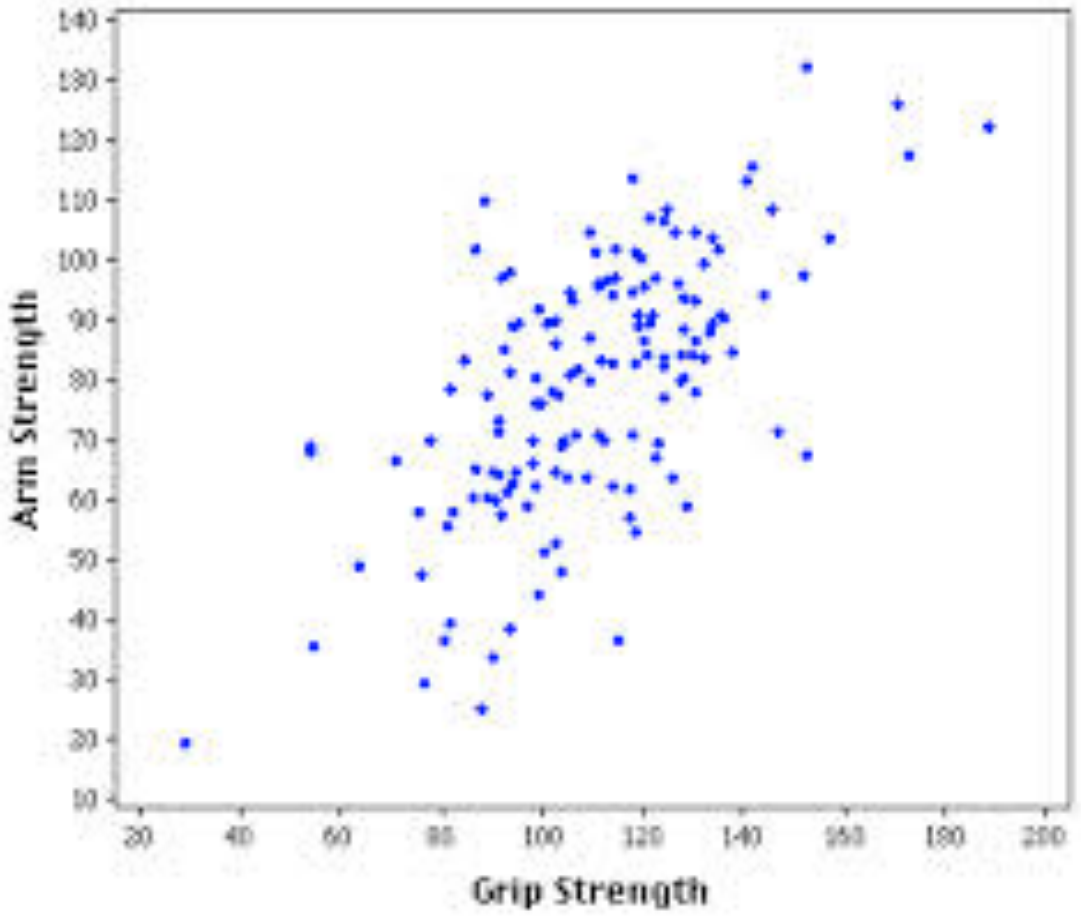

1e8
7
6
5
4
3
2
1
0
Obama's Donations
McCain's Donations
Dec 2006
Mar 2007
Jun 2007
Sep 2007
Dec 2007
Mar 2008
Jun 2008
Sep 2008
Dec 2008

When are line charts appropriate?

- When plotting a Q-R vs. a Q-I or a Q-R
- Mainly makes sense when the X axis is ordered in some way and distances between X axis points matter
  - e.g., is the rate of change in this interval the same as the other interval
- Want to be able to see "trends"
  - There is an assumption of interpolation between points and dependence of the Y-axis on the X-axis
- From a SQL standpoint, the query for generation is similar to bar charts, grouping by the X-axis

Scatterplots

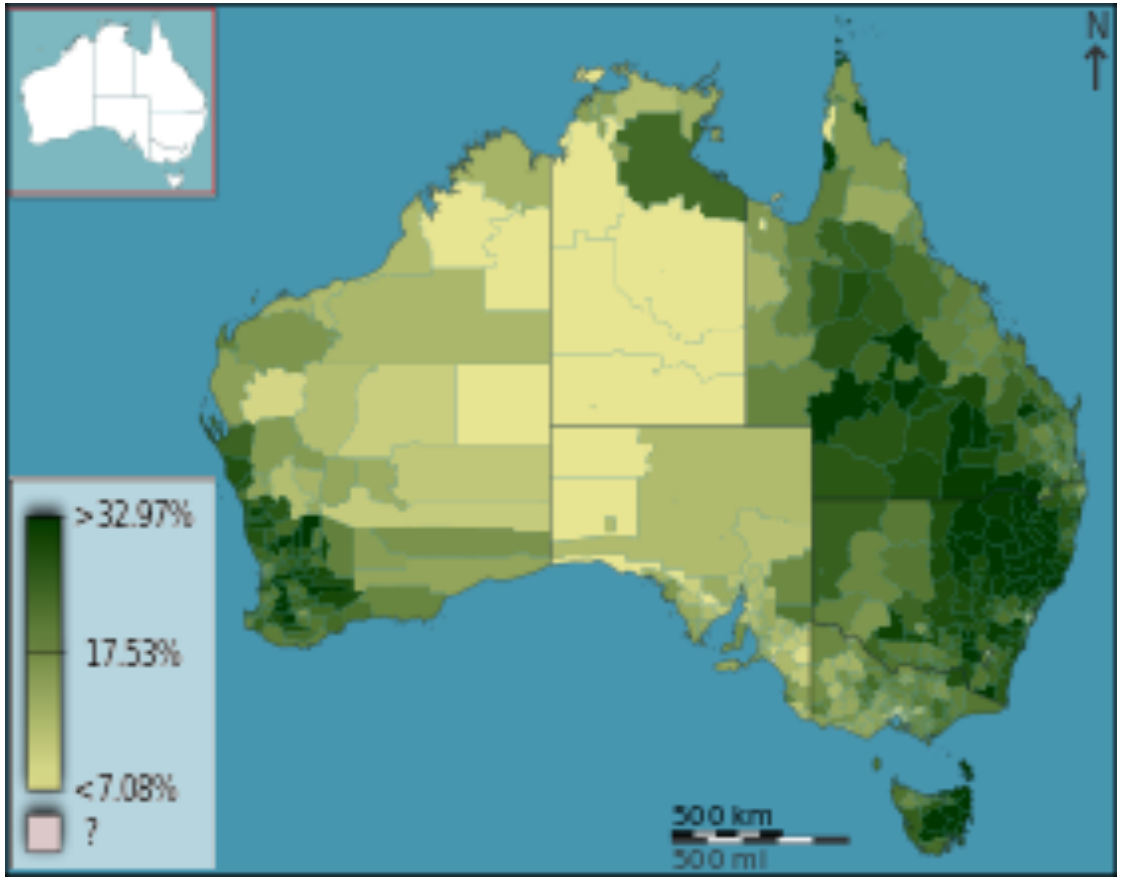




When are scatterplots appropriate?

- When plotting a Q-R vs. a Q-R
- No assumption of interpolation unlike line charts
- Care more about "density", understanding of "correlation"
- From a SQL query standpoint, one way to plot a scatterplot is to simply perform a SELECT X, Y FROM R with no grouping.
    - Additional aspects (e.g., color) can also be selected if needed
- However, there is a danger of too many rows being returned.
    - Imagine a relation of size 1B: 1B pairs returned
    - A safer option in that case is to bin the scatterplot into grid cells
    - Q: How would we do this?
    - A: CTE to add new "binned" columns corresponding to a CASE statement, followed by a GROUP BY on the new columns

Choropleths



When are choropleths appropriate?

- Choropleths map a Q-R vs. a two-dimensional Q-I variable

- From a SQL query standpoint, grouping can be done on a per-geographical region basis followed by overlaying on a map.

What type of visualization would you use?

- A plot of rainfall by location on a map
- A plot of average age by department
- A plot of total sales by year
- A plot of rainfall by average temperature for various locations

Composing Attributes

- The Polaris table algebra allows us to compose attributes prior to visualization in more sophisticated ways.
  - This algebra is the basis behind Tableau, a popular visual analytics platform

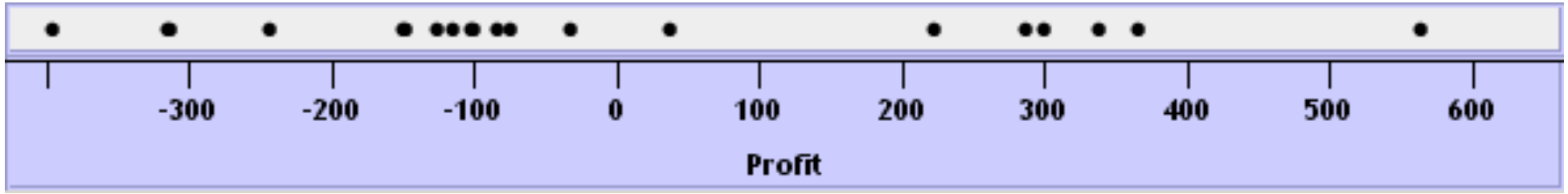
- The table algebra treats Q-R and Q-I as "Q-R" and N and O into "O" (imposing an arbitrary order if needed).

Table Algebra: Basic Operands

- Ordinal fields: interpret domain as a set that partitions table into rows and columns
  - Quarter = {(Qtr1),(Qtr2),(Qtr3),(Qtr4)}

Qtr1	Qtr2	Qtr3	Qtr4
95892	101760	105282	98225

- Quantitative fields: treat domain as a single element set and encode spatially as axes
  - Profit = {(Profit[-410,650])}

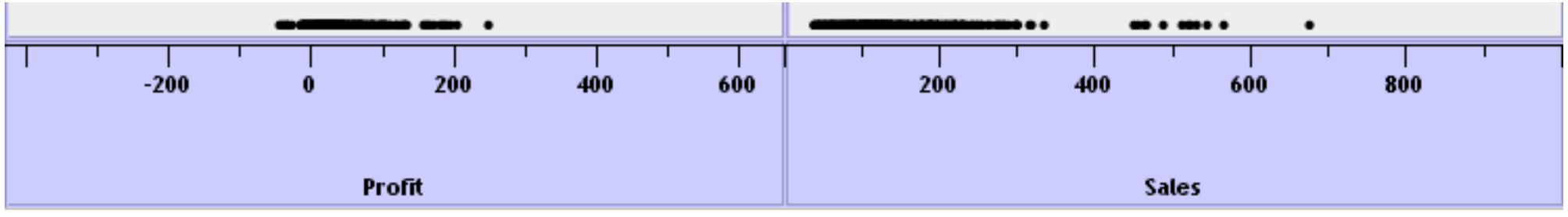


Concatenation Operator (+)

- Ordered union of set interpretation
  - Quarter + ProductType
  - = {(Qtr1),(Qtr2),(Qtr3),(Qtr4)} + {(Coffee), (Espresso)}
  - = {(Qtr1),(Qtr2),(Qtr3),(Qtr4),(Coffee),(Espresso)}

Qtr1	Qtr2	Qtr3	Qtr4	Coffee	Espresso
48	59	57	53	151	21

- Profit + Sales = {(Profit[-310,620]),(Sales[0,1000])}

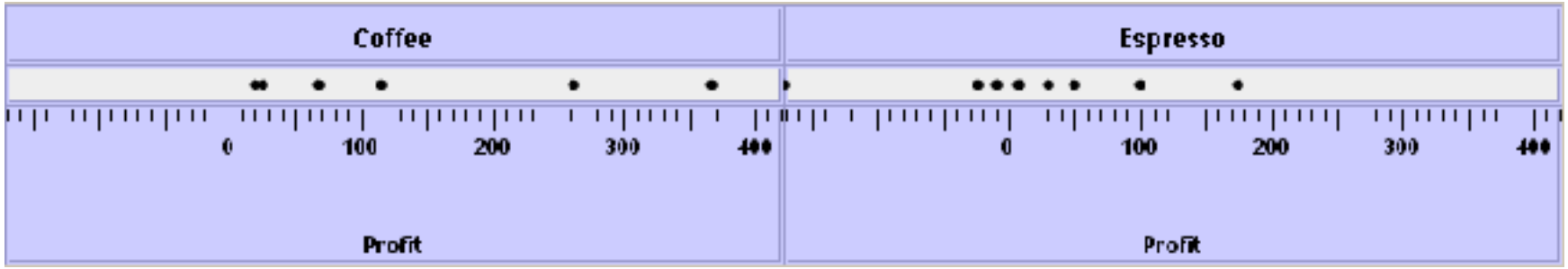


Cross operator (X)

- Cross-product of set interpretation
- Quarter x ProductType =
- {(Qtr1,Coffee), (Qtr1,Tea), (Qtr2, Coffee), (Qtr2,Tea), (Qtr3, Coffee), (Qtr3,Tea), (Qtr4, Coffee), (Qtr4,Tea)}

Qtr1		Qtr2		Qtr3		Qtr4	
Coffee	Espresso	Coffee	Espresso	Coffee	Espresso	Coffee	Espresso
181	19	160	20	178	12	134	33

- ProductType x Profit =



Nest Operator (/)

- Quarter x Month
  - would create entry twelve entries for each quarter. i.e., (Qtr1, December)
- Quarter / Month
  - would only create three entries per quarter
  - based on tuples in database not semantics
  - can be expensive to compute