Data Warehousing & OLAP

Text:

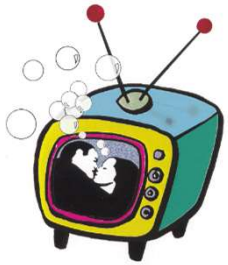
Chapter 25

Other References:

Database Systems: The Complete Book, 2nd edition, by Garcia-Molina, Ullman, & Widom

The Data Warehouse Toolkit, 3rd edition, by Kimball & Ross

Databases: the continuing saga



When last we left databases…

- We had decided they were great things
- We knew how to conceptually model them in ER diagrams
- We knew how to logically model them in the relational model
- We knew how to normalize our database relations
- We could write queries in different languages

Next: what data format to people use for analysis?

Learning Goals

- Compare and contrast OLAP and OLTP processing (e.g., focus, clients, amount of data, abstraction levels, concurrency, and accuracy).
- Explain the ETL tasks (i.e., extract, transform, load) for data warehouses.
- Explain the differences between a star schema design and a snowflake design for a data warehouse, including potential tradeoffs in performance.
- Argue for the value of a data cube in terms of:
  - The type of data in the cube (numeric, categorical, temporal, counts, sums)
  - The goals of OLAP (e.g., summarization, abstractions), and
  - The operations that can be performed (drill-down, roll-up, slicing/dicing).
- Estimate the complexity of a data cube, in terms of the number of equivalent aggregation queries.

What We Have Focused on So Far

- **OLTP (On-Line Transaction Processing)**
  - Transaction-oriented applications, typically for data entry and retrieval transaction processing.
  - The system responds immediately to user requests.
  - High throughput and insert- or update-intensive database management. These applications are used concurrently by hundreds of users.
- The key goals of OLTP applications are availability, speed, concurrency and recoverability.

source: Wikipedia

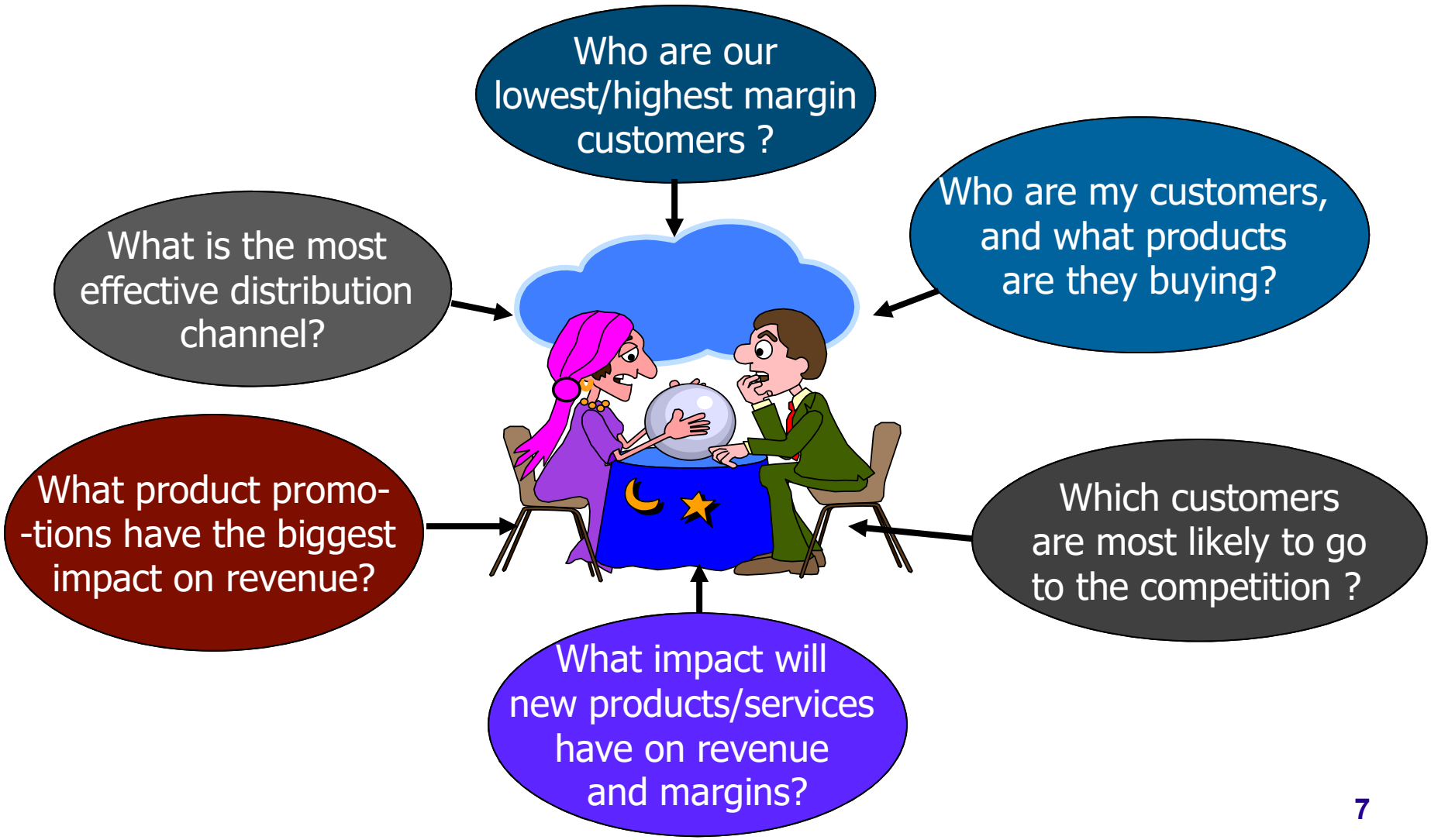
Operational Data

- Operational data is also called transactional information
- What OLTP works with
- Data that supports a company's day-to-day operations
- A company can have multiple operational data sources
- Some Examples:
  - Sales transactions
  - Airline ticket reservations
  - Withdrawing Cash

Can We Do More?

- Increasingly, organizations are analyzing current and historical data to identify useful patterns and support business strategies.
  - a.k.a. “Decision Support”, “Business Intelligence”
- The emphasis is on complex, interactive, exploratory analysis of very large datasets created by integrating data from across all parts of an enterprise.

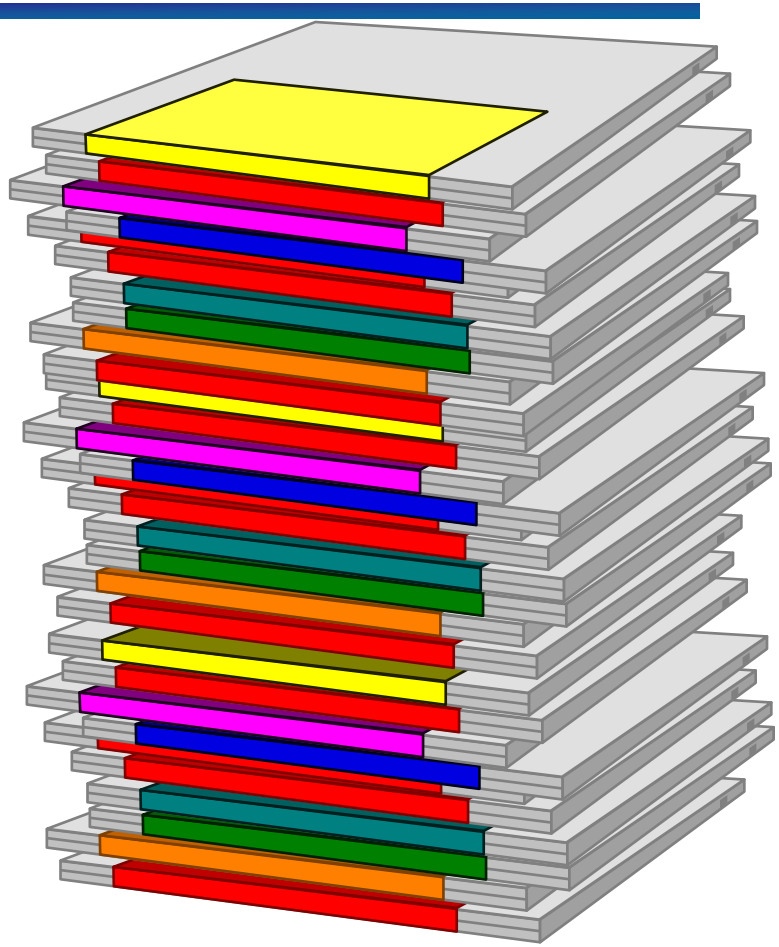
A Producer Wants to Know …



What is Data Warehouse?

- "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."

—W. H. Inmon



Recognized by many as the father of the data warehouse

Data Warehouses are Integrated

- Constructed by integrating multiple, heterogeneous data sources.
  - relational databases, XML, flat files, on-line transaction records
- Data cleaning and data integration techniques are applied.
  - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources.
    - e.g., Hotel price depends on: currency, various room taxes, whether breakfast or Internet is included, etc.
  - When data from different sources is moved to the warehouse, it is cleaned and converted into a common format.

Typical Data Warehouse Scenario (1/2)

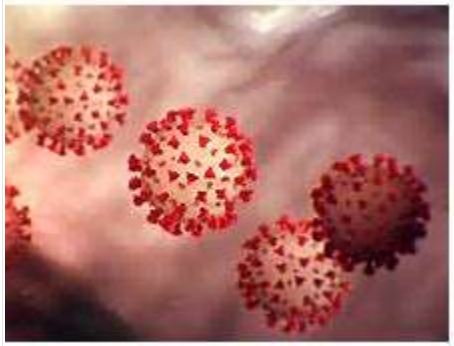
- Consider Canada Safeway's data sources:
  - Operational data from daily purchase transactions, in each store
  - Data about item placement on shelves
  - Supplier data
  - Data about employees, their compensation, etc.
  - Sales/promotion plans
  - Product categories and sub-categories; brands, types; customer demographics; time and date of sale
- Each of the above is essentially an autonomous OLTP database (or set of tables)
  - Local queries; no queries cutting across multiple databases
  - Data must be current at all times
  - Support for concurrency, recovery, and transaction management are a must.

Typical Data Warehouse Scenario (2/2)

- Consider the following use-case queries:
  - How does the sale of hamburgers for Feb. 2015 compare with that for Feb. 2014?
  - What were the sales of ketchup like last week (when hamburgers and ketchup were placed next to each other) compared to the previous week (when they were far apart)?
  - What was the effect of the promotion on ground beef on the sales of hamburger buns and condiments?
  - How has the reorganization of the store(s) impacted sales?
  - What was the total sales volume on all frozen food items (not just one item or a small set of items)?

Data Warehouse Integration Challenges

- When getting data from multiple sources, must eliminate mismatches (e.g., different currencies, units, schemas)
- e.g., when trying to warehouse all of the COVID-19 data across the world:
- Exercise: Provide some examples of mismatches that might occur in this example.



DW Integration Challenges (cont.)

- e.g., Shell may need to deal with data mismatches throughout the multinational organization:
  - Multiple currencies and dynamic exchange rates
  - Gallons vs. liters; thousands of cubic feet (of gas) vs. cubic meters vs. British Thermal Units (BTUs)
  - Different suppliers, contractors, unions, and business partner relationships
  - Different legal, tax, and royalty structures
  - Local, provincial, federal, and international regulations
  - Different statutory holidays (when reporting holiday sales)
  - Light Sweet Crude (Nigeria) vs. Western Canada Select (Alberta, heavier crude oil)
  - Joint ownership of resources (partners)
  - Retail promotions in its stores; different products

Operational DBMS vs. Data Warehouse summary

- Operational DBMS
  - Day-to-day operations: purchasing, inventory, banking, payroll, manufacturing, registration, accounting, etc.
  - Used to run a business
- Data Warehouse
  - Data analysis and decision making
  - Integrated data spanning long time periods, often augmented with summary information
  - helps to *"optimize"* the business

Why a Separate Data Warehouse?

- High performance for both systems
  - DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery
  - Warehouse—tuned for complex queries, multidimensional views, consolidation
- Different types of queries
  - Extensive use of statistical functions which are poorly supported in DBMS
  - Running queries that involve conditions over time or aggregations over a time period, which are poorly supported in a DBMS
  - Running related queries that are generally written as a collection of independent queries in a DBMS

Analytical Data

- Collected for decision support and data analysis
- Example of analytical information:
  - Patterns of air ticket reservations during Christmas break
- Sales trends over the past 10 years
- Analytical information is based on operational information

What are we talking about now? On-Line Analytical Processing

- Technology used to perform complex analysis of the data in a data warehouse.
  - OLAP enables analysts, managers, and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information.
  - The data has been **E**xtracted **T**ransformed and **L**oaded (ETL) from raw data to reflect the enterprise as understood by the user.
- OLAP queries are, typically:
  - Full of grouping and aggregation
  - Few, but complex queries -- may run for hours

OLTP vs. OLAP

	OLTP	OLAP
Typical User	Basically Everyone (Many Concurrent Users)	Managers, Decision Support Staff (Few)
Type of Data	Current, Operational, Frequent Updates	Historical, Mostly read-only
Type of Query	Short, Often Predictable	Long, Complex
# query	Many concurrent queries	Few queries
Access	Many reads, writes and updates	Mostly reads
DB design	Application oriented	Subject oriented
Schema	E-R model, RDBMS	Star or snowflake schema
Normal Form	Often 3NF	Unnormalized
Typical Size	MB to GB	GB to TB
Protection	Concurrency Control, Crash Recovery	Not really needed
Function	Day to day operation	Decision support

Clicker question: Are the following OLTP or OLAP?

- UBC wants to figure out how many students are currently taking CPSC 304

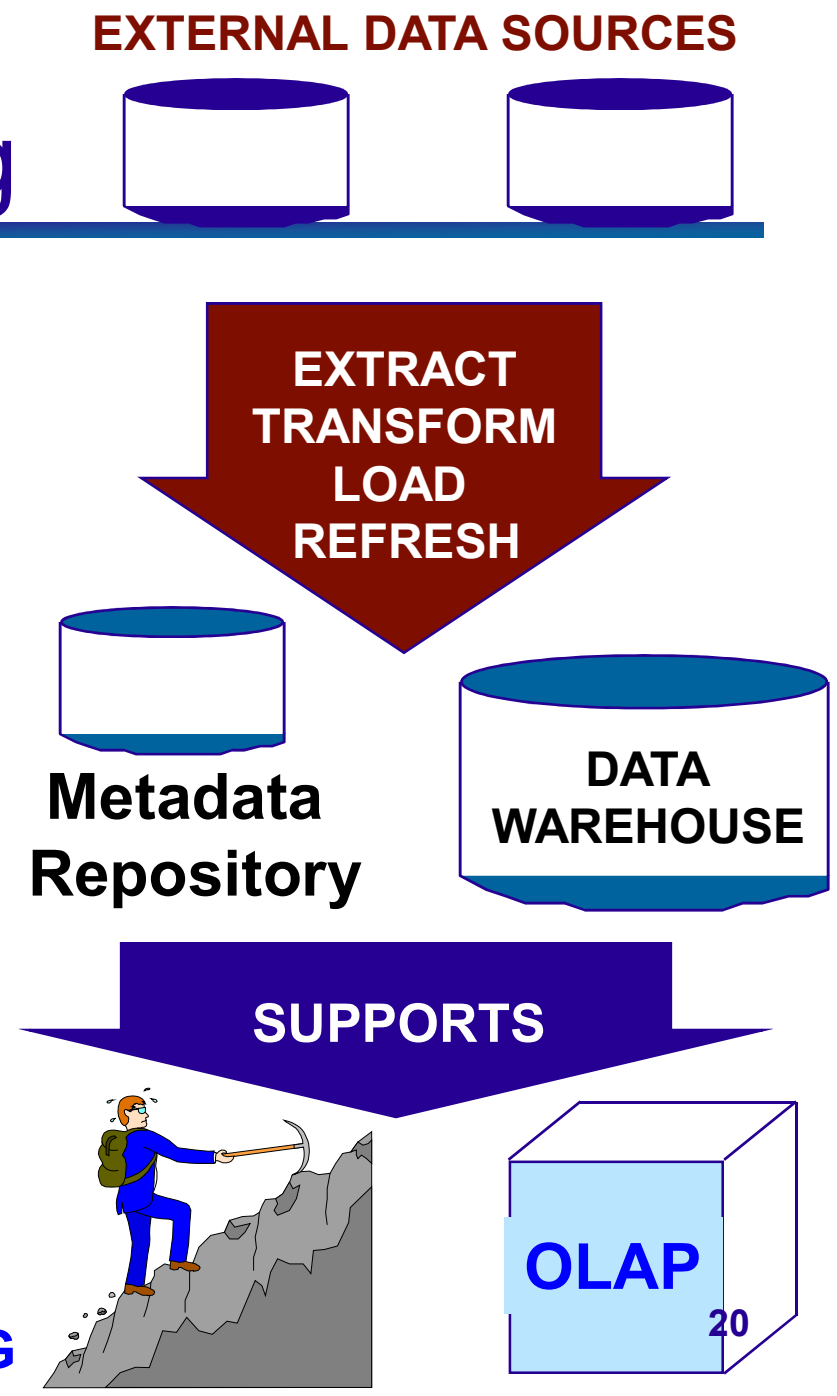
  A. OLTP B. OLAP

- UBC wants to figure out how many students have taken all courses over time and analyze for trends

  A. OLTP B. OLAP

Data Warehousing

The process of constructing and using data warehouses is called data warehousing.



Business Intelligence: 3 Major Areas

1. **Data Warehousing**
   - Consolidate and integrate operational OLTP databases from many sources into one large, well-organized repository
     - e.g., acquire data from different stores or branches
     - Must handle conflicts in schemas, semantics, platforms, integrity constraints, etc.
     - May need to perform **data cleaning**
   - Load data through periodic updates
     - Synchronization and currency considerations
   - Maintain an archive of potentially useful historical data including data that is an aggregation or summary, but has been pre-processed.

Data warehouses are then used for…

Business Intelligence: 3 Major Areas

2. **OLAP**
   - Perform complex SQL queries and views, including trend analysis, drilling down for more details, and rolling up to provide more easily understood summaries.
   - Queries are normally performed by domain (business) experts rather than database experts.
3. **Data Mining**
   - Exploratory search for interesting trends (patterns) and anomalies (e.g., outliers, deviations) using more sophisticated algorithms (as opposed to queries).

That is all great, but what are the challenges with data warehousing?

- Semantic Integration: Extract, Transform, Load challenges. We already talked about the COVID-19 example.
- Heterogeneous Sources: Must access data from a variety of source formats and repositories
- DB2, Oracle, SQL Server, Excel, Word, text-based files
- Hundreds of COTS (commercial off-the-shelf software) packages, with export facilities.
- Load, Refresh, and Purge Activities: Must load data, periodically refresh it, and purge old data
- How often?

Data Warehousing Challenges: Extract, Transform, and Load (ETL)

- ETL refers to the processes used to:
  - **Extract** data from homogeneous or heterogeneous data sources
    - Common data-source formats include relational databases, XML, Excel, and flat files.
  - **Transform** the data and store it in a common, standardized format or structure, suitable for querying and analysis.
    - An important function of transformation is *data cleaning*. This operation may take 80% or more of the effort!
  - **Load** the data into the data warehouse

Data Warehousing Challenges: Metadata and approximation

- Metadata Management: Must keep track of source, load time, and other information for all data in the data warehouse.
- Answering Queries Quickly: Approximate answers are often OK.
  - Better to give an approximate answer quickly, than an exact answer *n* minutes later
  - Sampling
  - Snapshot (if the data keeps changing or we want an easily accessible point-in-time result (e.g., end-of-month sales figures))

Data Warehousing Challenges: The need for speed

- Pre-compute and store (materialize) some answers
  - Use a *Data Cube* to store summarized/aggregated data to answer queries, instead of having to go through a much bigger table to find the same answer.
  - The computation is similar in spirit to relational query optimization which is studied in detail in CPSC 404.

OLAP Queries

- OLAP queries are full of groupings and aggregations.
- The natural way to think about such queries is in terms of a ***multidimensional model***, which is an extension of the table model in regular relational databases.
- This model focuses on:
  - a set of numerical ***measures***: quantities that are important for business analysis, like sales, etc.
  - a set of ***dimensions***: entities on which the measures depend on, like location, date, etc.

Multidimensional Data Model

- The main relation, which relates dimensions to a measure via foreign keys, is called the ***fact table***.
  - Recall that a FK in one table refers to a candidate key (and most of the time, the primary key) in another table.
  - The fact table has FKs to the dimension tables.
  - These mappings are essential.
- Each dimension can have additional attributes and an associated ***dimension table***.
  - Attributes can be numeric, categorical, temporal, counts, sums
- Fact tables are *much* larger than dimensional tables.
- There can be multiple fact tables.
  - You may wish to have many measures in the same fact table.

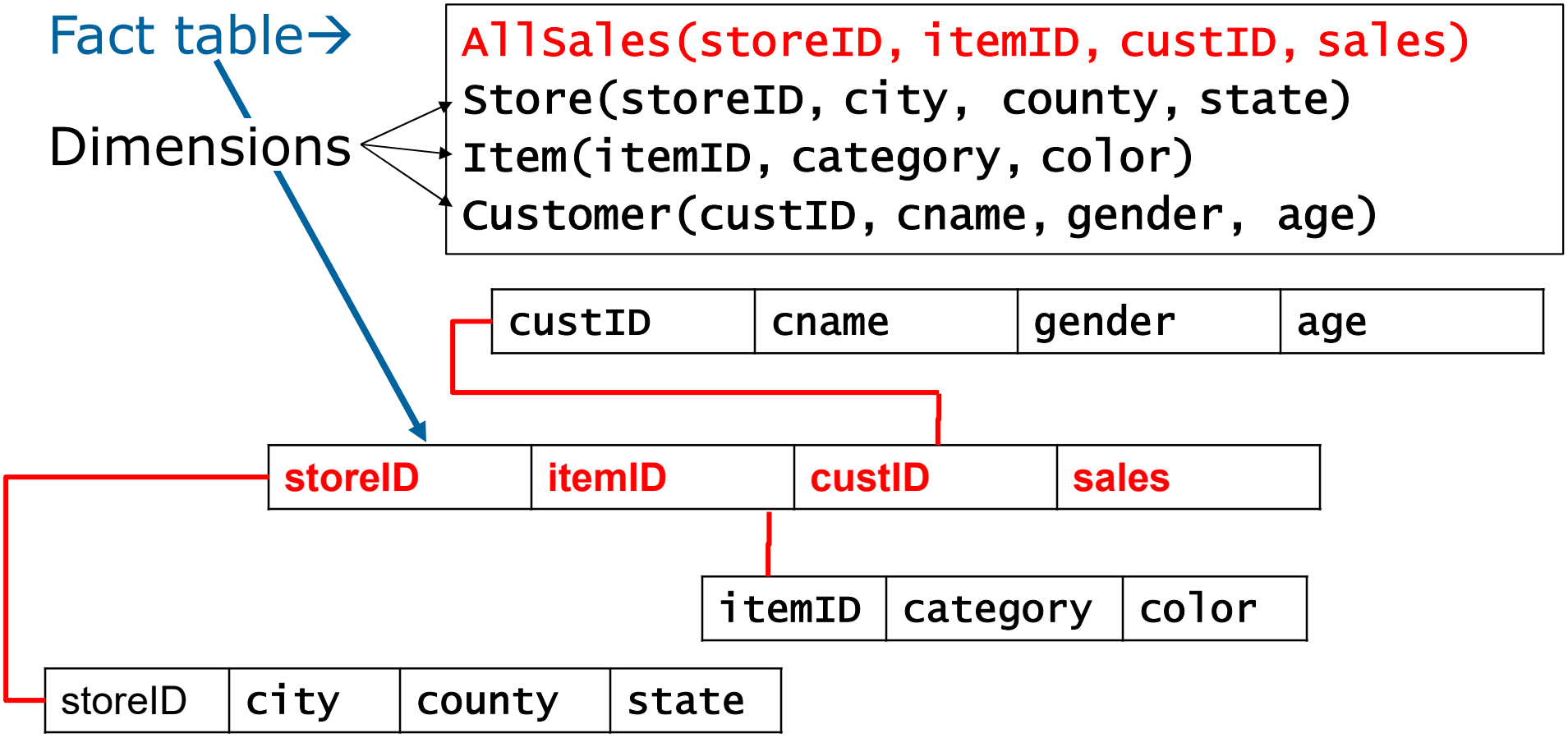
Design Issues

- The schema that is very common in OLAP applications, is called a star schema:
  - one table for the fact, and
  - one table per dimension
- The fact table is in BCNF.
- The dimension tables are not normalized. They are small; updates/inserts/deletes are relatively less frequent. So, redundancy is less important than good query performance

Running Example

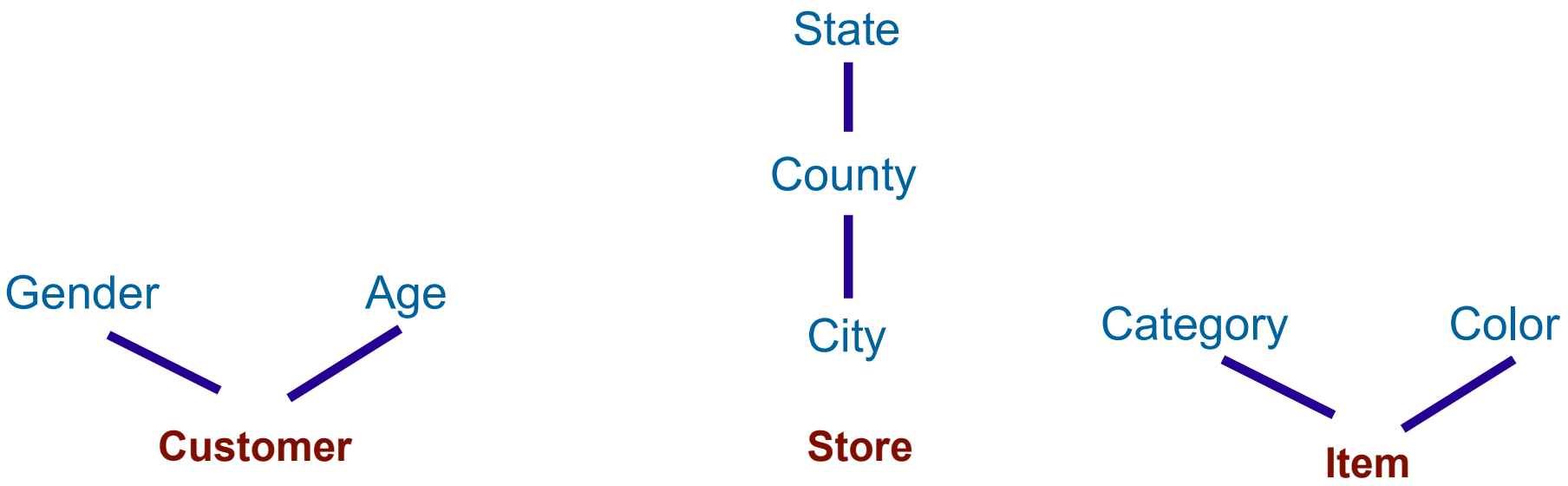
Star Schema – fact table references dimension tables

- Join → Filter → Group → Aggregate



Dimension Hierarchies

For each dimension, the set of values can be organized in a hierarchy:



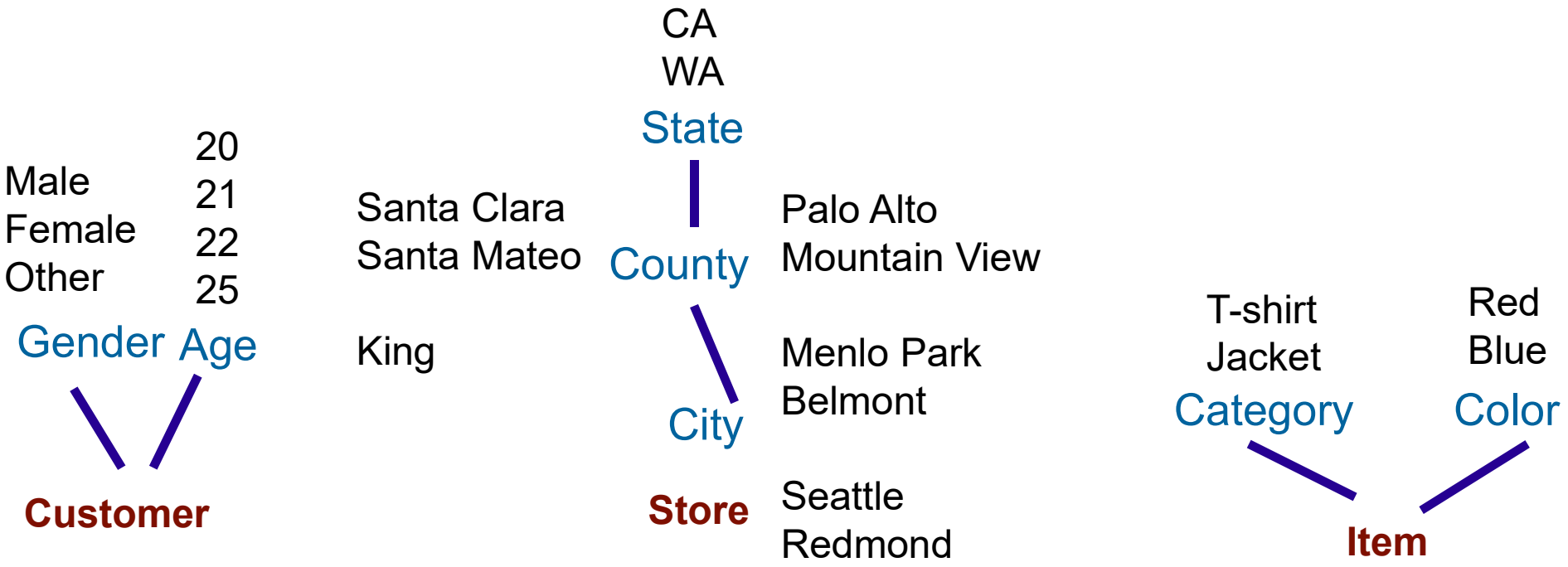
Consider dimensions

Consider building worldwide application about COVID data. What are some dimensions along with values in the hierarchy that you might consider (e.g., you may have a “time” dimension, which would consider a hierarchy of times day→week→month→year)

Time
Location
Variant
Population: gender, age
Severity of disease
Testing type

Running Example (cont.)

```
AllSales(storeID, itemID, custID, sales)
Store(storeID, city, county, state)
Item(itemID, category, color)
Customer(custID, cname, gender, age)
```



Full Star Join

- An example of how to find the *full star join* (or *complete star join*) among 4 tables (i.e., fact table + all 3 of its dimensions) in a Star Schema:
  - Join on the foreign keys

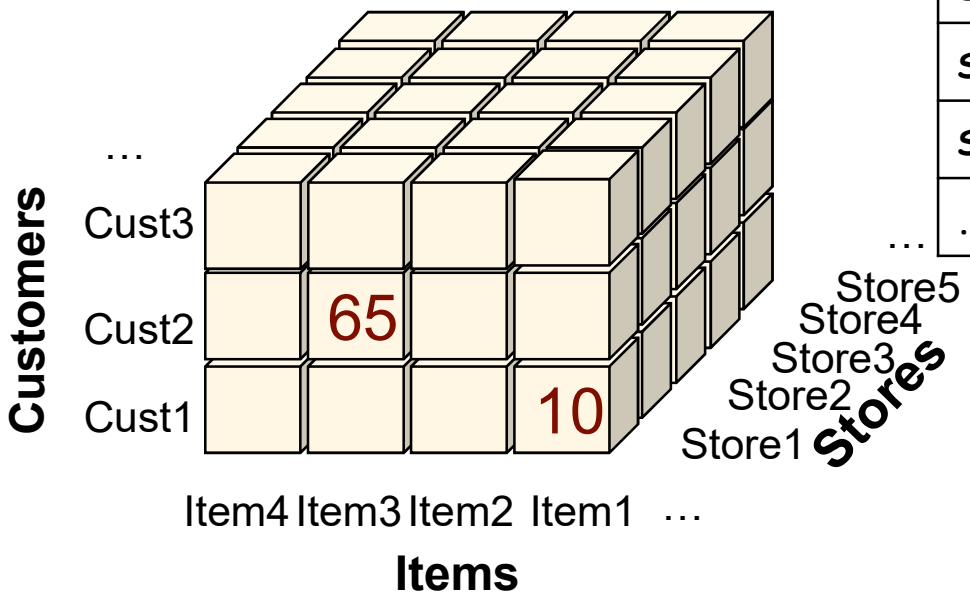
```
SELECT *
FROM    AllSales F, Store S, Item I, Customer C
WHERE  F.storeID = S.storeID and
           F.itemID = I.itemID and
           F.custID = C.custID;
```

- If we join fewer than all dimensions, then we have a *star join*.
- In general, OLAP queries can be answered by computing some or all of the star join, then by filtering, and then by aggregating.

Full Star Join Summarized

Find total sales by store, item, and customer.

```
SELECT storeID, itemID, custID, SUM(sales)
FROM    AllSales F
GROUP BY storeID, itemID, custID;
```

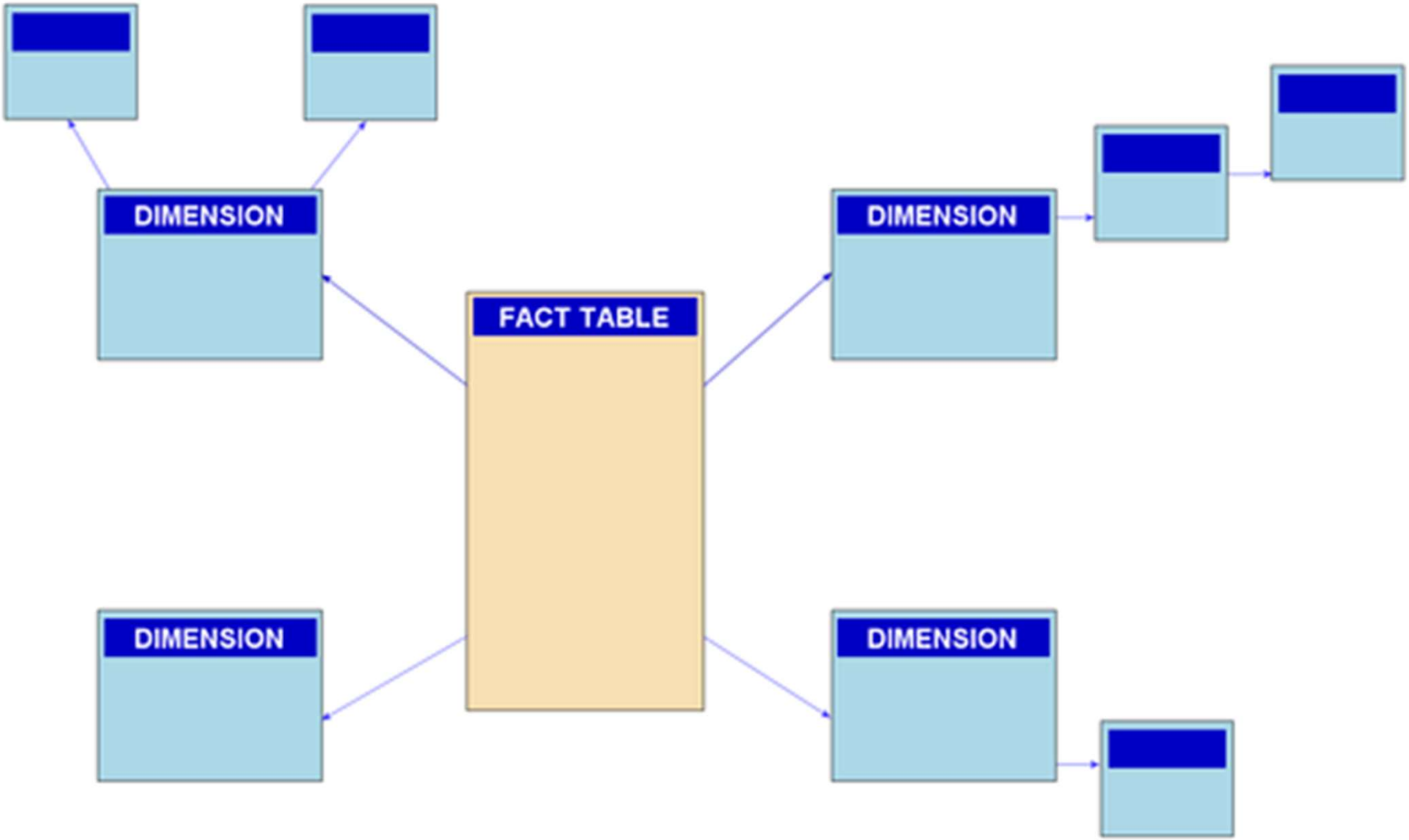


storeID	itemID	custID	Sum (sales)
store1	item1	cust1	10
store1	item3	cust2	65
…	…	…	…

Snowflake Schema

- An alternative organization is a snowflake schema:
  - each dimension is normalized into a set of tables
  - usually, one table per *level* of hierarchy, per dimension
- Example: TIMES table would be split into:
  - TIMES(timeid, date)
  - DWEEK(date, week)
  - DMONTH(date, month)
- Snowflake schema features:
  - Query formulation is inherently more complex (possibly many joins per dimension).
- Neither schema is fully satisfactory for OLAP applications.
- The star schema is more popular, and is gaining interest

Snowflake Schema example



Source: http://en.wikipedia.org/wiki/Snowflake_schema

Star vs. Snowflake

	Star	Snowflake
Ease of maintenance	Has redundant data and hence is less easy to maintain/change	No redundancy, schemas are easier to maintain and change.
Ease of Use	Lower complex query writing; easier to understand	More complex queries and hence less easy to understand
Query Performance	Fewer foreign keys and hence shorter query execution time (faster)	More foreign keys and hence longer query execution time (slower)
Joins	Fewer Joins	More Joins
Dimension table	A single dimension table for each dimension	May have more than one dimension table for each dimension
When to use	Star schema is the default choice	When dimension table is relatively big in size, or we expect a lot of updates
Normalization	Dimension Tables are not Normalized	Dimension Tables are Normalized

Great! Now we have a schema!

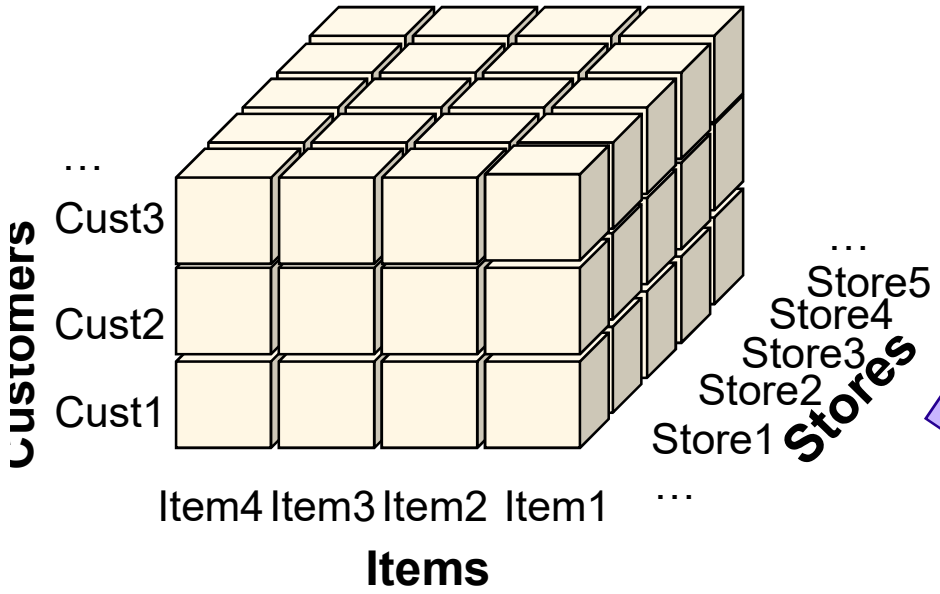
- What can we do with it?

OLAP Queries – Roll-up

- Roll-up allows you to summarize data by:
  - Changing the level of granularity of a particular dimension
  - Dimension reduction

Roll-up Example 1 (Hierarchy)

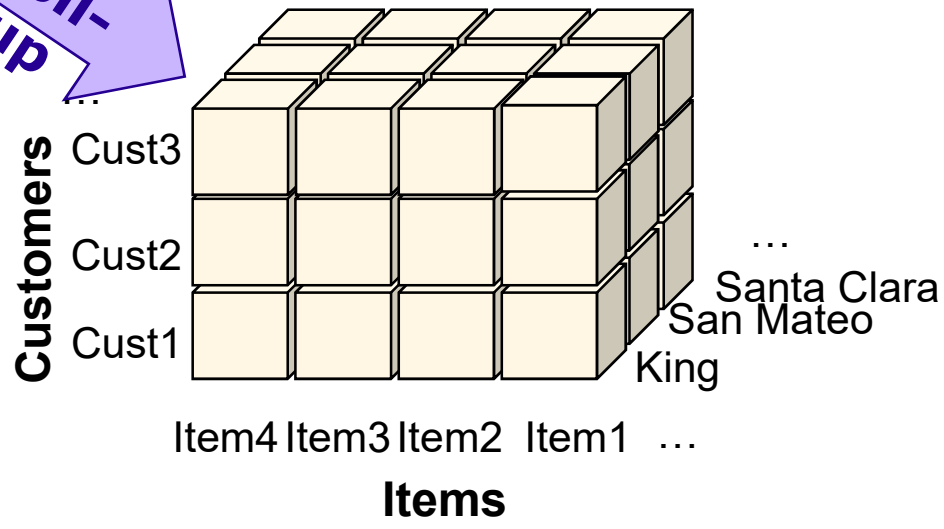
Use Roll-up on total sales by store, item, and customer to find total sales by item and customer for each county.




```
SELECT storeID, itemID, custID,
                SUM(sales)
FROM    AllSales F
GROUP BY storeID, itemID, custID;
```

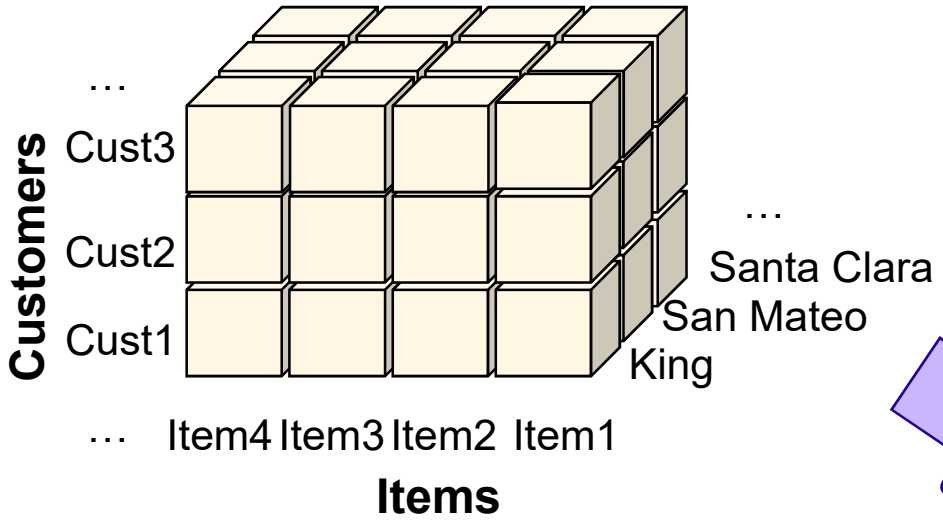
Roll-up

```
SELECT  county, itemID, custID,
                SUM(sales)
FROM     AllSales F, Store S
WHERE  F.storeID = S.storeID
GROUP BY county, itemID, custID;
```



Roll-up Example 2 (Hierarchy)

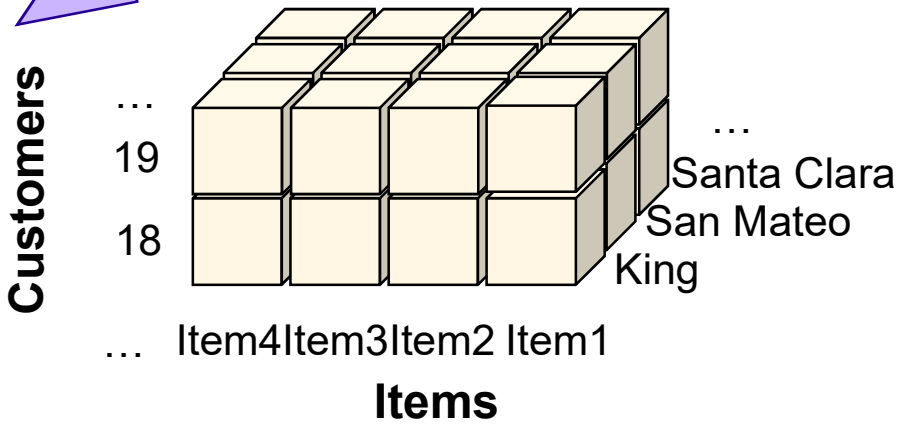
Use Roll-up on total sales by item, customer, and county to find total sales by item, age and county.




```
SELECT county, itemID, custID,
                SUM(sales)
FROM    AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY county, itemID, custID;
```

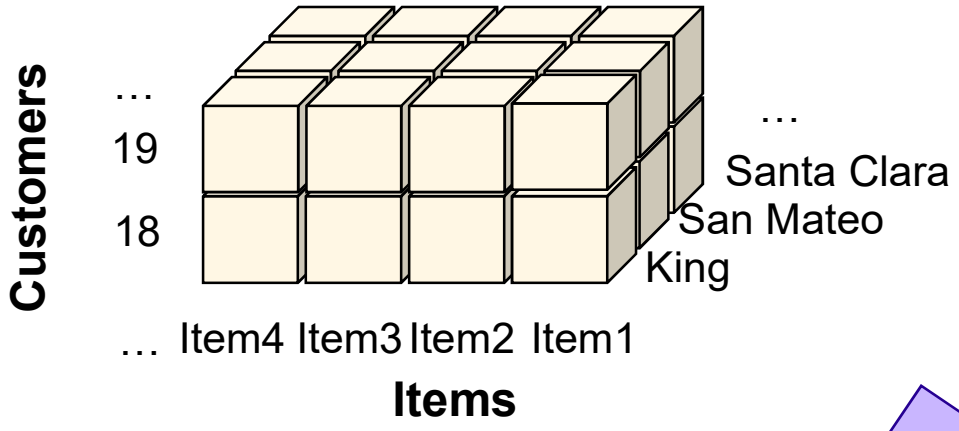
Roll-up

```
SELECT county, itemID, age,
SUM(sales)
FROM    AllSales F, Store S, Customer C
WHERE F.storeID = S.storeID and
F.custID = C.custID
GROUP BY county, itemID, age;
```



Roll-up Example 3 (Dimension)

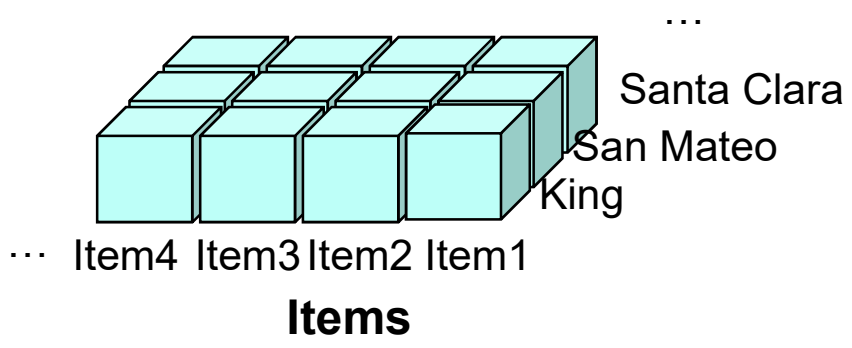
Use Roll-up on total sales by item, age and county to find total sales by item for each county.




```
SELECT county, itemID, age,
       SUM(sales)
FROM   AllSales F, Store S,
       Customer C
WHERE F.storeID = S.storeID AND
       F.custID = C.custID
GROUP BY county, itemID, age;
```

Roll-up

```
SELECT county, itemID, SUM(sales)
FROM   AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY county, itemID;
```

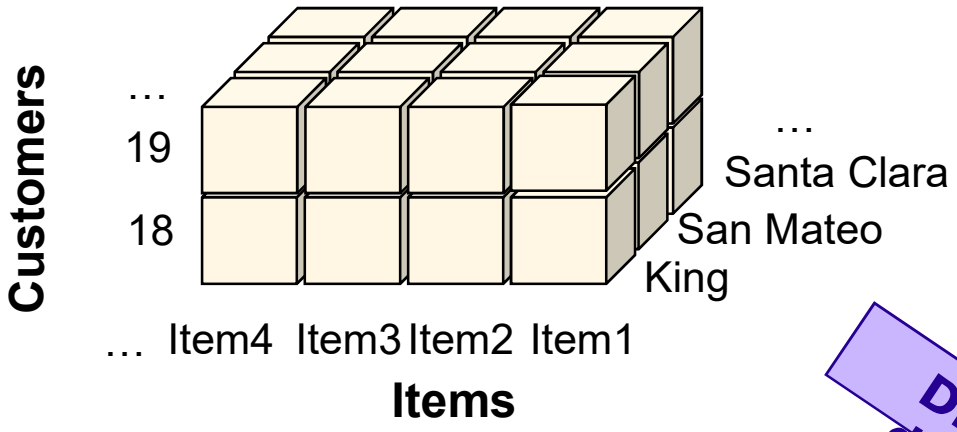


OLAP Queries – Drill-down

- Drill-down: reverse of roll-up
  - From higher level summary to lower level summary (i.e., we want more detailed data)
  - Introducing new dimensions

Drill-down Example 1 (Hierarchy)

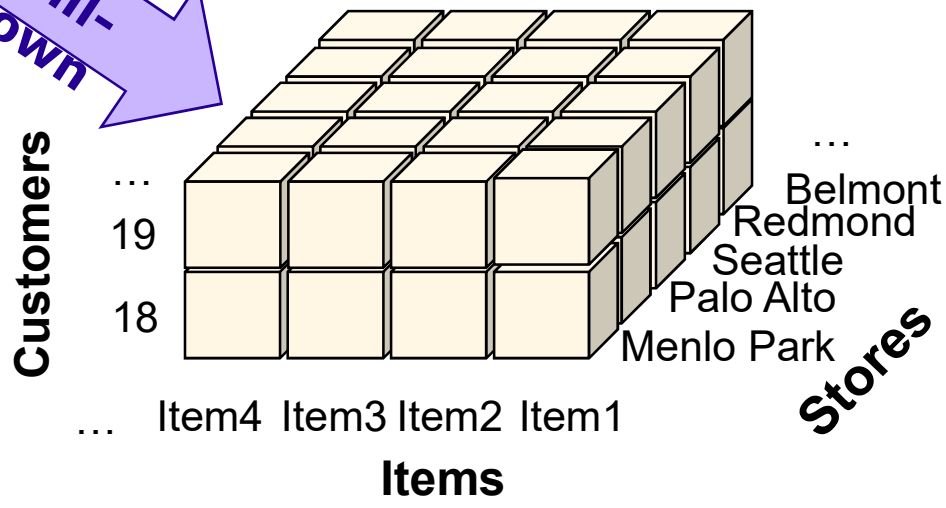
Use Drill-down on total sales by item and age for each county to find total sales by item and age for each city.




```
SELECT county, itemID, age,
                SUM(sales)
FROM    AllSales F, Store S,
                Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID
GROUP BY county, itemID, age;
```

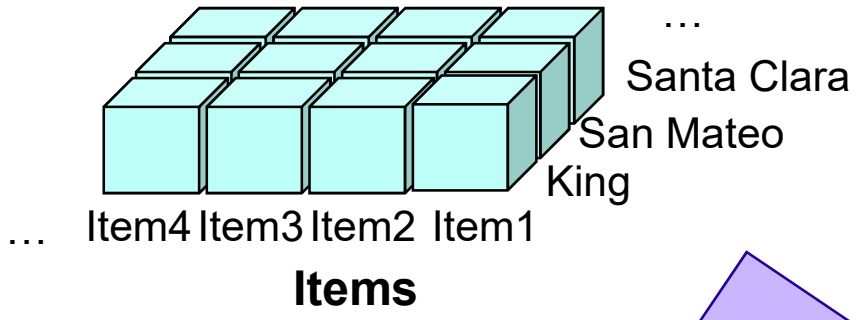
Drill-down

```
SELECT city, itemID, age,
SUM(sales)
FROM    AllSales F, Store S,
Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID
GROUP BY city, itemID, age;
```



Drill-down Example 2 (Dimension)

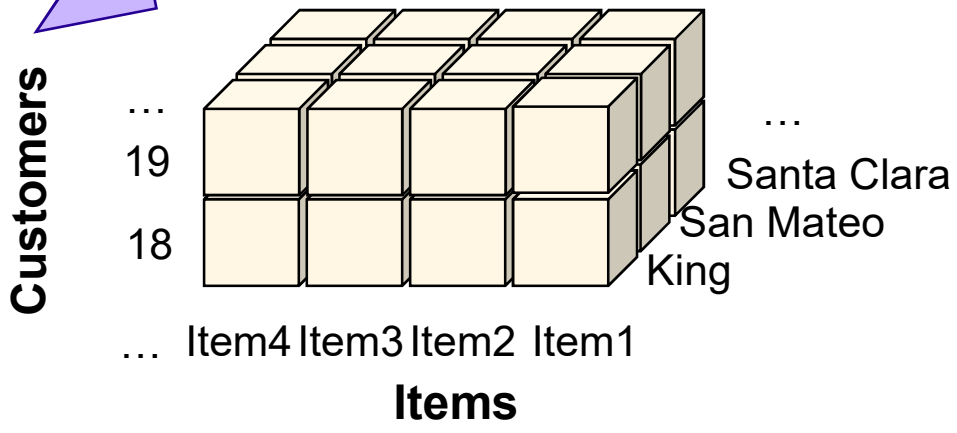
Use Drill-down on total sales by item and county to find total sales by item and age for each county.




```
SELECT county, itemID, SUM(sales)
FROM    AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY county, itemID;
```

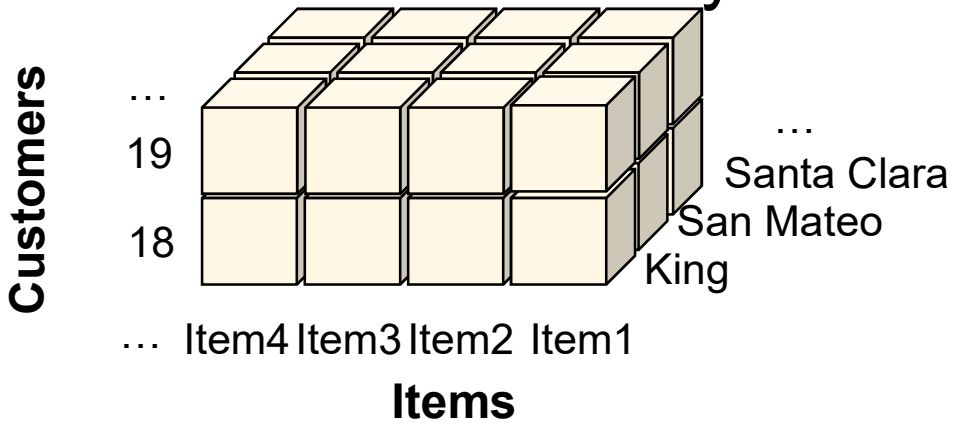
Drill-down

```
SELECT county, itemID, age,
              SUM(sales)
FROM    AllSales F, Store S,
              Customer C
WHERE F.storeID = S.storeID AND
              F.custID = C.custID
GROUP BY county, itemID, age;
```



OLAP Queries – Slicing

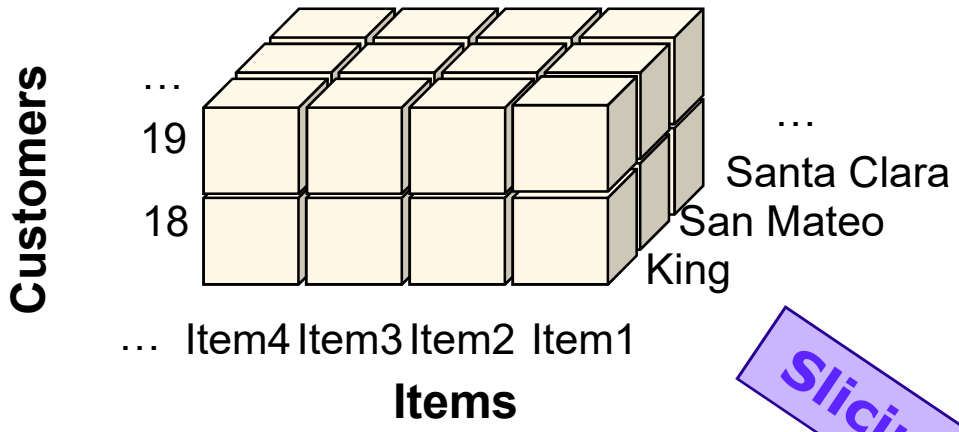
- The slice operation produces a slice of the cube by picking a specific value for one of the dimensions.
- To start our example, let's specify:
  - Total sales by item and age for each county




```
SELECT county, itemID, age,
                SUM(sales)
FROM    AllSales F, Store S,
                Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID
GROUP BY county, itemID, age;
```

Slicing Example 1

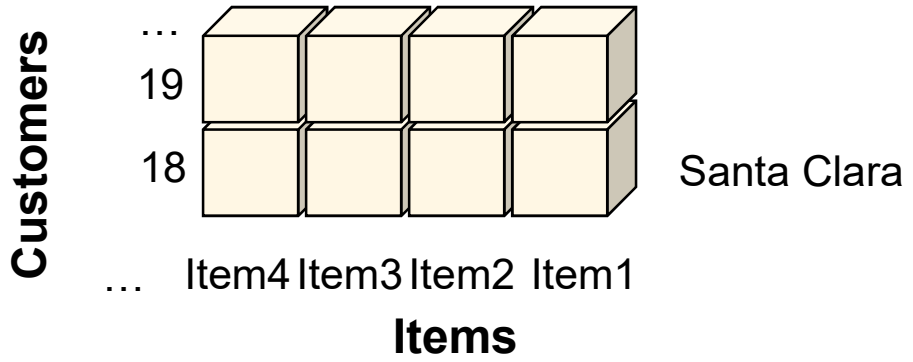
Use Slicing on total sales by item and age for each county to find total sales by item and age for Santa Clara.




```
SELECT county, itemID, age,
                SUM(sales)
FROM    AllSales F, Store S,
Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID
GROUP BY county, itemID, age;
```

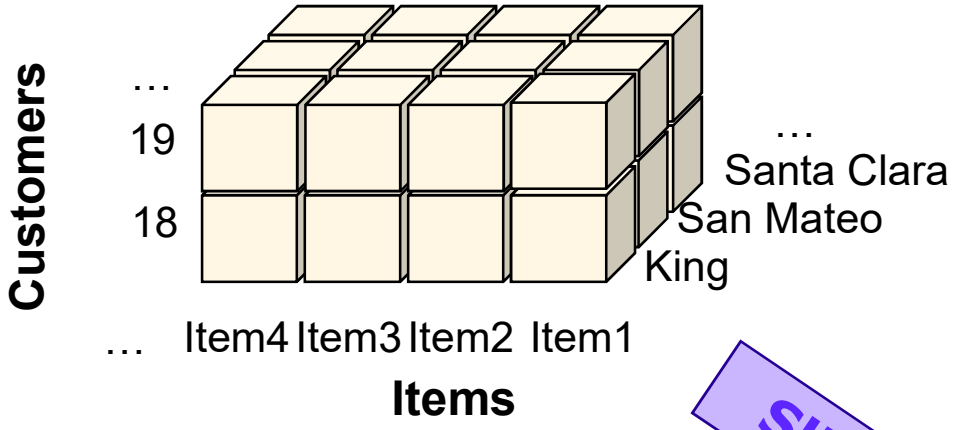
Slicing

```
SELECT itemID, age, SUM(sales)
FROM    AllSales F, Store S,
Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID AND
                S.county = 'Santa Clara'
GROUP BY itemID, age;
```



Slicing Example 2

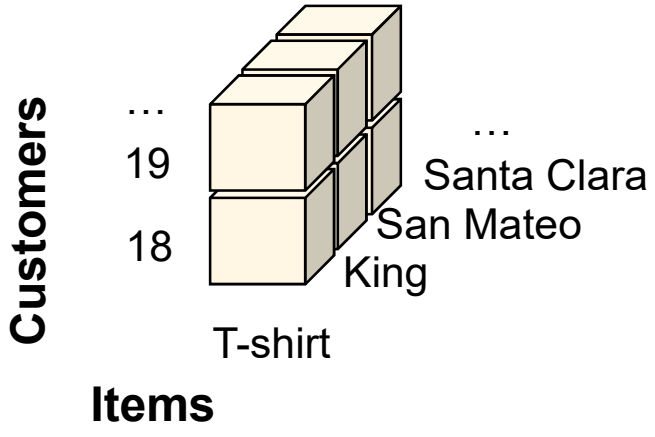
Use Slicing on total sales by item and age for each county to find total sales by age and county for T-shirts.




```
SELECT county, itemID, age,
                SUM(sales)
FROM    AllSales F, Store S,
                Customer C
WHERE F.storeID = S.storeID AND
                F.custID = C.custID
GROUP BY county, itemID, age;
```

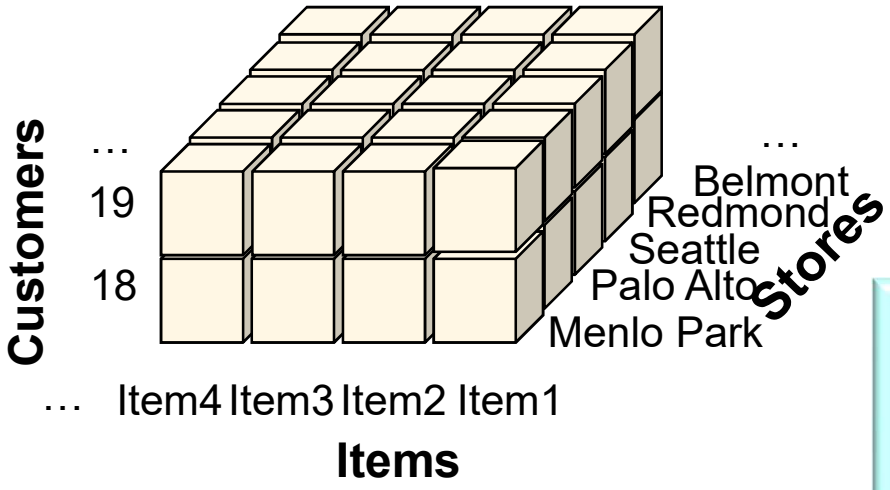
Slicing

```
SELECT county, age, SUM(sales)
FROM    AllSales F, Store S, Customer C, Item I
WHERE F.storeID = S.storeID AND
                F.custID = C.custID AND
                F.itemID = I.itemID AND
                category  = 'Tshirt'
GROUP BY county, age;
```



OLAP Queries – Dicing

- The dice operation produces a sub-cube by picking specific values for multiple dimensions.
- To start our example, let's specify:
  - Total sales by age, item, and city




```
SELECT city, itemID, age, SUM(sales)
FROM   AllSales F, Store S, Customer C
WHERE F.storeID = S.storeID AND
            F.custID = C.custID
GROUP BY city, itemID, age;
```

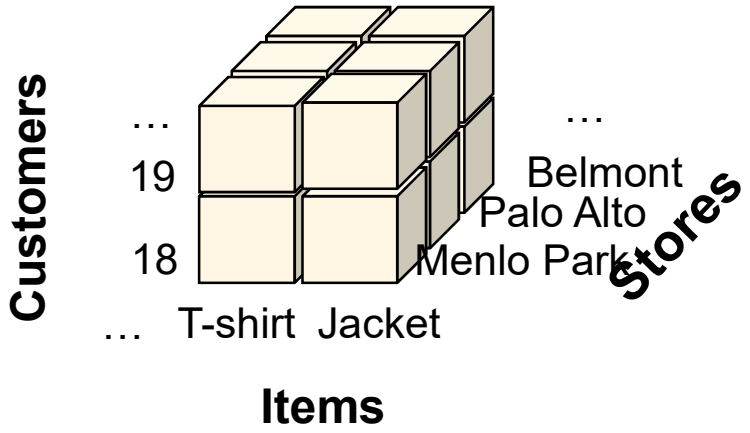
Dicing Example 1

Use Dicing on total sales by age, item, and city to find total sales by age, category, and city for red items in the state of California (CA).


```
SELECT city, itemID, age, SUM(sales)
FROM   AllSales F, Store S, Customer C
WHERE F.storeID = S.storeID AND
            F.custID = C.custID
GROUP BY city, itemID, age;
```

Dicing

```
SELECT  category, city, age, SUM(sales)
FROM AllSales F, Store S, Customer C, Item I
WHERE  F.storeID = S.storeID AND
            F.custID = C.custID AND
            F.itemID = I.itemID AND
            color = 'red' AND state = 'CA'
GROUP BY category, city, age;
```

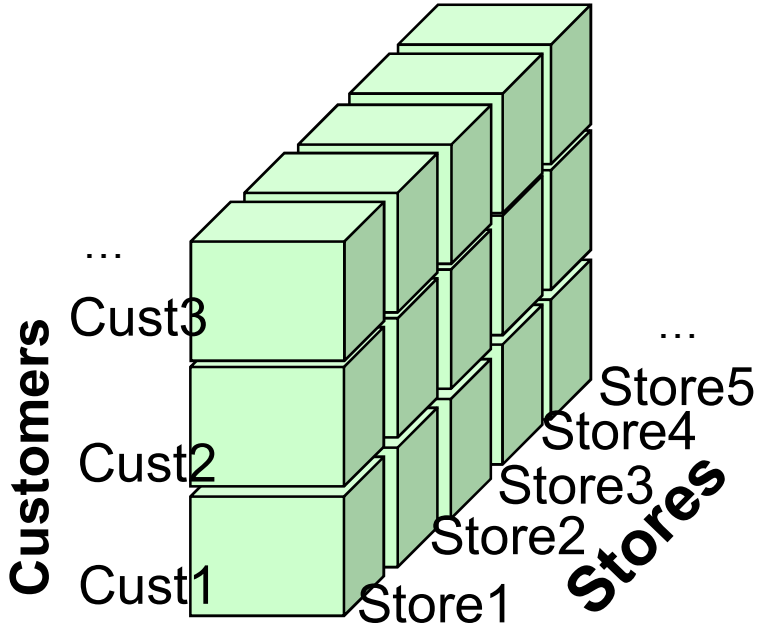


OLAP Queries – Pivoting

- Pivoting is a visualization operation that allows an analyst to rotate the cube in space in order to provide an alternative presentation of the data.

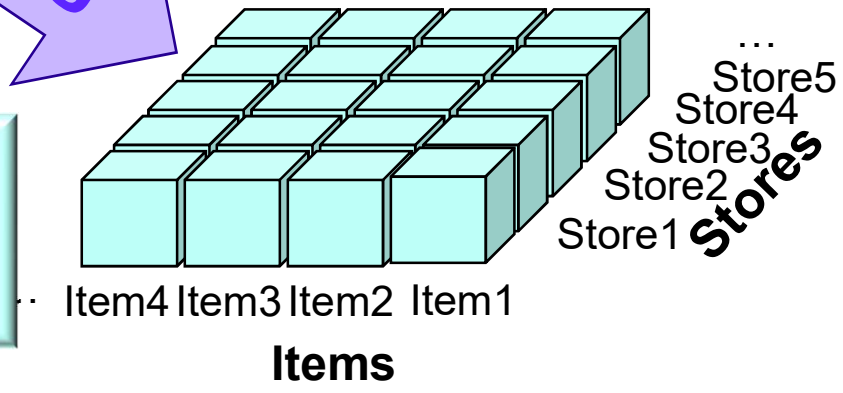
Pivoting Example 1

From total sales by store and customer, pivot to find total sales by item and store.




```
SELECT storeID, custID, sum(sales)
FROM  AllSales
GROUP BY storeID, custID;
```

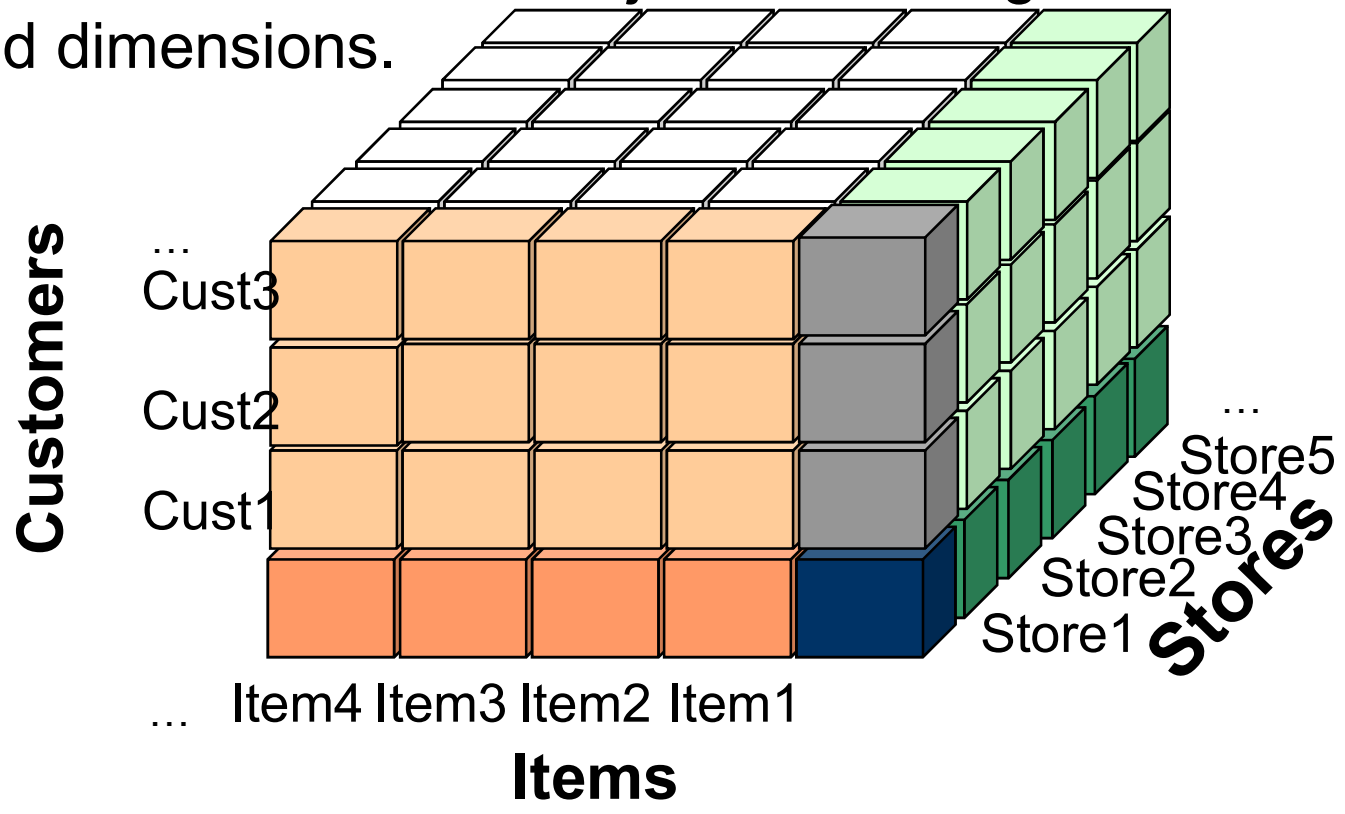
Pivoting




```
SELECT storeID, itemID, sum(sales)
FROM  AllSales
GROUP BY storeID, itemID;
```

Data Cube

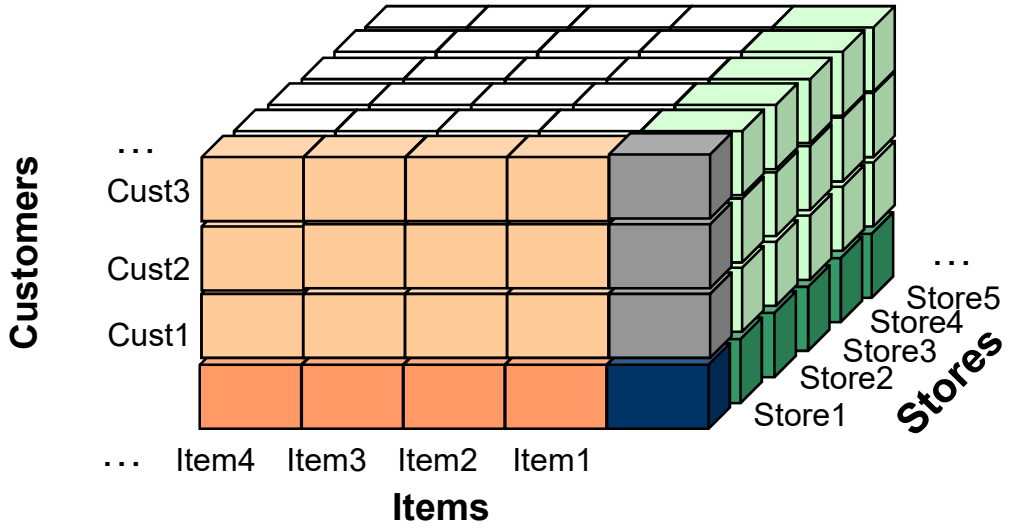
- A *data cube* is a *k*-dimensional object containing both fact data and dimensions.



- A cube contains pre-calculated, aggregated, summary information to yield fast queries.

Data Cube (cont.)

- The small, individual blocks in the multidimensional cube are called *cells*, and each cell is uniquely identified by the *members* from each dimension.



- The cells contain a *measure group*, which consists of one or more numeric *measures*. These are facts (or aggregated facts). An example of a measure is the dollar value in sales for a particular product

Estimating size of a cube

- Consider a car sales cube with dimensions of models, colours, and years
- With 2 models, 2 colours, and 2 years, how many tuples are there in the cube, ***assuming*** there is data for every combo. of (model, year, color)?
- 2 x 2 x 2 tuples in the group-by (model, year, color).
- Each of model, year, colour can independently be "All".
- (2+1) x (2+1) x (2+1) = 3 x 3 x 3 combos in all.
- Formally, consider a cube with n dimensions with dimension i having C_i values. The size of the cube is $\prod_{i=1}^{n}(Ci + 1)$

Up until now, we've assumed that all cube entries have data

- A cube is *dense* if it has data for all combinations of dimension attributes
  - In practice, a cube is dense if > p% combinations are present for some suitable threshold of p
- Otherwise it is *sparse*

Estimating the size of a sparse cube

- Sparse cube size ≈ dense cube size × sparsity factor
- E.g., suppose car sales cube from previous example has a sparsity factor of 10%. Then the estimated size of sparse car sales cube = 10% of 1386 or 139 tuples
- We care about estimating cube size to help guide how we (1) compute the most “useful” subset of a cube or (2) best compute the full cube – both coming up shortly

The CUBE Operator

Generalizing the previous example, if there are *k* dimensions, we have 2^k possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions. A CUBE BY operator generates that.

- It's equivalent to rolling up AllSales on all eight subsets of the set {storeID, itemID, custID }.
- Each roll-up corresponds to an SQL query of the form:

```
SELECT SUM (sales)
FROM   AllSales S
GROUP BY grouping-list
```

Lots of research on optimizing the CUBE operator!

The CUBE Operator (cont.)

- Roll-up, Drill-down, Slicing, Dicing, and Pivoting operations are expensive.
- SQL:1999 extended GROUP BY to support CUBE (and ROLLUP).
- GROUP BY CUBE provides efficient computation of multiple granularity aggregates by sharing work (e.g., passes over fact table, previously computed aggregates).

WITH CUBE

Not implemented in MySQL

```
Select dimension-attrs, aggregates
From   tables
Where  conditions
Group By dimension-attrs With Cube
```

```
SELECT storeID, itemID, custID,
       sum(sales)
FROM   AllSales
GROUP BY storeID, itemID, custID WITH
CUBE
```

storeID	itemID	custID	Sum
store1	item1	cust1	10
store1	item1	Null	70
store1	Null	cust1	145
store1	Null	Null	325
Null	item1	cust1	10
Null	item1	Null	135
Null	Null	cust1	670
Null	Null	Null	3350

.....

Finding Answers Quickly

- Large datasets and complex queries mean that we'll need improved querying capabilities
  - Materializing views
  - Finding Top N Queries
  - Using Online aggregation

Queries over Views

Reminder: use a view as follows:

View

```
Create view TshirtSales AS
SELECT category, county, age, sales
FROM    AllSales F, Store S, Customer C, Item I
WHERE F.storeID = S.storeID AND F.custID = C.custID AND
              F.itemID = I.itemID ANDcategory  = 'Tshirt'
```

Query

```
SELECT category, county, age, SUM(sales)
From      TshirtSales
GROUP BY category, county, age;
```

Materializing Views

- Decision support activities require queries against complex view definitions to be computed quickly.
- Sophisticated optimization and evaluation techniques are not enough since OLAP queries are typically aggregate queries that require joining huge tables.
- Pre-computation is essential for interactive response times.
- A view whose tuples are stored in the database is said to be materialized.

Issues in View Materialization: Which views should we materialize?

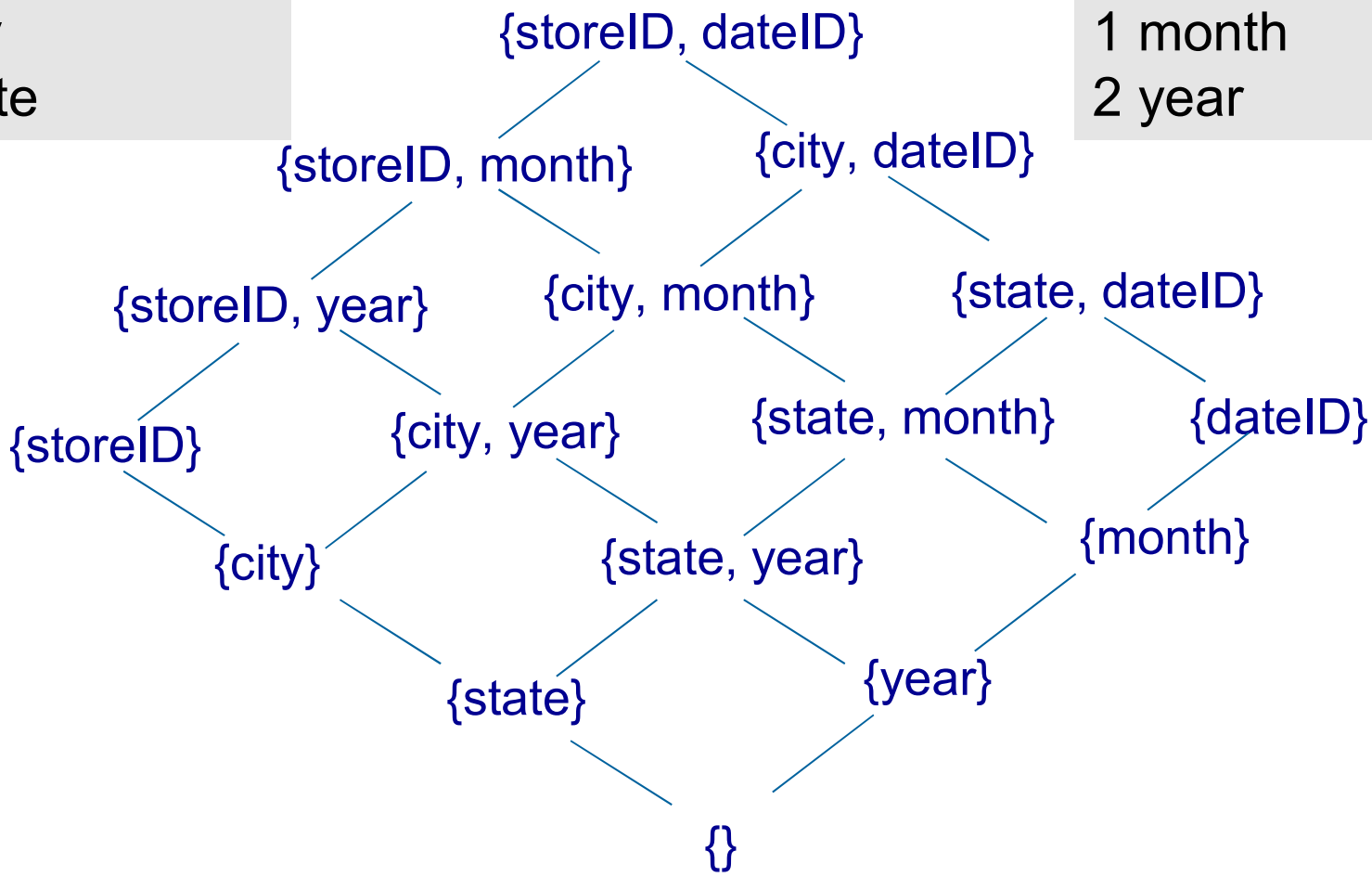
- Which views should we materialize?
- Based on size estimates for the views, suppose there is space for *k* views to be materialized. Which ones should we materialize?
- The goal is to materialize a small set of carefully chosen views to answer most of the queries quickly.
- **Fact:** Selecting *k* views to materialize such that the average time taken to evaluate all views of a lattice is minimized is a NP-hard problem.

The Exponential Explosion of Views

- Assume that we have two dimensions, each with a hierarchy

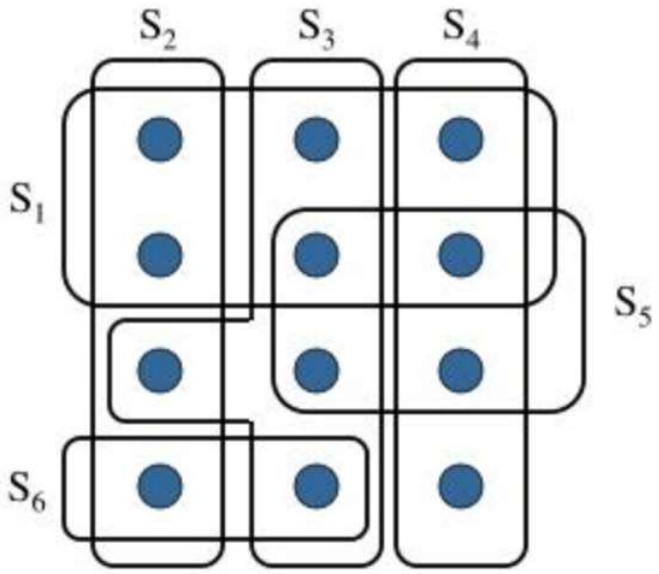
Store dimension
0 storeID
1 city
2 state

Calendar dimension
0 dateID
1 month
2 year



Maximum Coverage Problem Example

Given 12 ground facts/elements, 6 subsets, and a value for *k*, find the *k* subsets (e.g., *k* = 3) that between them cover as many ground elements as possible.

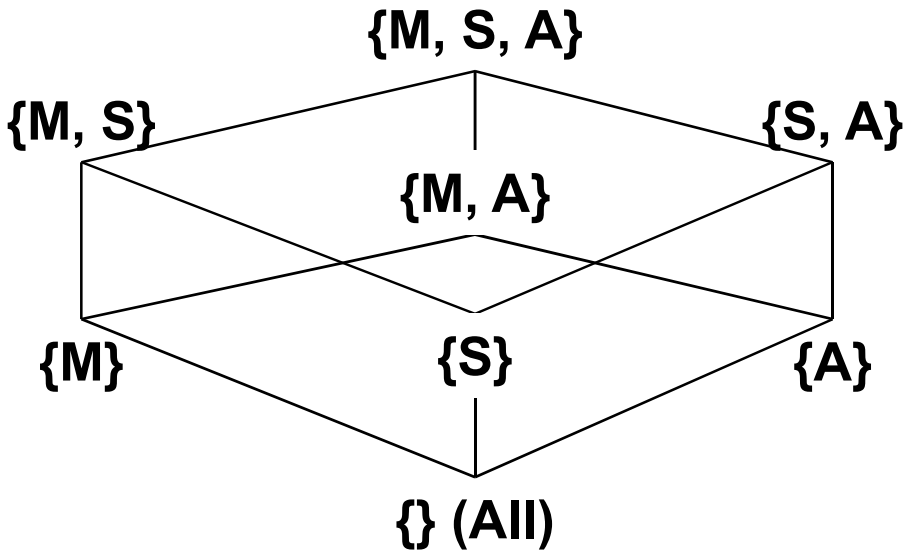


Difference between a NP-hard problem and a problem that you solve efficiently (polynomial time) is like the difference between solving a Sudoku puzzle vs. checking whether a given solution is valid.

Maximum Coverage problem has a structure similar to finding the top *k* views. We can find approximately optimal solutions quickly for both.

Let's revisit the student table: Student(snum,sname,major,standing,age)

- We can aggregate the student table based on major, standing, or age



Group by all 3 – any ordering gives you same # of tuples

```
SELECT major, standing, age, count(*)
FROM         student
GROUP BY major, standing, age
ORDER BY major, standing, age
```

MAJOR	ST	AGE	COUNT(*)
Accounting	JR	19	1
Animal Science	FR	18	1
Architecture	SR	22	1
Civil Engineering	SR	21	1
Computer Engineering	FR	18	1
Computer Engineering	SR	19	1
Computer Science	JR	18	1
Computer Science	JR	20	1
Computer Science	SO	17	1
Computer Science	SO	19	1
Economics	JR	20	1
Education	SR	21	1
Electrical Engineering	FR	17	2
English	SR	21	1
Finance	FR	18	2
History	SR	20	1
Kinesiology	SO	19	1
Law	JR	20	1
Mechanical Engineering	SO	19	1
Psychology	JR	20	1
Psychology	SO	18	1
Veterinary Medicine	SR	21	1

22 rows selected.

```
SELECT age, standing, major, count(*)
FROM student
GROUP BY age, standing, major
ORDER BY age, standing, major
```

AGE	ST	MAJOR	COUNT(*)
17	FR	Electrical Engineering	2
17	SO	Computer Science	1
18	FR	Animal Science	1
18	FR	Computer Engineering	1
18	FR	Finance	2
18	JR	Computer Science	1
18	SO	Psychology	1
19	JR	Accounting	1
19	SO	Computer Science	1
19	SO	Kinesiology	1
19	SO	Mechanical Engineering	1
19	SR	Computer Engineering	1
20	JR	Computer Science	1
20	JR	Economics	1
20	JR	Law	1
20	JR	Psychology	1
20	SR	History	1
21	SR	Civil Engineering	1
21	SR	Education	1
21	SR	English	1
21	SR	Veterinary Medicine	1
22	SR	Architecture	1

22 rows selected.

Group by 2 option 1: Major, Standing

```
SELECT major, standing, count(*)
FROM student
GROUP BY major, standing
ORDER BY major, standing

MAJOR                     ST   COUNT(*)
------------------------- -- ----------
Accounting                JR          1
Animal Science            FR          1
Architecture              SR          1
Civil Engineering         SR          1
Computer Engineering      FR          1
Computer Engineering      SR          1
Computer Science          JR          2
Computer Science          SO          2
Economics                 JR          1
Education                 SR          1
Electrical Engineering    FR          2
English                   SR          1
Finance                   FR          2
History                   SR          1
Kinesiology               SO          1
Law                       JR          1
Mechanical Engineering    SO          1
Psychology                JR          1
Psychology                SO          1
Veterinary Medicine       SR          1

20 rows selected.
```

Group by 2 option 2: Standing, Age

```
SELECT standing, age, count(*)
FROM student
GROUP BY standing, age
ORDER BY standing, age

ST  AGE     COUNT(*)
-- ---------- ----------
FR   17            2
FR   18            4
JR   18            1
JR   19            1
JR   20            4
SO   17            1
SO   18            1
SO   19            3
SR   19            1
SR   20            1
SR   21            4
SR   22            1

12 rows selected.
```

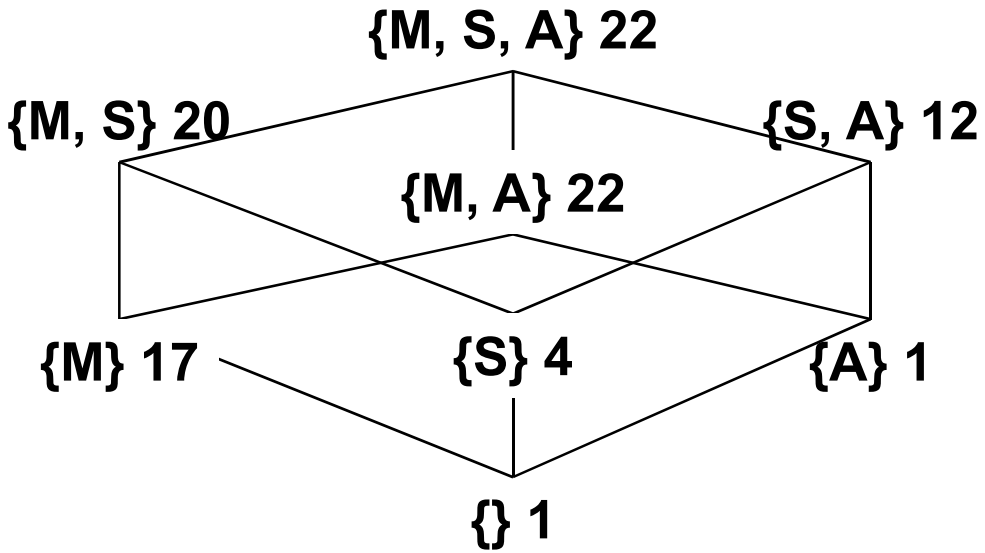
Group by 2 option 3: Major, Age

```
SELECT major, age, count(*)
FROM student
GROUP BY major, age
ORDER BY major, age
```

MAJOR	AGE	COUNT(*)
Accounting	19	1
Animal Science	18	1
Architecture	22	1
Civil Engineering	21	1
Computer Engineering	18	1
Computer Engineering	19	1
Computer Science	17	1
Computer Science	18	1
Computer Science	19	1
Computer Science	20	1
Economics	20	1
Education	21	1
Electrical Engineering	17	2
English	21	1
Finance	18	2
History	20	1
Kinesiology	19	1
Law	20	1
Mechanical Engineering	19	1
Psychology	18	1
Psychology	20	1
Veterinary Medicine	21	1

22 rows selected.

In computing costs to query, we consider the # of tuples that you have to look at



The # is the # of tuples

Assuming all of the views were materialized, executing the query

```
SELECT standing, count(*)
FROM student
GROUP BY standing
```

would cost 4 because that's the cheapest way to access the necessary tuples

We can also use {Major, Standing} to compute {Standing} for a cost of 20

```
SELECT major, standing, count(*)
FROM student
GROUP BY major, standing
ORDER BY major, standing
```

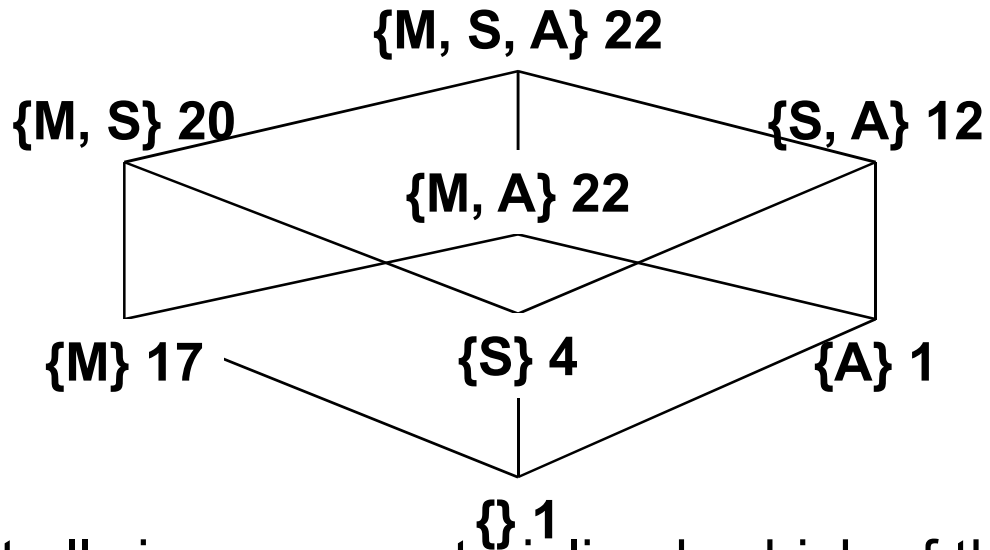
```
SELECT standing, count(*)
FROM student
GROUP BY standing
ORDER BY standing
```

```
MAJOR                     ST   COUNT(*)
------------------------- -- ----------
Accounting                JR           1
Animal Science            FR           1
Architecture              SR           1
Civil Engineering         SR           1
Computer Engineering      FR           1
Computer Engineering      SR           1
Computer Science          JR           2
Computer Science          SO           2
Economics                 JR           1
Education                 SR           1
Electrical Engineering    FR           2
English                   SR           1
Finance                   FR           2
History                   SR           1
Kinesiology               SO           1
Law                       JR           1
Mechanical Engineering    SO           1
Psychology                JR           1
Psychology                SO           1
Veterinary Medicine       SR           1

20 rows selected.
```

```
ST   COUNT(*)
-- ----------
SR           7
SO           5
FR           6
JR           6
```

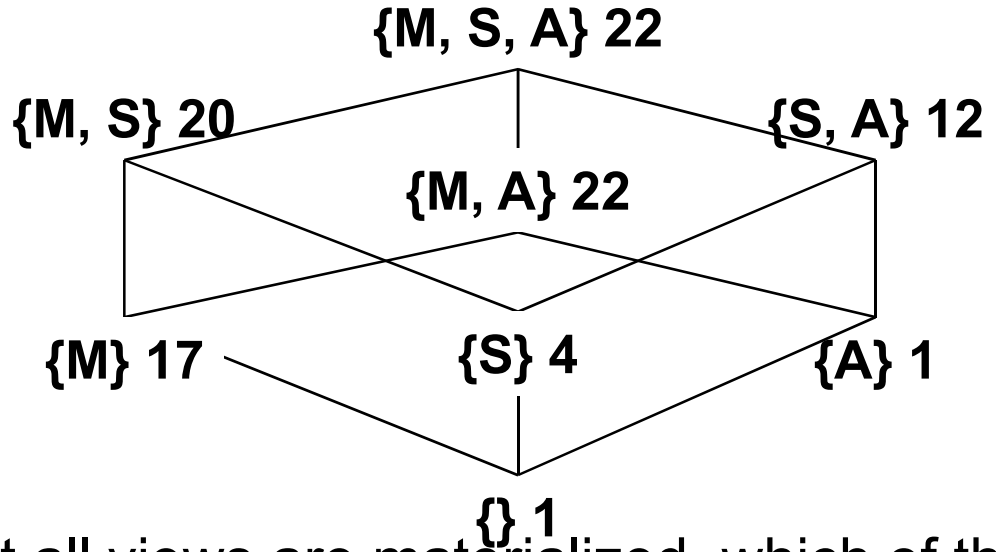
Clicker question: which views can be used for which query? Part 1



Assume that all views are materialized, which of the following views can answer a query about Standing (S)?

A. {M,S,A}
B. {M,A}
C. {}
D. All of the above
E. None of the above

Clicker question: which views can be used for which query? Part 1

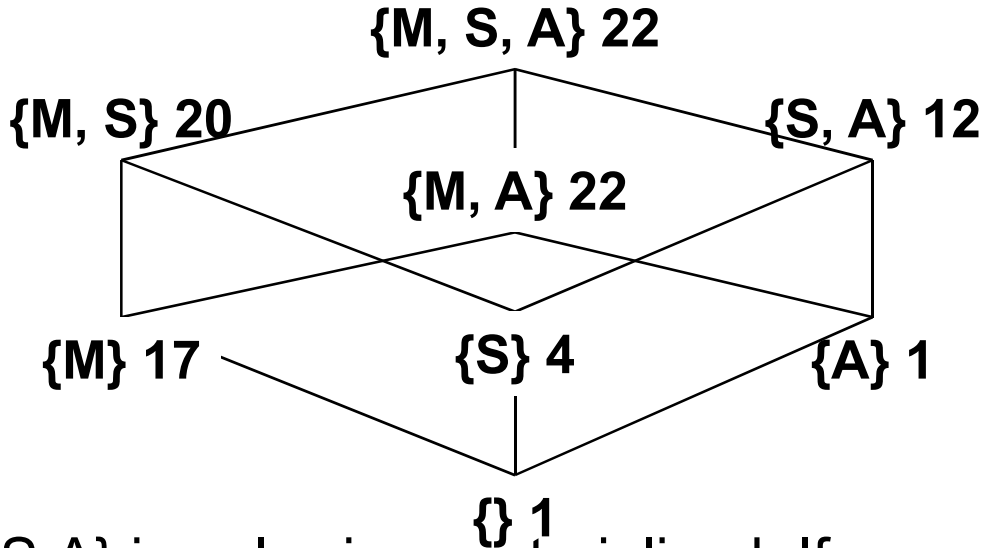


Assume that all views are materialized, which of the following views can answer a query about Standing (S)?

A. {M,S,A}
B. {M,A}
C. {}
D. All of the above
E. None of the above

The correct answer is A; any thing that aggregates S can be used.

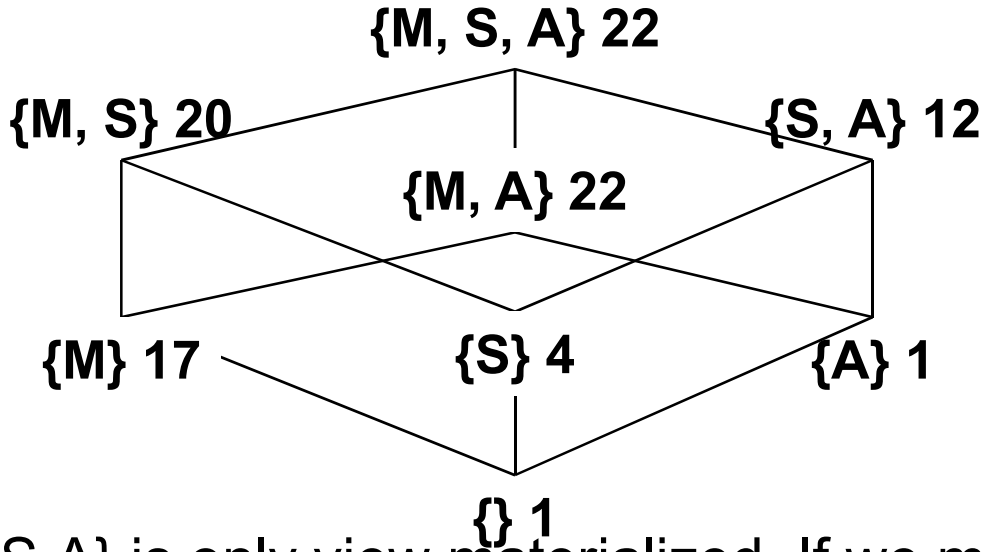
Clicker question: which views can be used for which query? Part 2



Assume {M,S,A} is only view materialized. If we materialize {S,A}, queries over which lattice parts could be answered using {S,A}?

A. {S,A}
B. {{M,S,A}{S,A}
C. {{S,A},{S},{A},{}}
D. None of the above

Clicker question: which views can be used for which query? Part 2

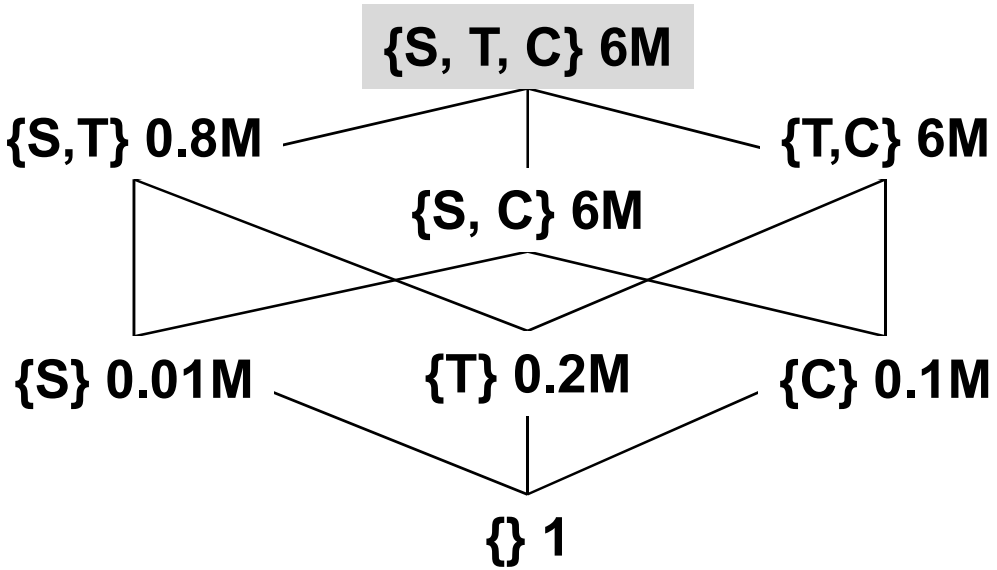


Assume {M,S,A} is only view materialized. If we materialize {S,A}, queries over which lattice parts could be answered using {S,A}?

A. {S,A}
B. {{M,S,A}{S,A}
C. {{S,A},{S},{A},{}}
D. None of the above

The correct answer is C; {S,A} can be used to answer anything that uses an aggregation on {S,A}

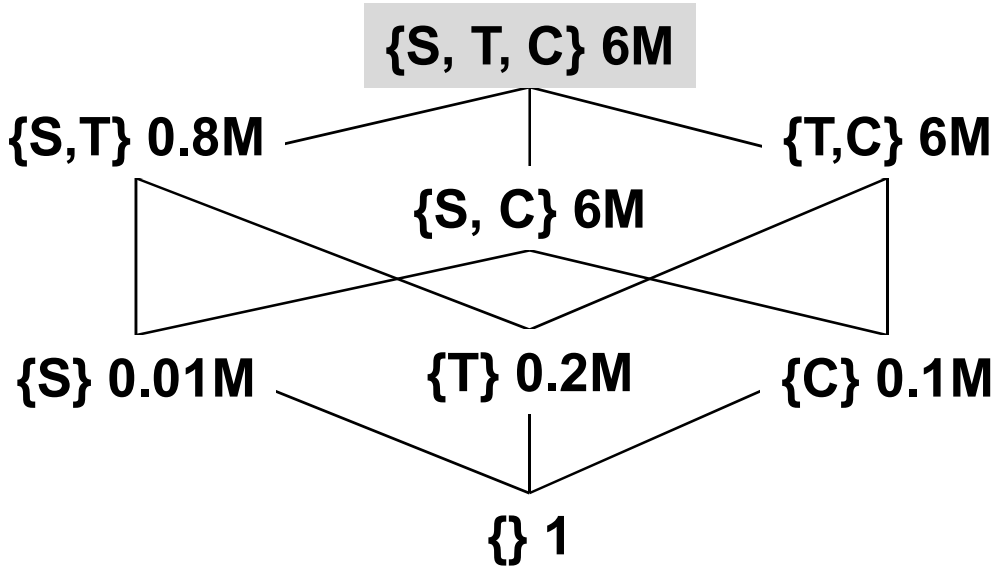
The HRU algorithm for materializing views



- The question is: which views can we materialize to answer queries the cheapest?
- Initially, only the top-most view is materialized

HRU [Harinarayan, Rajaraman, and Ullman, 1996]—SIGMOD Best Paper award—is a greedy algorithm that does not guarantee an optimal solution, though it usually produces a good solution. This solution is a good trade-off in terms of the space used and the average time to answer an OLAP query.

Benefit of Materializing a View



Intuitively, for each view under consideration, determine (1) if it can be used to answer a query and (2) if so, how much does it save?

Formally:

Define the benefit (savings) of view v relative to S as **B(v,S)**.

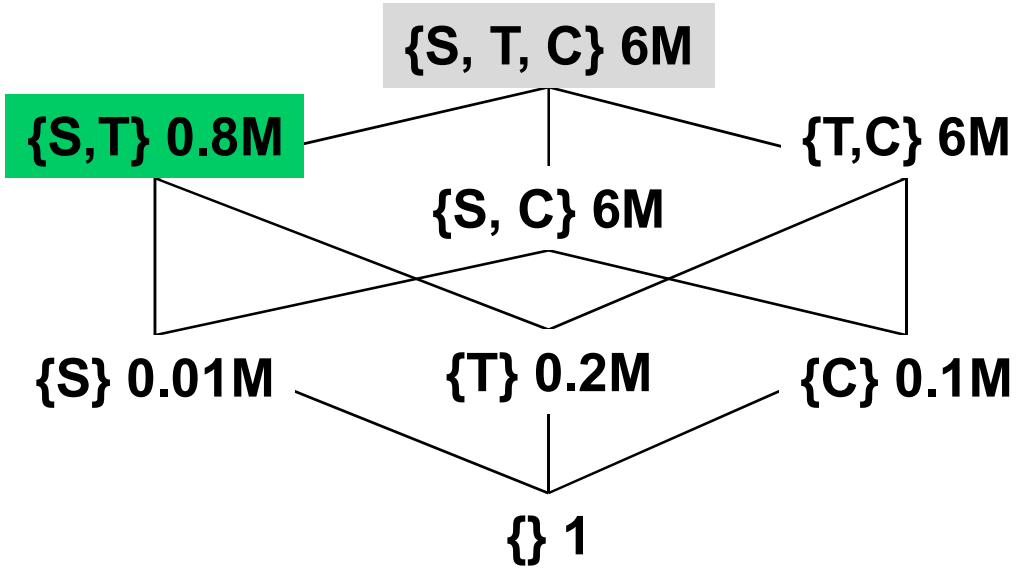
B(v, S) = 0
For each w ≦ v
    u = view of least cost in S such that w ≦ u
    if C(v) < C(u) then B_w = C(u) – C(v)
    else B_w = 0
    B(v,S) = B(v,S) + B_w
end

S = set of views selected for materialization

b ≦ a means b is a descendant of a (including itself) – b can be answered using only a (e.g., {S} ≦ {S,T}

C(v) = cost of view v, which we're approximating by its size

Benefit of Materializing a View



- The number associated with each node represents the number of rows in that view (in millions)
- Initial state has only the top most view materialized

Define the benefit (savings) of view v relative to S as **B(v,S)**.

B(v, S) = 0
For each w ≦ v
    u = view of least cost in S such that w ≦ u
    if C(v) < C(u) then $B_w = C(u) – C(v)$
    else $B_w = 0$
    $B(v,S) = B(v,S) + B_w$
end

Example

S = {{S, T, C}}, **v = {S, T}**

$B_{\{S, T\}}$ **= 5.2 M**

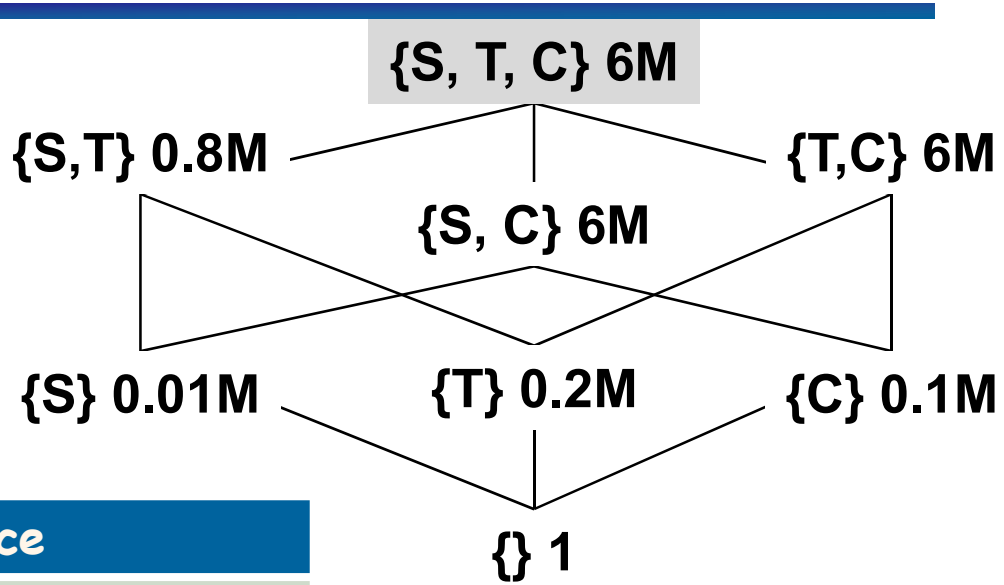
$B_{\{S\}}$ **= 5.2 M**

$B_{\{T\}}$ **= 5.2 M**

$B_{\{\}}$ **= 5.2 M**

B(v,S) = 5.2M *4

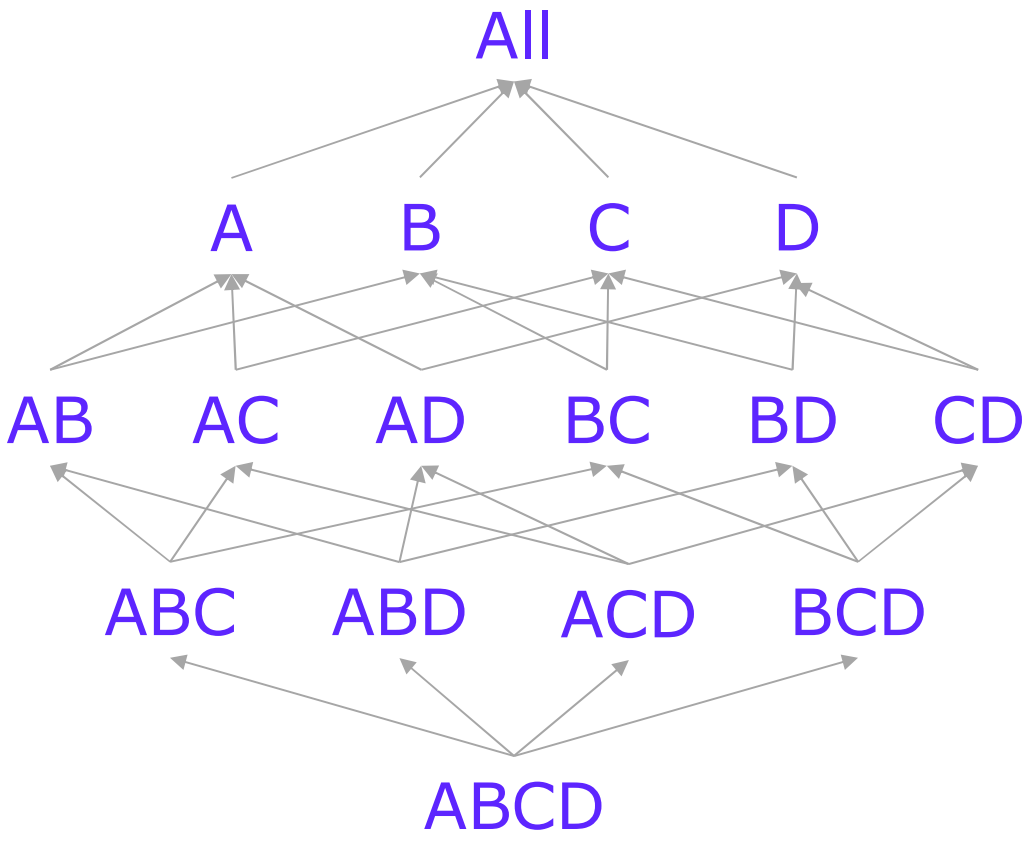
HRU Algorithm Example



View	1st choice	2nd choice
{S, T}	(6-0.8)M *4 = 20.8M	
{S, C}	(6-6) *4 = 0	(6-6) *2 = 0
{T, C}	(6-6) *4 = 0	(6-6) *2 = 0
{S}	(6-0.01) M*2 = 11.98M	(0.8-0.01)M*2 = 1.58M
{T}	(6-0.2) M*2 = 11.6M	(0.8-0.2)M*2 = 1.2M
{C}	(6-0.1) M*2 = 11.8M	(6-0.1)M + (0.8-0.1)M = 6.6M
{}	6M – 1	0.8M – 1

Assuming we start with {S, T, C}, for k=2, {S, T} and {C} will be materialized.

That's how to materialize *some* of the views. What if we want *all* views?



Given:

- A cube to materialize
- For each view in the cube:
  - Cost to materialize the view if the inputs are already sorted (A)
  - Cost to materialize the view if the inputs are NOT already sorted (S)
  - Need completely different algorithm!

Learning Goals Revisited

- Compare and contrast OLAP and OLTP processing (e.g., focus, clients, amount of data, abstraction levels, concurrency, and accuracy).
- Explain the ETL tasks (i.e., extract, transform, load) for data warehouses.
- Explain the differences between a star schema design and a snowflake design for a data warehouse, including potential tradeoffs in performance.
- Argue for the value of a data cube in terms of:
  - The type of data in the cube (numeric, categorical, temporal, counts, sums)
  - The goals of OLAP (e.g., summarization, abstractions), and
  - The operations that can be performed (drill-down, roll-up, slicing/dicing).
- Estimate the complexity of a data cube, in terms of the number of equivalent aggregation queries.