Administrative Notes
November 22 & 23, 2023

- Project Setup
  - Tutorial run as office hours through the 24th
- Upcoming project milestones:
  - Milestone 4: Implementation (Dec. 1). THIS IS WHEN YOUR CODE IS DUE.
  - Milestone 5: Demo (Dec 4-7)
    - Sign up next week!
  - Milestone 6: Individual & peer evaluations (Dec. 1)
- Final exam: December 14 @8:30am.

CPSC 304 – Administrative notes November 15 and 19, 2024

- Project:
  - Milestone 4: Project implementation – due November 29
    - You cannot change your code after this point!
  - Milestone 5: Group demo – week of December 2
  - Milestone 6: Individual Assessment – Due November 29
- Tutorials: basically project group work time/office hours
- Final exam: December 16 at 12pm! Osborne A

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 purpose of the star schema design, including potential tradeoffs.
- Argue for the value of a data cube in terms of:
  - 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.
- Apply the HRU algorithm to find the best *k* views to materialize.

What We Have Focused on So Far

- **OLTP (Online 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

Can We Do More?

- Increasingly, organizations are analyzing current and historical data to identify useful patterns and support long-term 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.

Let's imagine that you're trying to make some decisions at UBC like…

- Deciding what students to admit to what programs
- Understanding what students are at risk and need more support

These tasks require …

- Data that is large
- Data that comes from multiple sources
- Doesn’t need to be up to the minute accurate in order to be useful
- The ability to answer very complex queries very quickly.

OLTP is not the best choice to handle this scenario

What about just putting all the data together?

- One current direction (data lakes) says “hey, let’s just put all the data in the same spot, and people who ask queries can figure it out.”
- This is great for getting answers quickly if you don’t care about accuracy.
- But very hard to make super accurate decisions on.
- Among other things, the data might not even be in the same schema.
- Data Lakes are fast to build, but very slow to query.

What about Data Warehouses?

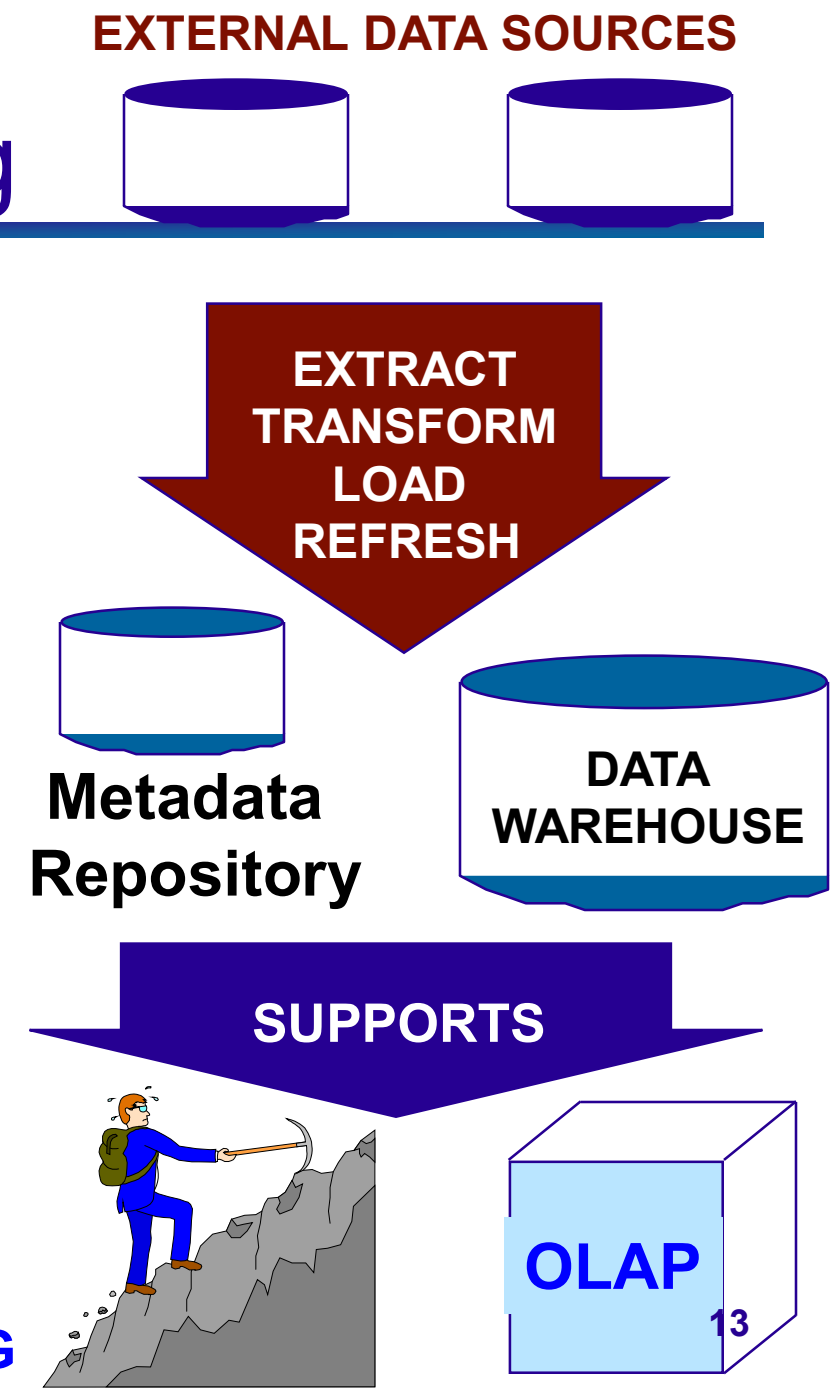
- Data warehouses import all the data into one application
- Data cleaning and integration techniques are applied to ensure consistency in naming conventions, schemas, etc.
- Everything is put into a common format
- Data warehouses are slow to build, but very fast to query.

Data Warehouse Data 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 UBC data across the university.
- Exercise: Provide some examples of mismatches that might occur in this example.

Data Warehousing

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



Data Warehouses support

OLAP (Online Analytical Processing)

- 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 experts rather than database experts.

Data Mining

- Exploratory search for interesting trends (patterns) and anomalies (e.g., outliers, deviations) using more sophisticated algorithms (as opposed to queries).

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

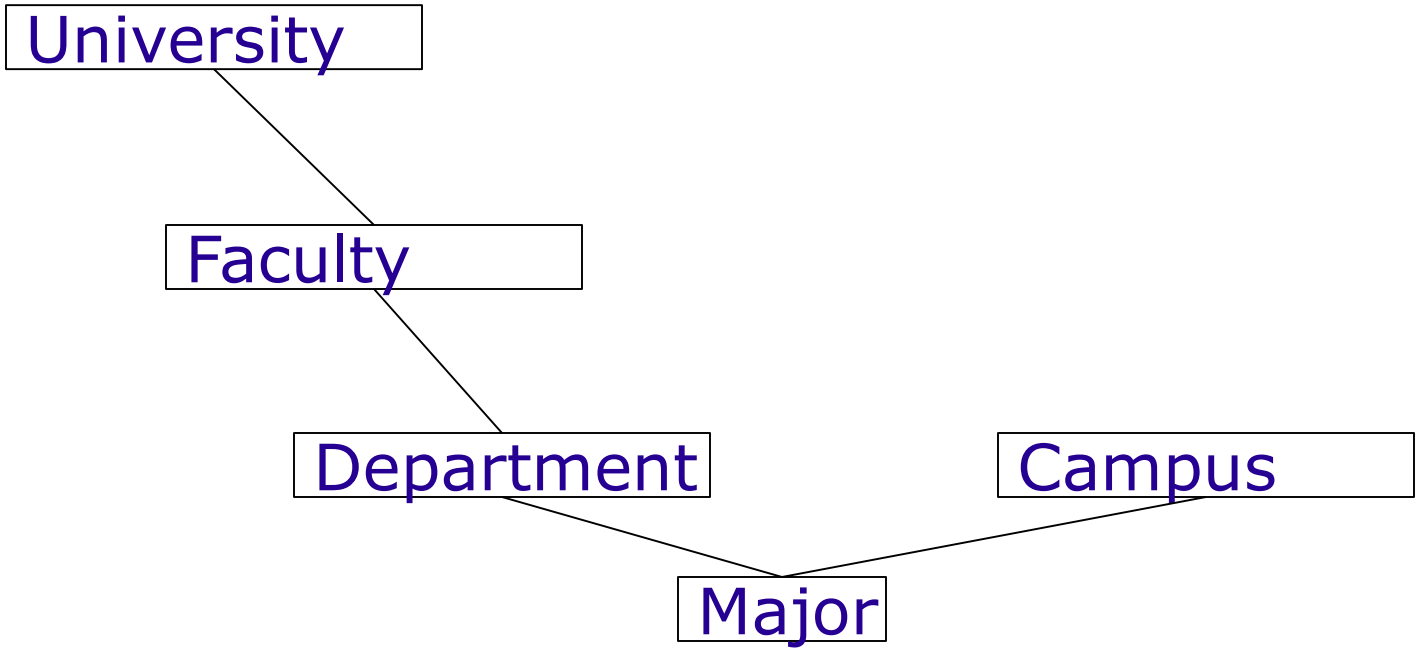
  A. OLTP B. OLAP

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

- There are interesting things that we can investigate if we try various aggregations of major, standing (year, enrolled, etc.), and age
- We're also going to ignore sname because it's not very interesting to do analysis on
- This is over simplistic: there are other things that you want to look at, too, like grades, but let's just use this as an example.

We can do interesting aggregations on major, standing, and age

- You can aggregate them in interesting ways, often in a hierarchy or two
- For example, if we look at majors…



So what we want is a design where…

- We can query about students as our central object
- We can query about majors, standings, and ages, as they relate to students
- 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.

Multidimensional Data Model

- The main relation, which relates dimensions to a measure via foreign keys, is called the ***fact table*** (e.g., students)
- Each dimension can have additional attributes and an associated ***dimension table*** (e.g., majors)
  - Attributes can be numeric, categorical, temporal, counts, sums
- Fact tables are usually *much* larger than dimensional tables.
- There can be multiple fact tables (e.g., students, courses, grades)

Multidimensional Data Model

- This model focuses on:
  - a set of numerical ***measures***: quantities that are important for business analysis, like numbers of majors, etc.
  - a set of ***dimensions***: entities on which the measures depend on, like majors.

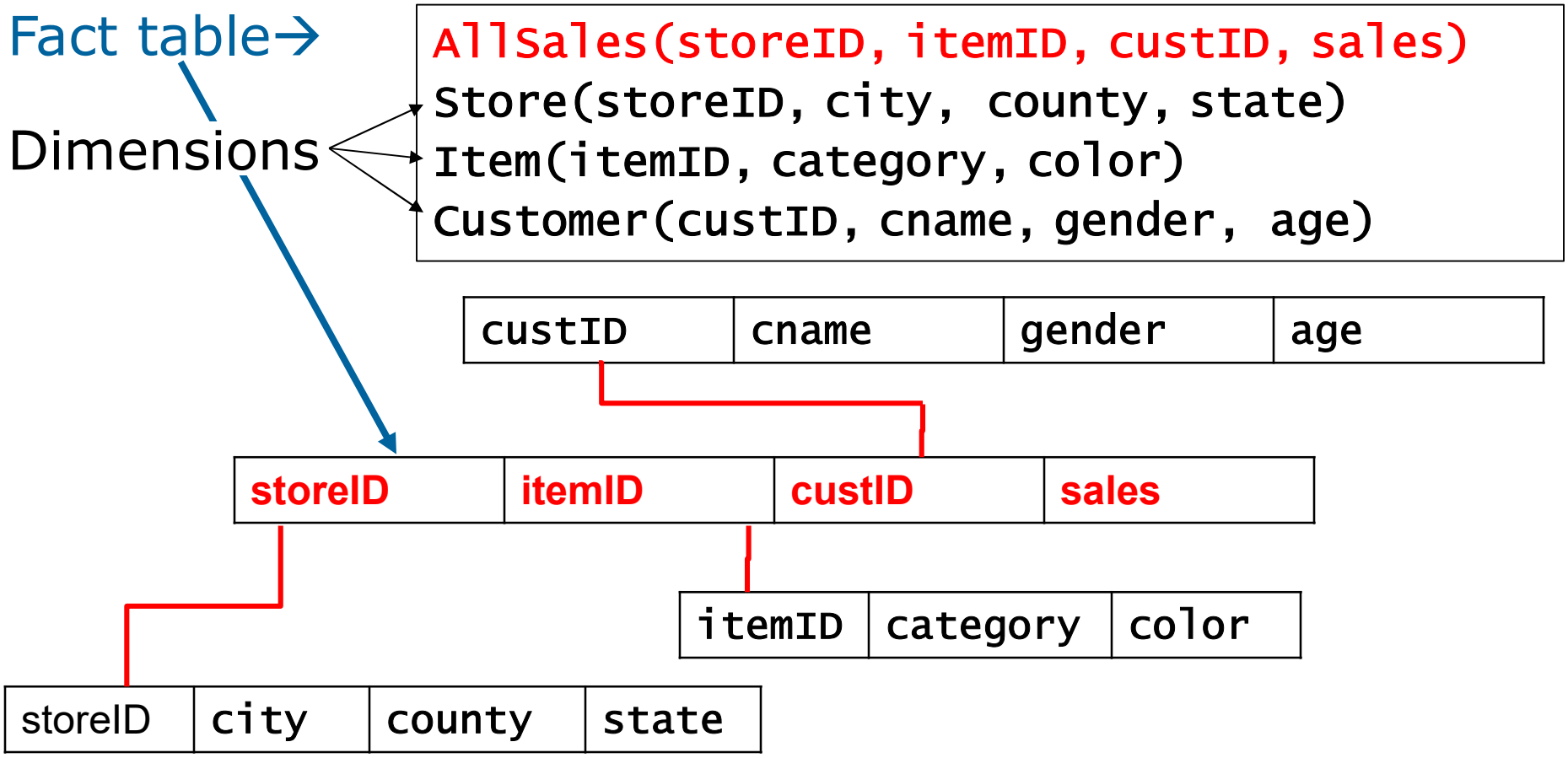
Design Issues

- 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

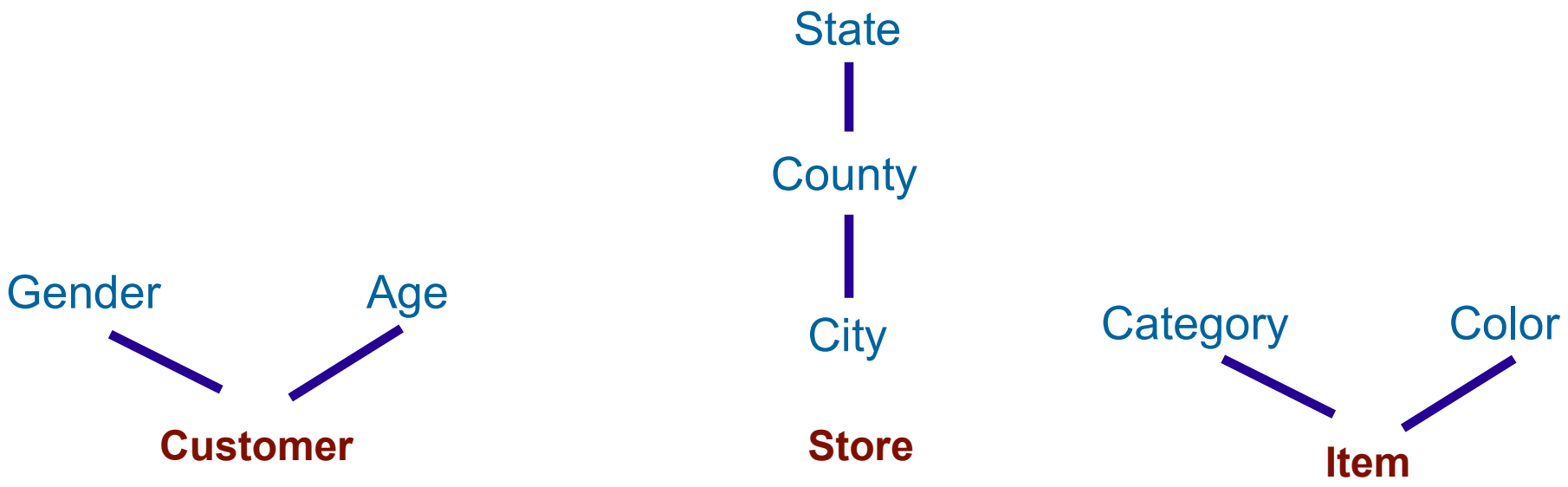
Star Schema – fact table references dimension tables

- Join → Filter → Group → Aggregate



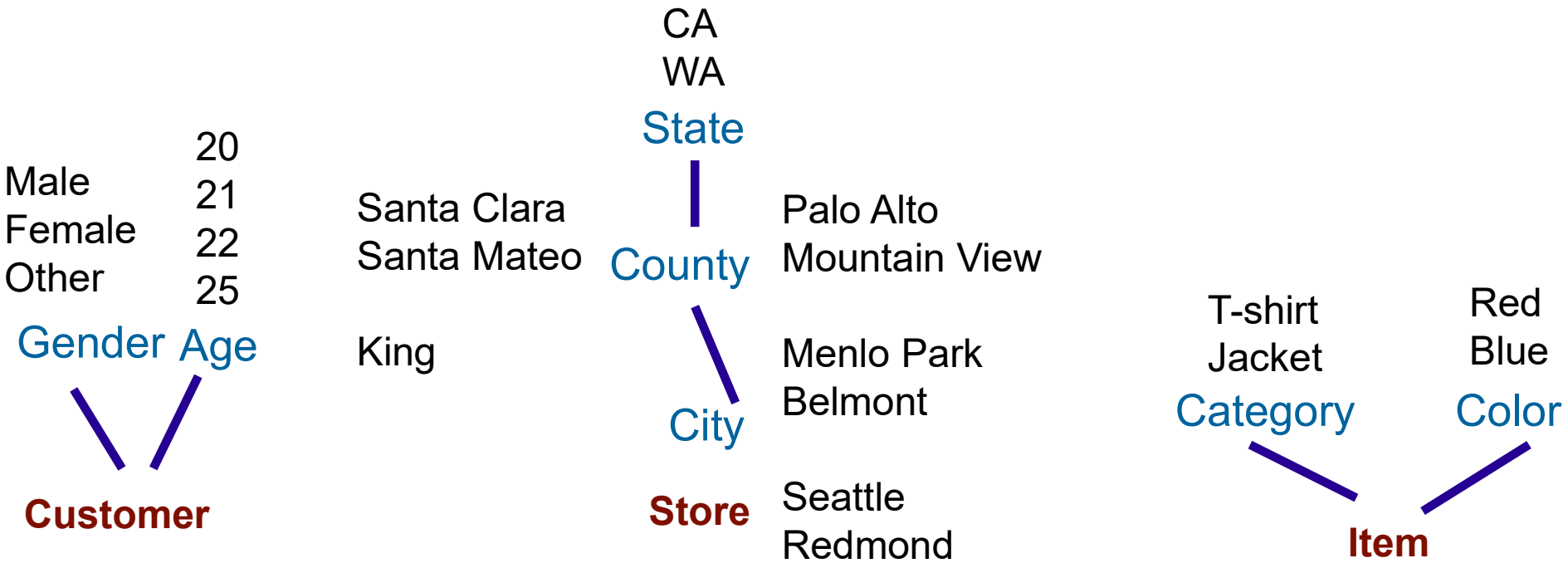
Dimension Hierarchies

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



Running Example (cont.)

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



Making this concrete

- Spend a couple of minutes thinking about a domain (maybe your project, or, when in doubt, students/grades usually work) that you might want to run OLAP queries on.
- Design a star schema for it
- Design some dimensions & hierarchies
- There are no rights or wrongs here; the goal is just to give you some practice/chance to realize questions about the topic.

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.

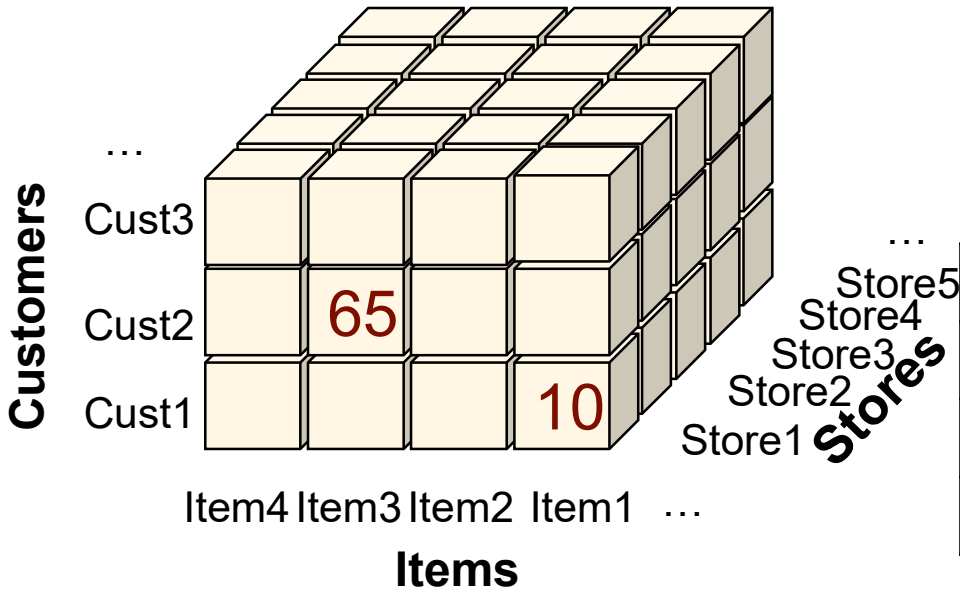
Find total sales by store, item, and customer.

Full Join:

```
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;
```

Desired outcome

```
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
…	…	…	…

Great! Now we have a schema!

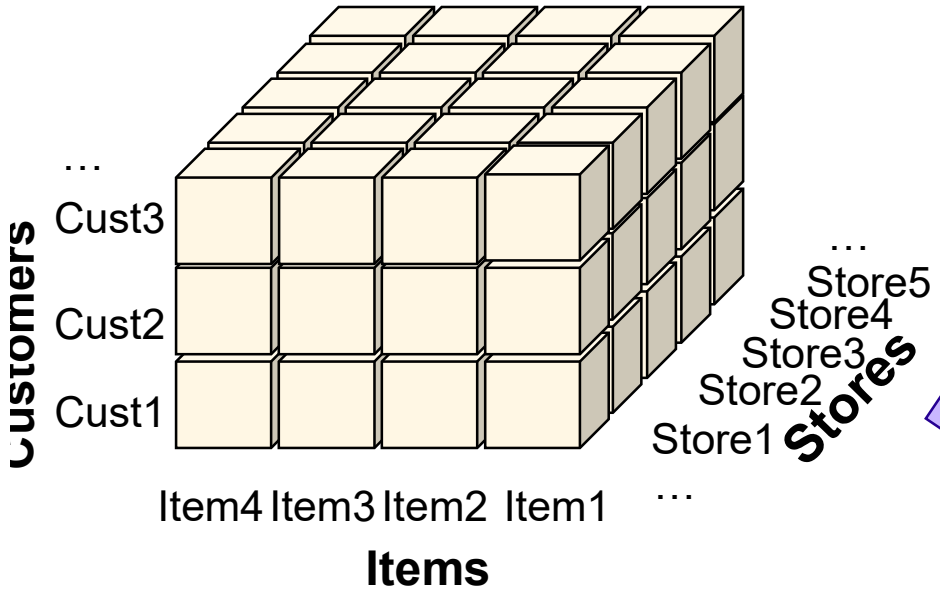
- What can we do with it?

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)

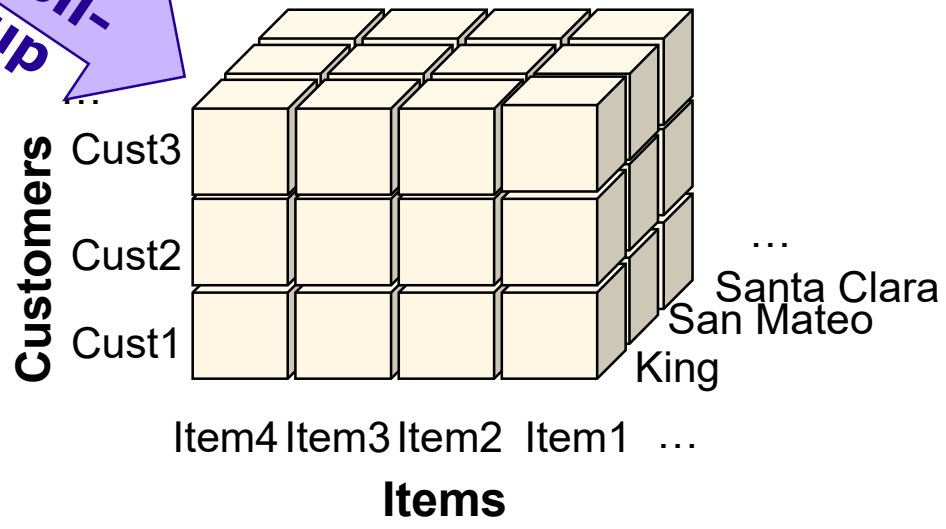
Use Roll-up on total sales by store, item, and customer to find total sales by item and customer for each county.



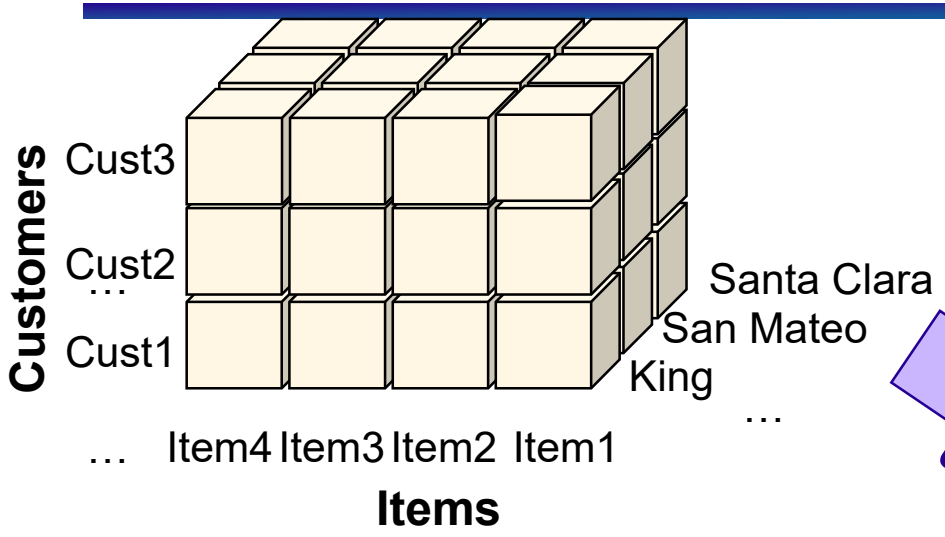

```
SELECT storeID, itemID, custID,
                SUM(sales)
FROM    AllSales F
GROUP BY storeID, itemID, custID;
```

Roll-up

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

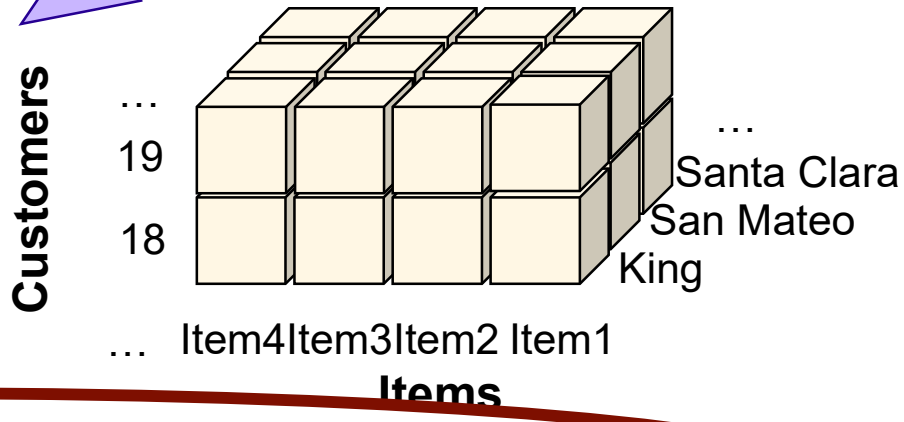


Clicker question: To use Roll-up on total sales by item, customer, and county (top row) to find total sales by item, age and county (bottom row)…




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

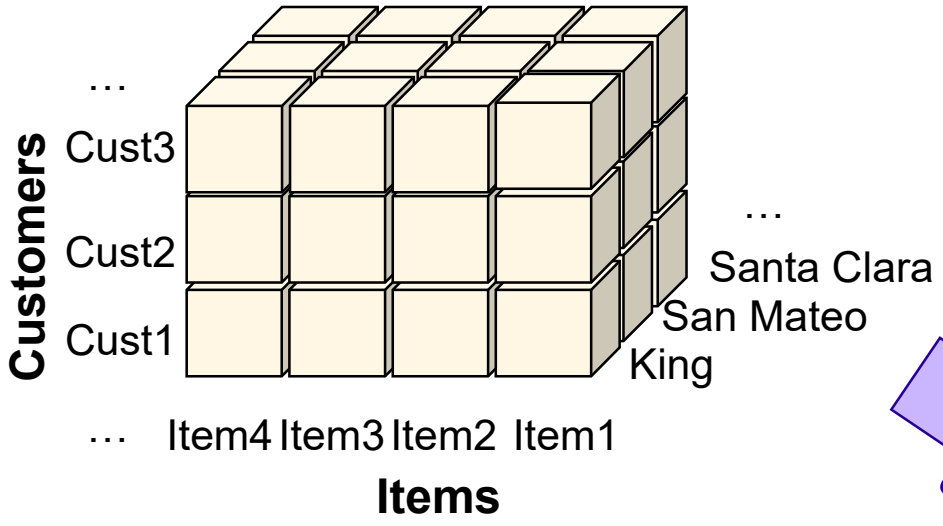
Roll-up



A. Add “age” to the group by and select clauses
B. Add “age” to the select and where clauses
C. Replace custID with Age in the group by and select clauses
D. None of the above

Roll-up Example 2 (Hierarchy) Clicker Question answer

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



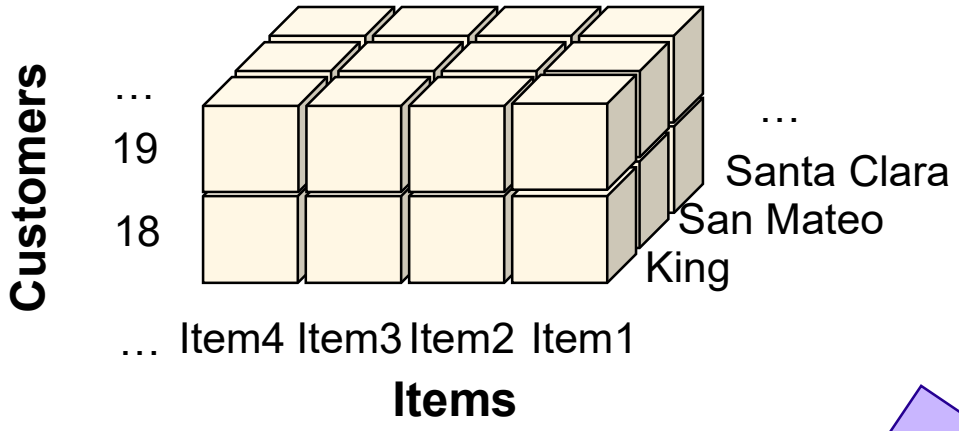

```
SELECT county, itemID, custID,
                SUM(sales)
FROM    AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY county, itemID, custID;
```

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)

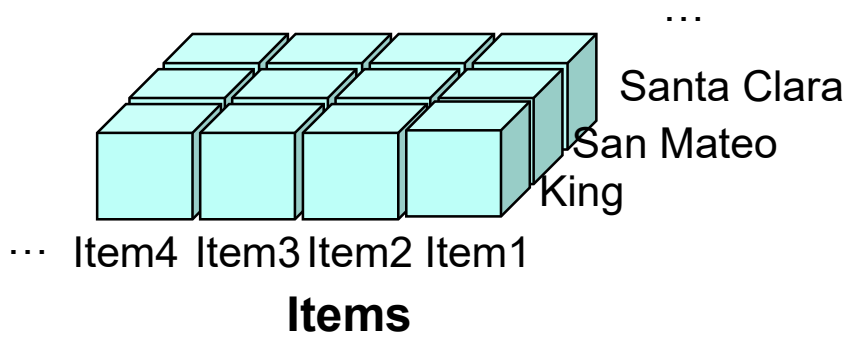
Use Roll-up on total sales by item, age and county to find total sales by item for each county.




```
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;
```

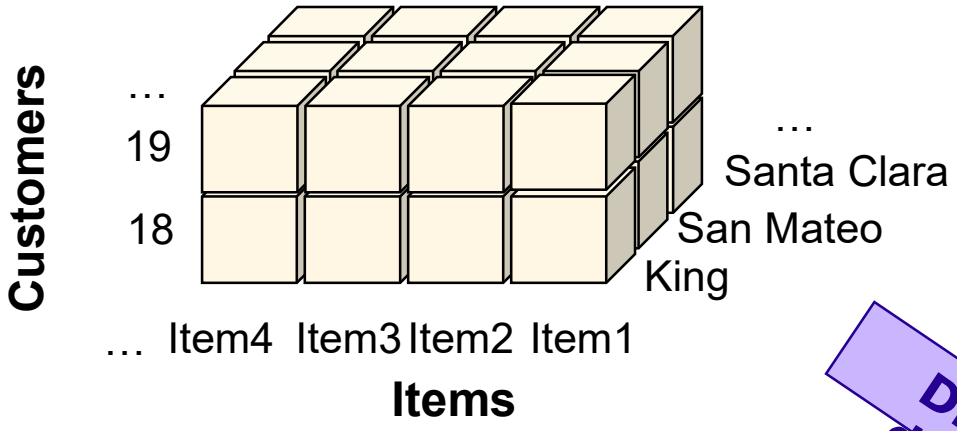


OLAP Queries – Drill-down

- 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)

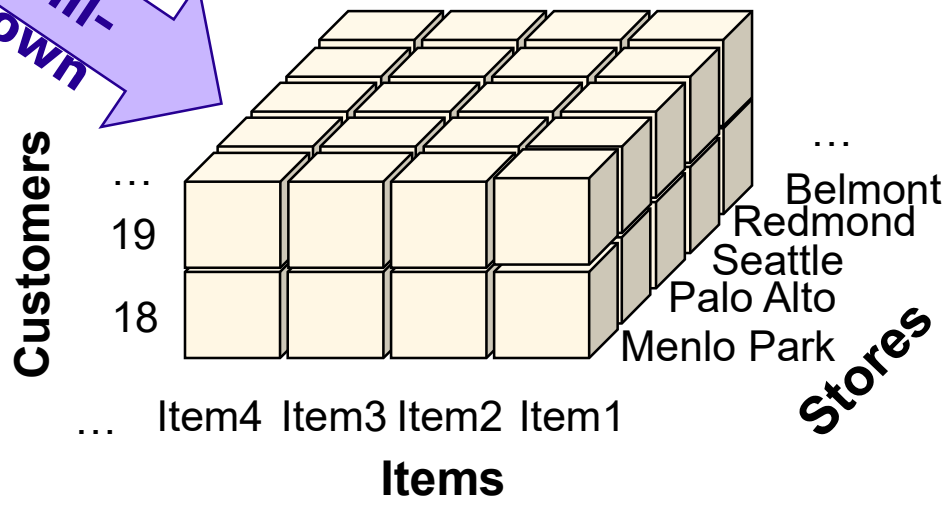
Use Drill-down on total sales by item and age for each county to find total sales by item and age for each city.




```
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;
```

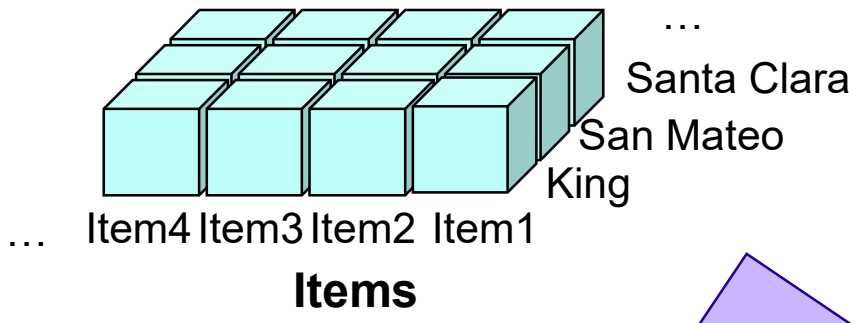
Drill-down

```
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)

Clicker question: to 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;
```

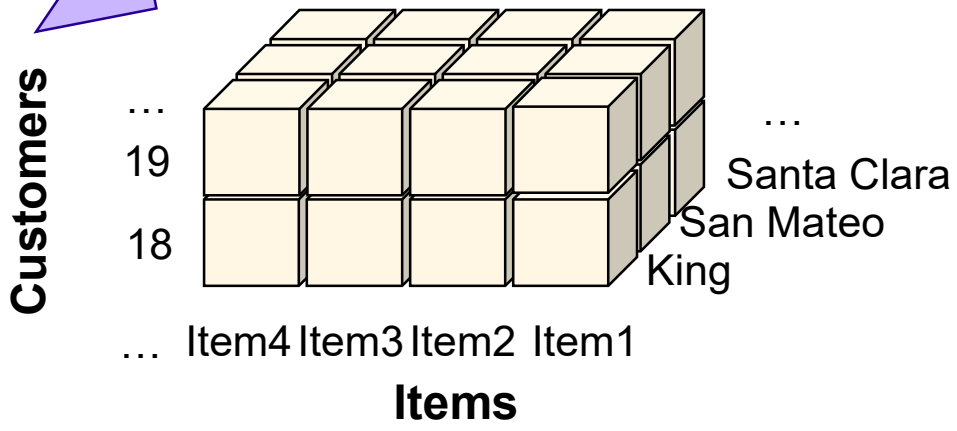
Drill-down

A. Add "age" to the group by and select clauses & Customer to the from clause

B. Add "age" to the select and where clauses & Customer to the from clause

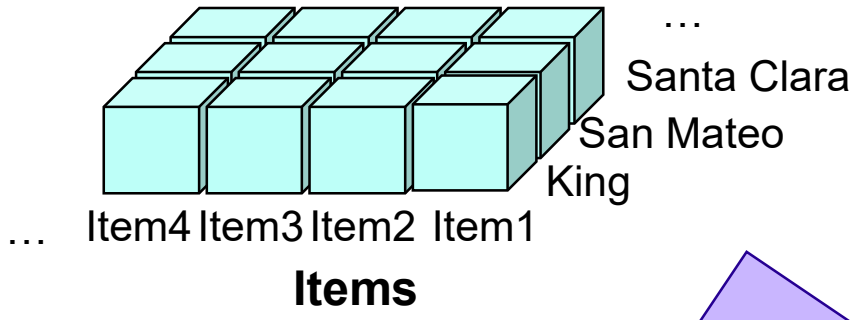
C. Replace itemID with Age in the group by and select clauses

D. None of the above



Drill-down Example 2 (Dimension) (Clicker question answer)

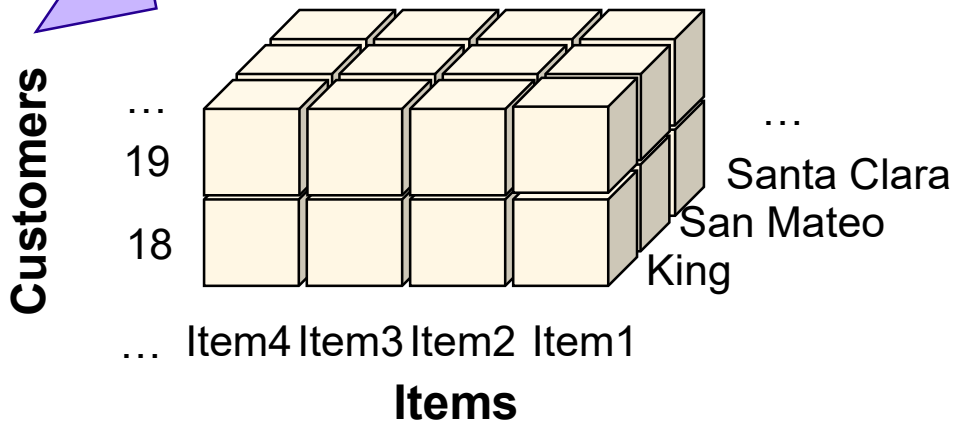
Use Drill-down on total sales by item and county to find total sales by item and age for each county.




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

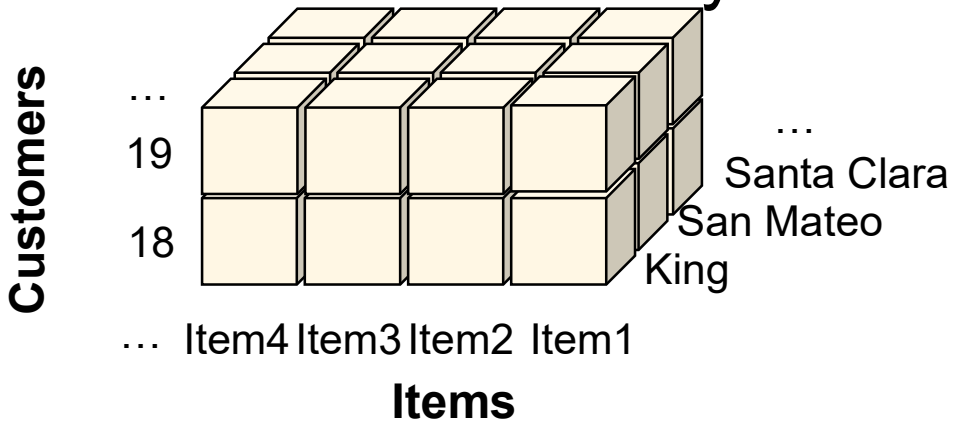
Drill-down

```
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

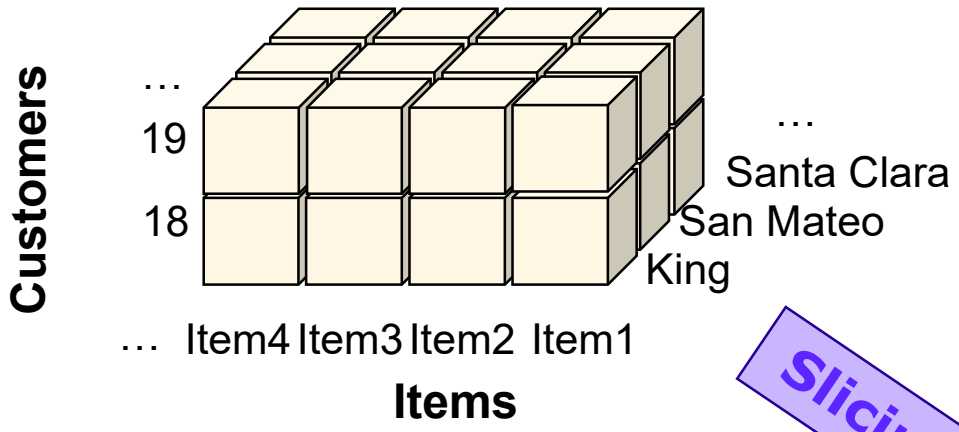
- The slice operation produces a slice of the cube by picking a range or a specific value for one of the dimensions.
- To start our example, let's specify:
  - Total sales by item and age for each county




```
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

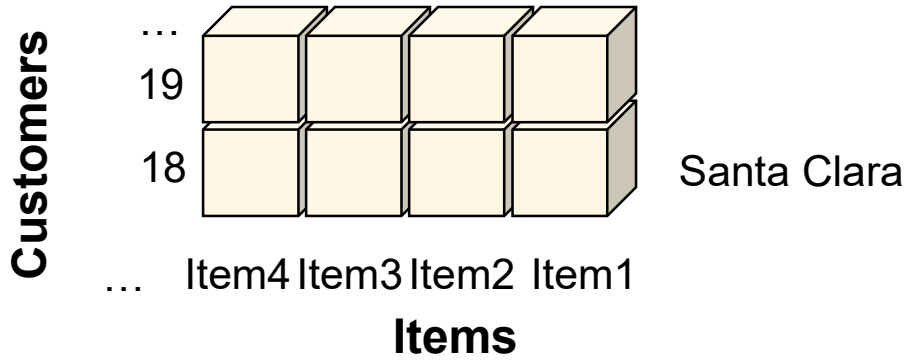
Use Slicing on total sales by item and age for each county to find total sales by item and age for Santa Clara.




```
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;
```

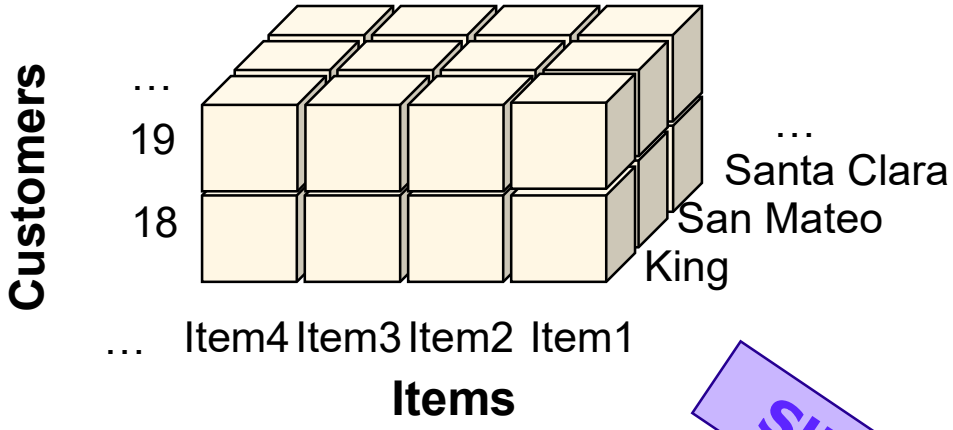
Slicing

```
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

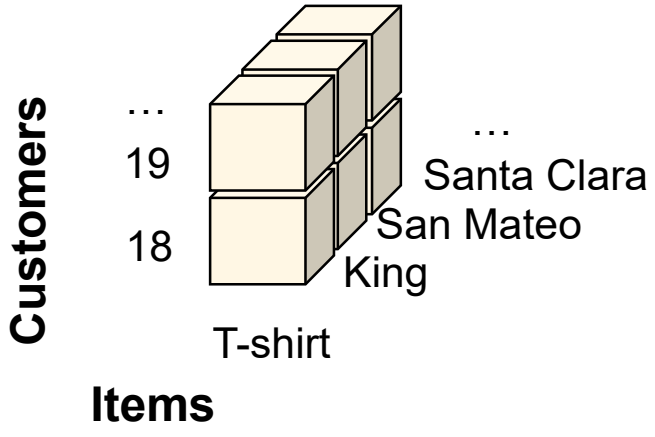
Use Slicing on total sales by item and age for each county to find total sales by age and county for T-shirts.




```
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;
```

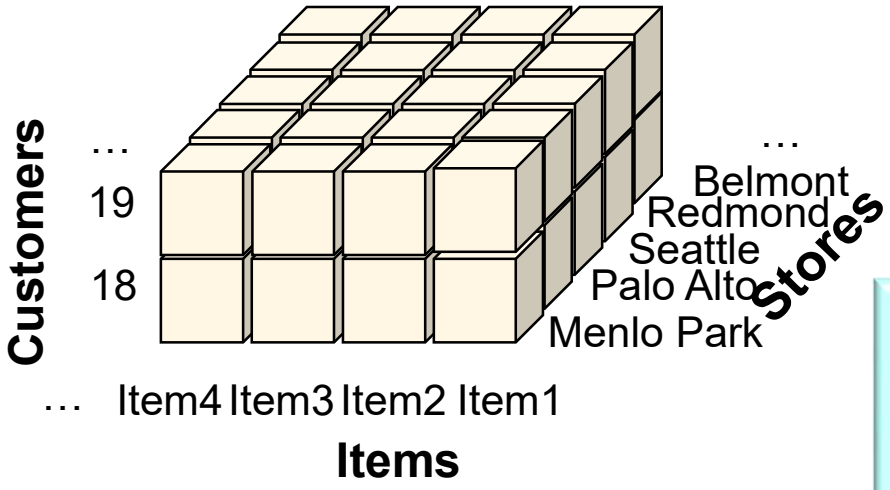
Slicing

```
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

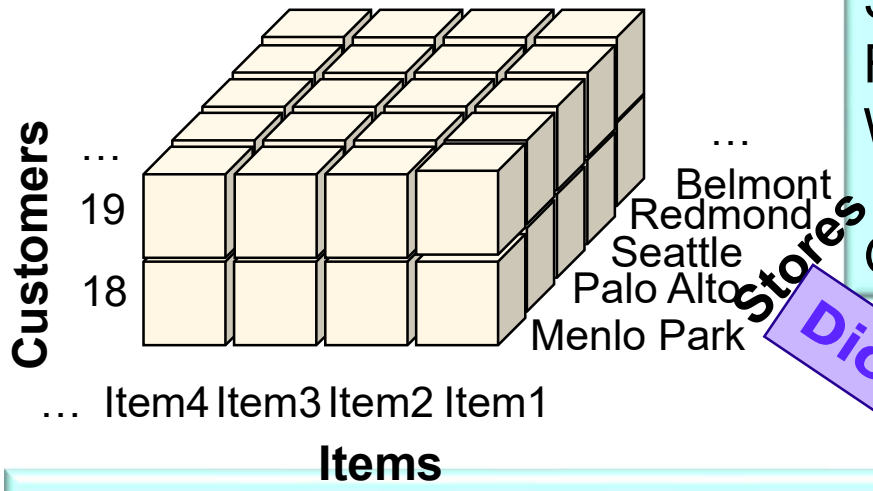
- The dice operation produces a sub-cube by picking ranges or specific values for multiple dimensions.
- To start our example, let's specify:
  - Total sales by age, item, and city




```
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

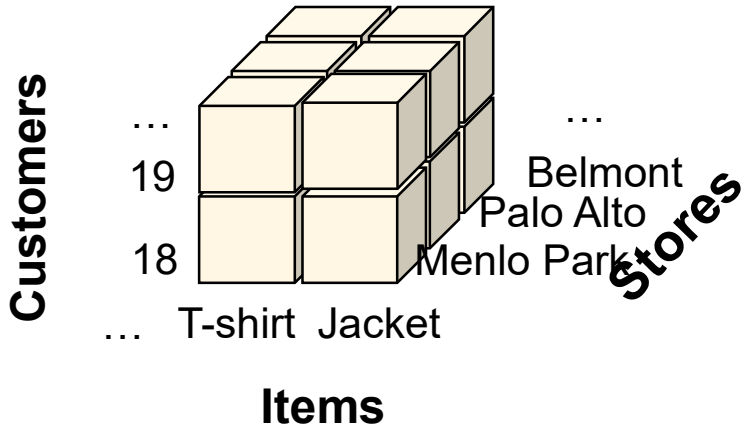
Use Dicing on total sales by age, item, and city to find total sales by age, category, and city for red items in the state of California.




```
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;
```



Clicker Question

- Consider a fact table Sales(saleID, itemID, color, size, qty, unitPrice), and the following three queries:
- Q1: SELECT itemID, color, size, Sum(qty*unitPrice) FROM Sales GROUP BY itemID, color, size
- Q2: SELECT itemID, size, Sum(qty*unitPrice) FROM Sales GROUP BY itemID, size
- Q3: SELECT itemID, size, Sum(qty*unitPrice) FROM Sales WHERE size < 10 GROUP BY itemID, size
- Which of the following statements is correct?

  A: Going from Q2 to Q3 is an example of roll-up.

  B: Going from Q2 to Q1 is an example of drill-down.

  C: Going from Q3 to Q2 is an example of roll-up.

  D: Going from Q1 to Q2 is an example of drill-down.

Clicker Question

- Consider a fact table Sales(saleID, itemID, color, size, qty, unitPrice), and the following three queries:
- Q1: SELECT itemID, color, size, Sum(qty*unitPrice) FROM Sales GROUP BY itemID, color, size
- Q2: SELECT itemID, size, Sum(qty*unitPrice) FROM Sales GROUP BY itemID, size
- Q3: SELECT itemID, size, Sum(qty*unitPrice) FROM Sales WHERE size < 10 GROUP BY itemID, size
- Which of the following statements is correct?
  - A: Going from Q2 to Q3 is an example of roll-up.
  - B: Going from Q2 to Q1 is an example of drill-down.
  - C: Going from Q3 to Q2 is an example of roll-up.
  - D: Going from Q1 to Q2 is an example of drill-down.

Slicing

Correct

Filtering (new term)

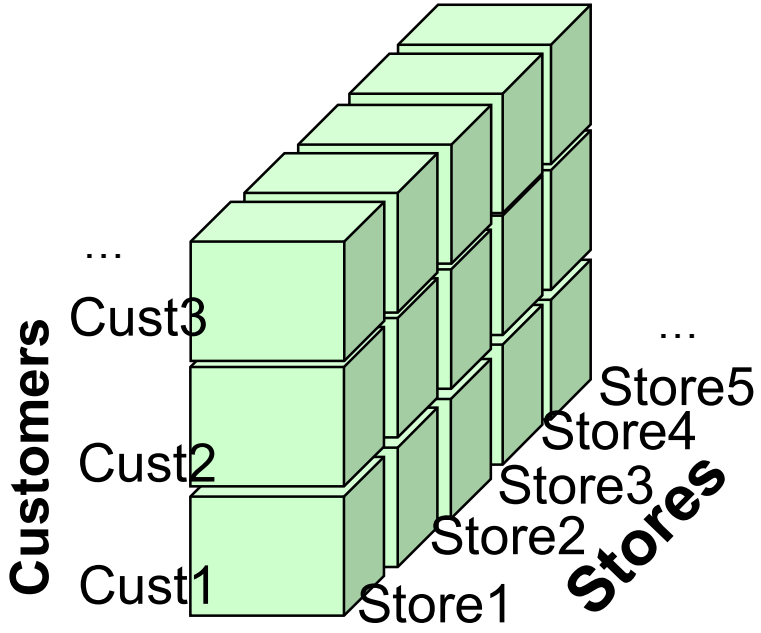
Roll-up

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.

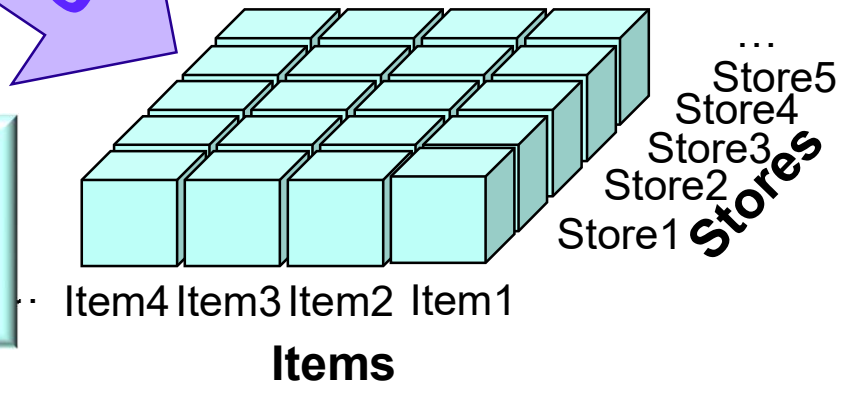
Pivoting Example 1

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;
```

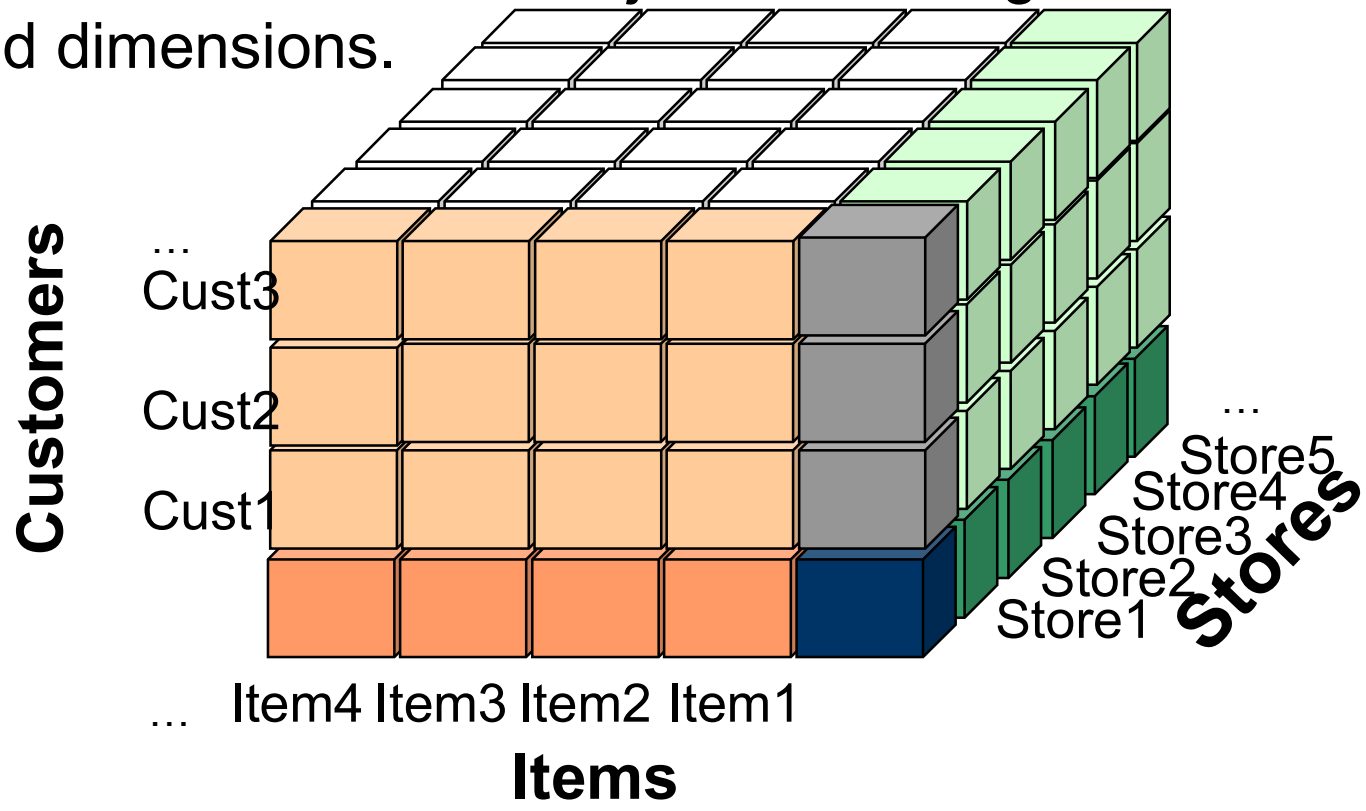
Pivoting




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

Data Cube

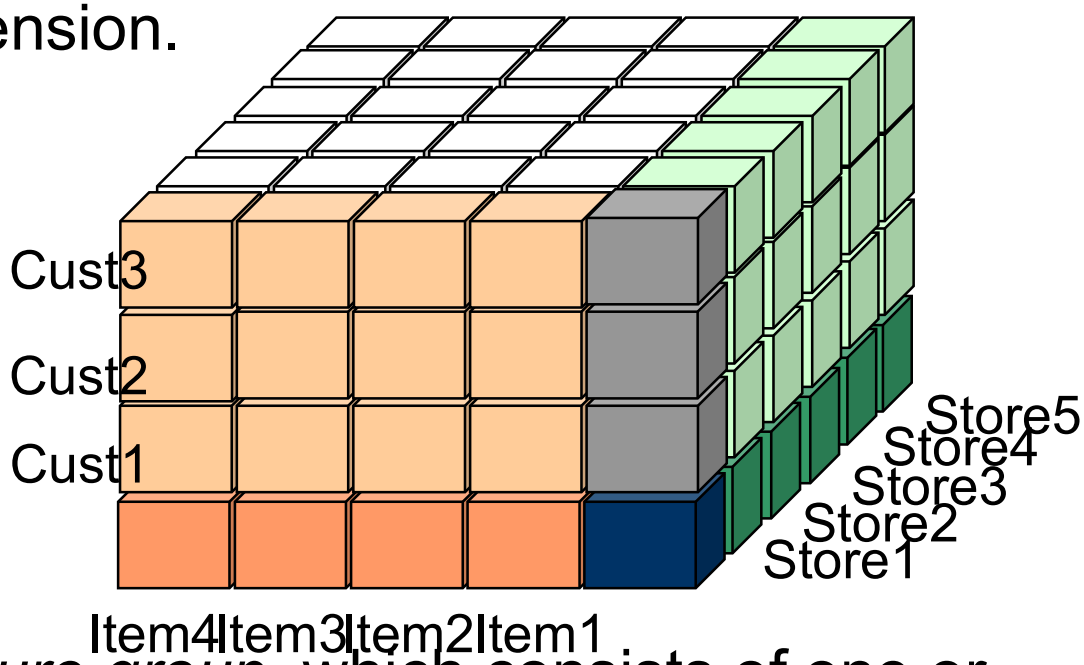
- A *data cube* is a *k*-dimensional object containing both fact data and dimensions.



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

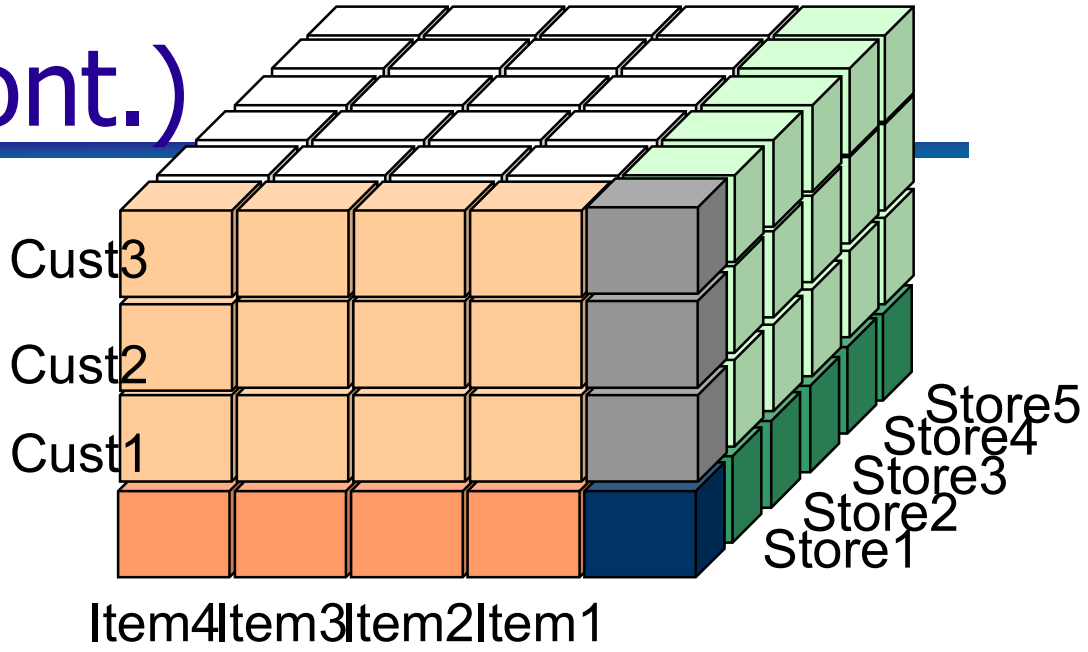
Data Cube (cont.)

- 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

Data Cube (cont.)



- White: per customer, per item, per store (all separate)
- Dark blue: all customers, all items, all stores (all aggregated)
- Grey: per customer, all items, all stores
- Light orange: per customer, per items, all stores
- Light green: per customer, all items, per store
- Dark orange: all customers, per item, all stores
- Dark green: all customers, all items, per store

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 $\prod_{i=1}^{n}(Ci + 1)$

Cube size estimation exercise

- Consider a car example with dimensions model, year, colour
- Consider if there are 10 models, 20 years, and 5 colours
- What is the size of the cube? Note: all math is correct.

A. 10 x 20 x 5 = 1000 tuples

B. 11 x 21 x 6 = 1386 tuples

C. Neither of the above

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*

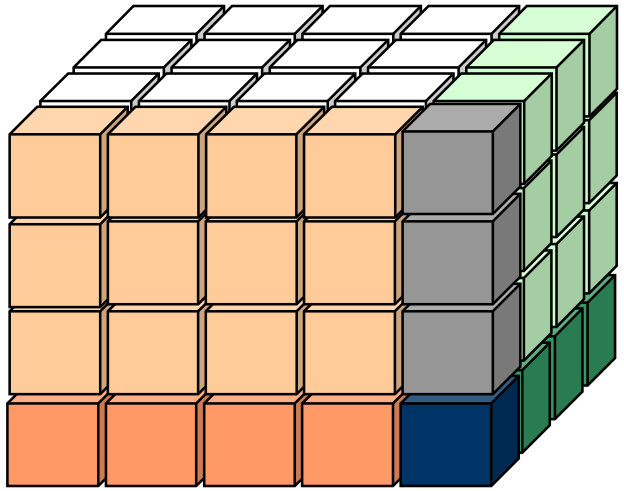
Estimating the size of a sparse cube

- 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

Clicker Question

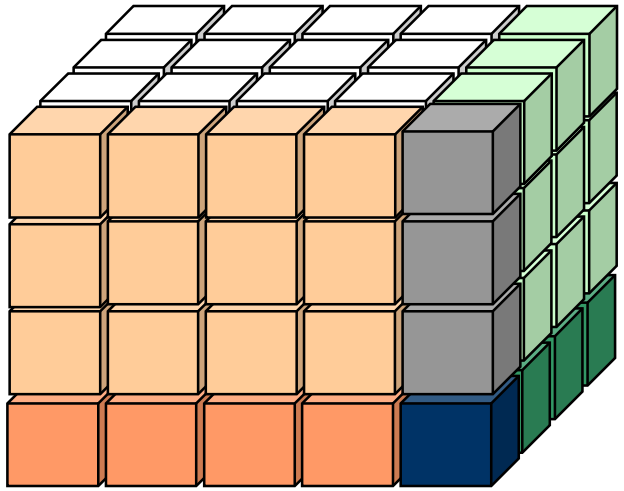
If we have 2 stores, 5 items, and 10 customers, how many potential "entries" are there in the data cube? (The cube diagram is just an arbitrary example; assume a dense cube.)

- A: 17
- B: 100
- C: 117
- D: 198
- E: none of the above



Clicker Question

If we have 2 stores, 5 items, and 10 customers, how many potential "entries" are there in the data cube? (The cube diagram is just an arbitrary example; assume a dense cube.)



- A: 17
- B: 100
- C: 117
- D: 198
- E: none of the above

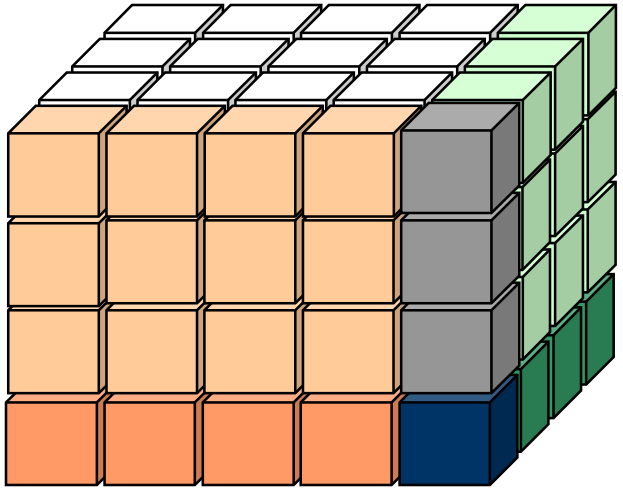
Thinking this through requires thinking through the aggregations

(2+1) * (5+1) * (10+1) = 3 * 6 * 11

Clicker Question

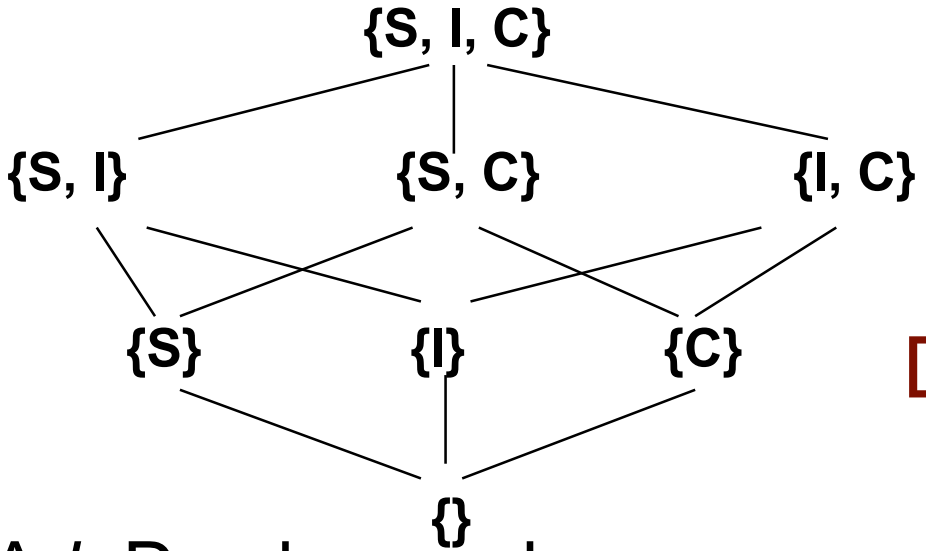
- How many standard (GROUP BY) SQL queries are required for computing all of the cells of a 3 dimensional cube?

- A: 2
- B: 4
- C: 6
- D: 8
- E: 10

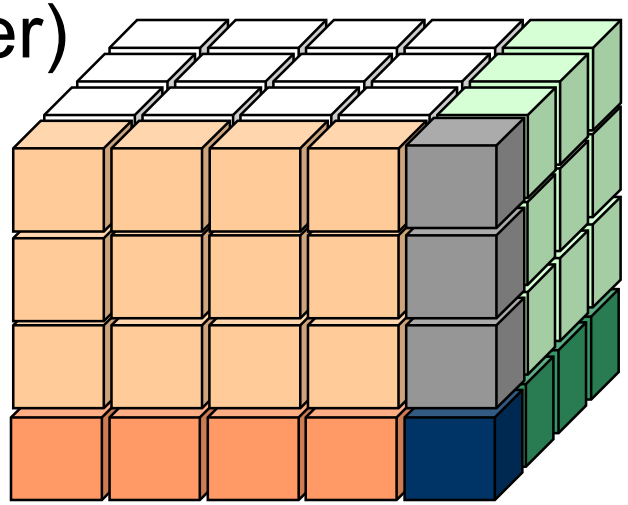


Clicker Question

- How many standard SQL queries are required for computing all of the cells of a 3 dimensional cube? (Ex: Store, Item, Customer)



D: 8

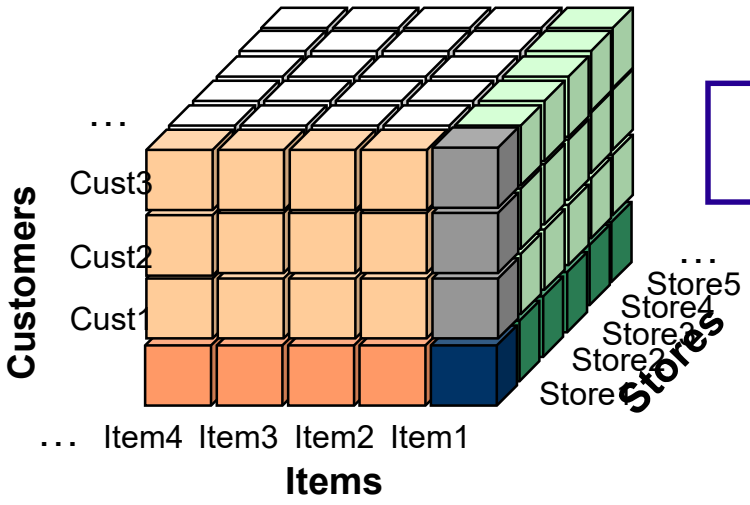


A *k*-D cube can be represented as a series of (*k*-1)-D cubes.

  - The cube is *exponential* in the number of cells.

Seva & Rachel stopped here

Representing a Cube in a Two-Dimensional Table



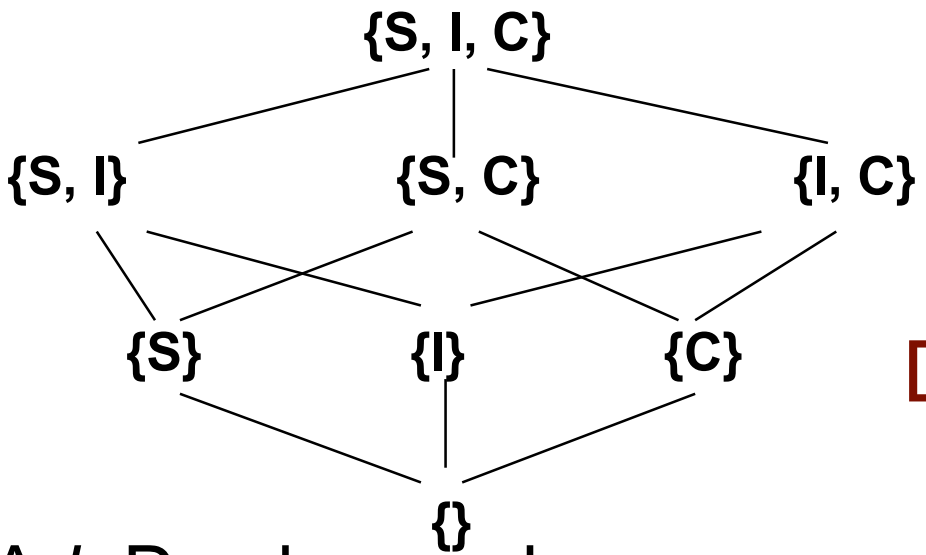
storeID	itemID	custID	Sum
store1	item1	cust1	10
store1	item1	Null	70
store1	Null	cust1	145
store1	Null	Null	325
Null	item1	cust1	10
Null	item1	Null	135
Null	Null	cust1	670
Null	Null	Null	3350

. . . . . .

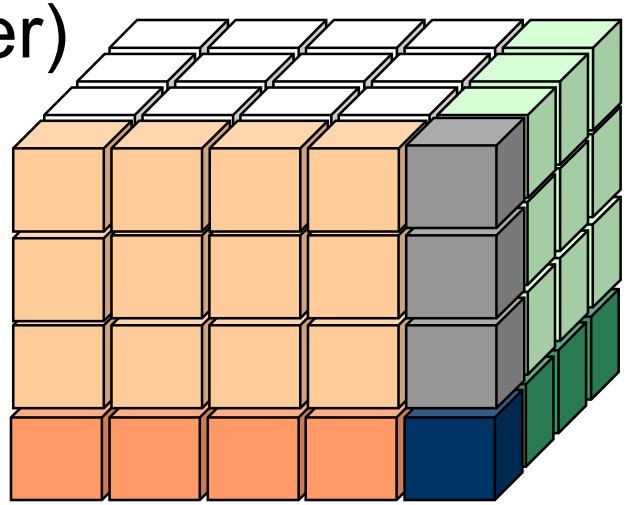
Add to the original cube: faces, edges, and corners … which are represented in the 2-D table using NULLs.

Clicker Question

- How many standard SQL queries are required for computing all of the cells of a 3 dimensional cube? (Ex: Store, Item, Customer)



D: 8



A *k*-D cube can be represented as a series of (*k*-1)-D cubes.

- The cube is *exponential* in the number of cells.

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 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

.....

WITH ROLLUP

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

Can be used in dimensions that are organized in a hierarchy:

```
SELECT state, county, city, sum(sales)
FROM   AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY state, county, city WITH ROLLUP
```

State	County	city	Sum
CA	Santa Clara	Palo Alto	325
CA	Santa Clara	Mountain view	805
CA	Santa Clara	Null	1130
CA	Null	Null	1980
Null	Null	Null	3350

.....

WITH CUBE Example Implemented WITH ROLLUP

Implement the WITH CUBE operator using the WITH ROLLUP operator

```
SELECT storeID, itemID, custID, sum(sales)
FROM AllSales
GROUP BY storeID, itemID, custID with rollup
UNION

SELECT storeID, itemID, custID, sum(sales)
FROM AllSales
GROUP BY itemID, custID, storeID with rollup
UNION

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

Clicker Question

- Consider a fact table Facts(D1,D2,D3,x), and the following queries:

Q1: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3
Q2: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with cube

- Suppose attributes D1, D2, and D3 have n1, n2, and n3 different values respectively, and assume that each possible combination of values appears at least once in table Facts. Pick the one tuple (a,b,c,d,e) in the list below such that when n1=a, n2=b, and n3=c, then the result sizes of queries Q1 and Q2 are d and e, respectively.
- A: (2, 2, 2, 8, 64)
- B: (5, 4, 3, 60, 64)
- C: (5, 10, 10, 500, 726)
- D: (4, 7, 3, 84, 160)

Hint: It may be helpful to first write formulas describing how d, and e depend on a, b, and c.

Clicker Question

- Consider a fact table Facts(D1,D2,D3,x), and the following queries:

Q1: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3
Q2: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with cube

- Suppose attributes D1, D2, and D3 have n1, n2, and n3 different values respectively, and assume that each possible combination of values appears at least once in table Facts. Pick the one tuple (a,b,c,d,e) in the list below such that when n1=a, n2=b, and n3=c, then the result sizes of queries Q1 and Q2, are d and e, respectively.
- A: (2, 2, 2, 8, 64)
- B: (5, 4, 3, 60, 64)
- C: (5, 10, 10, 500, 726)
- D: (4, 7, 3, 84, 160)

d = a*b*c
e = (a+1)*(b+1)*(c+1)

Clicker Question

- Consider a fact table Facts(D1,D2,D3,x), and the following queries:

Q1: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3
Q2: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with cube
Q3: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with rollup

- Suppose attributes D1, D2, and D3 have n1, n2, and n3 different values respectively, and assume that each possible combination of values appears at least once in table Facts. Pick the one tuple (a,b,c,d,e,f) in the list below such that when n1=a, n2=b, and n3=c, then the result sizes of queries Q1, Q2, and Q3 are d, e, and f respectively.
- A: (2, 2, 2, 8, 64, 15)
- B: (5, 4, 3, 60, 64, 80)
- C: (5, 10, 10, 500, 726, 556)
- D: (4, 7, 3, 84, 160, 84)

Hint: It may be helpful to first write formulas describing how d, e, and f depend on a, b, and c.

Clicker Question

- Consider a fact table Facts(D1,D2,D3,x), and the following queries:

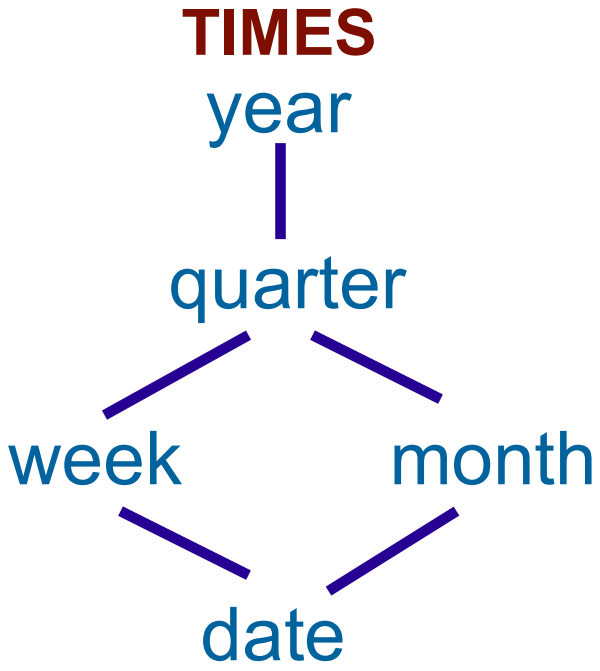
```
Q1: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3
Q2: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with cube
Q3: Select D1,D2,D3,Sum(x) From Facts Group By D1,D2,D3 with rollup
```

- Suppose attributes D1, D2, and D3 have n1, n2, and n3 different values respectively, and assume that each possible combination of values appears at least once in table Facts. Pick the one tuple (a,b,c,d,e,f) in the list below such that when n1=a, n2=b, and n3=c, then the result sizes of queries Q1, Q2, and Q3 are d, e, and f respectively.
- A: (2, 2, 2, 8, 64, 15)
- B: (5, 4, 3, 60, 64, 80)
- C: (5, 10, 10, 500, 726, 556)
- D: (4, 7, 3, 84, 160, 84)

```
d = a*b*c
e = a*b*c+a*b+a*c+b*c+a+b+c+1
e = (a+1)*(b+1)*(c+1)
f =  a*b*c + a*b + a + 1
```

“Date” or “Time” Dimension

- Date or Time is a special kind of dimension.
- It has some special and useful OLAP functions.
  - e.g., durations or time spans, fiscal years, calendar years, and holidays
  - Business intelligence reports often deal with time-related queries such as comparing the profits from this quarter to the previous quarter … or to the same quarter in the previous year.



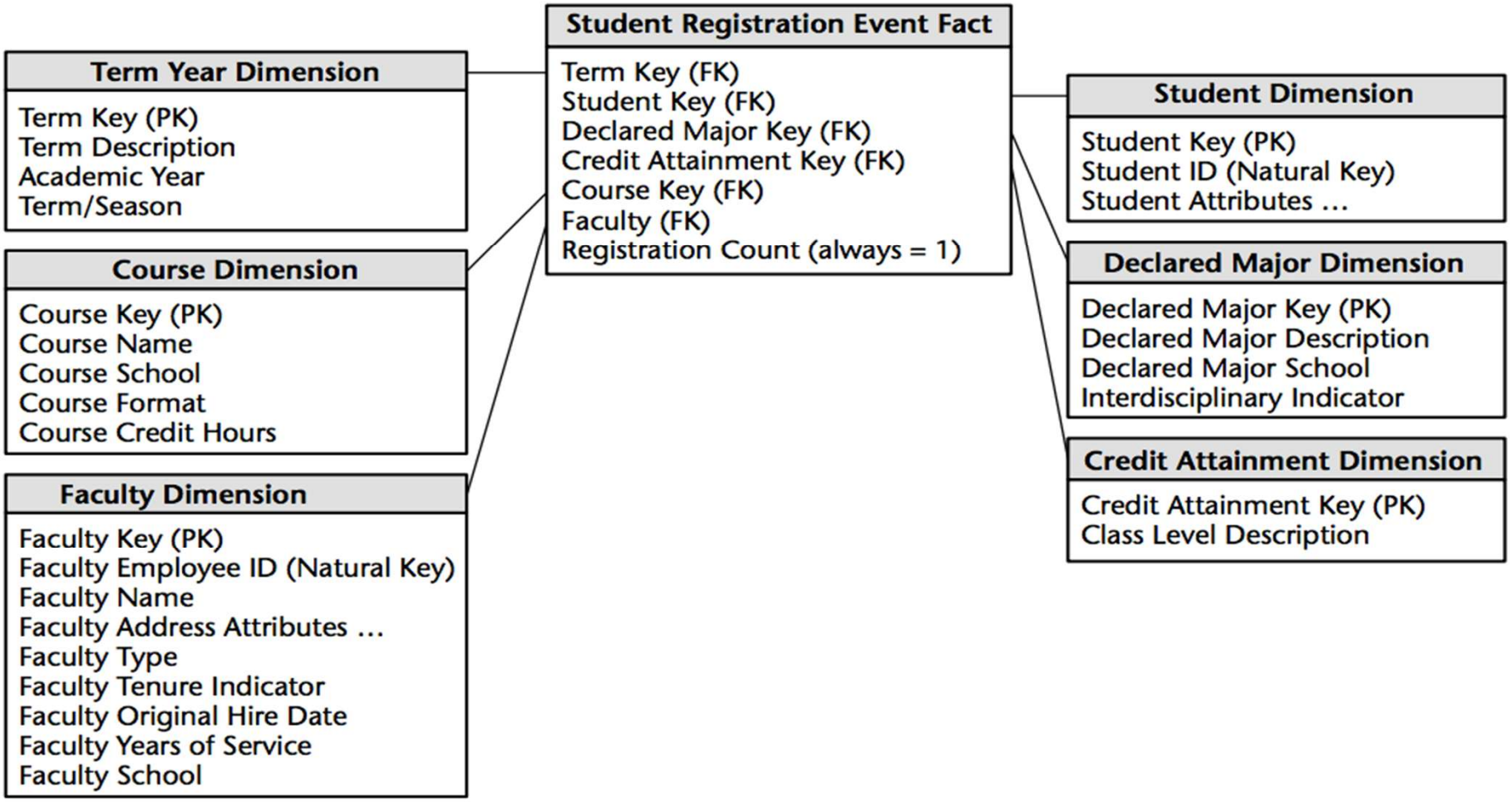
Measures in Fact Tables

- **Additive** facts are measurements in a fact table that can be added across all the dimensions. e.g., sales
- **Semi-additive** facts are numeric facts that can be added along some dimensions in a fact table but not others.
  - balance amounts are common semi-additive facts because they are additive across all dimensions except time.
- **Non-additive** facts cannot logically be added between rows.
  - Ratios and percentages
  - A good approach for non-additive facts is to store the fully additive components and later compute the final non-additive fact.

Factless Fact Tables

- **Factless fact table:** A fact table that has no facts but captures certain many-to-many relationships between the dimension keys. It's most often used to represent events or to provide coverage information that does not appear in other fact tables.

Factless Fact Table Example



Source: *The Data Warehouse Toolkit* textbook

Multidimensional Data Model

- Multidimensional data can be stored in one of 3 ways (modes):
  - **ROLAP** (relational online analytical processing)
    - We access the data in a relational database and generate SQL queries to calculate information at the appropriate level when an end user requests it.
  - **MOLAP** (multidimensional online analytical processing)
    - Requires the pre-computation and storage of information in the cube — the operation known as processing. Most MOLAP solutions store such data in an optimized multidimensional array storage, rather than in a relational database.

MOLAP vs. ROLAP

	MOLAP	ROLAP
Data Compression	Can require up to 50% less disk space. A special technique is used for storing sparse cubes.	Requires more disk space
Query Performance	Fast query performance due to optimized storage, multidimensional indexing and caching	Not suitable when the model is heavy on calculations; this doesn't translate well into SQL.
Data latency	Data loading can be quite lengthy for large data volumes. This is usually remedied by doing only incremental processing.	As data always gets fetched from a relational source, data latency is small or none.
handling non-aggregatable facts	Tends to suffer from slow performance when dealing with textual descriptions	Better at handling textual descriptions

Which Storage Mode is Recommended?

- Almost always, choose MOLAP.
- Choose ROLAP if one or more of these are true:
  - There is a very large number of members in a dimension—typically hundreds of millions of members.
  - The dimension data is frequently changing.
  - You need real-time access to current data (as opposed to historical data).
  - You don't want to duplicate data.
    - Reference: [Harinath, et *al.*, 2009]

- **HOLAP** (**hybrid online analytical processing**) is a combination of ROLAP and MOLAP, which allows storing part of the data in a MOLAP store and another part of the data in a ROLAP store, allowing us to exploit the advantages of each.