On-Line Analytic Processing (OLAP)

Warehouses
Data Cubes
Outer Join

Instructor: Shel Finkelstein

Reference:

A First Course in Database Systems, 3rd edition, Sections 5.2, 6.3.8, 10.6 and 10.7

Some slides taken from courses taught by Jeffrey Ullman and Hector Garcia-Molina at Stanford.

Important Notices

- Final Exam is on Wednesday, March 22, noon-3pm in our usual classroom.
  - Final is Cumulative, with more focus on second half of quarter.
  - Please bring a red Scantron sheet (ParSCORE form number f-1712) sold at the Bookstore, and #2 pencils. (Some questions will be multiple choice.)
    - Ink and #3 pencils <u>don't</u> work.
  - You may bring a single two-sided 8.5" x 11" sheet of paper with as much info written (or printed) on it as you can fit and read unassisted, just as for the Midterm.
    - No sharing of these sheets will be permitted.
  - You must show your UCSC id when you turn in your Final and Scantron.
  - The Final from Fall 2016 has been posted on Piazza (Resources → Exams). Answers to that Final will be posted during the last week of classes.
- Gradiance Assignment #5 (on Functional Dependencies and Normal Forms) is due by Friday, March 17, 11:59pm.

More Important Notices

- Lab4 assignment was posted on Monday, Feb 27.
  - Due by Sunday, March 12, 11:59pm (2 weeks).
  - Lab4 focusses on material in Lecture 10 (Application Programming), including JDBC and Stored Procedures/Functions.
  - If you don't attend Lectures and Labs, you probably will find Lab4 difficult.
- There will be Lab Sections during the last week of classes.
  - These Lab Sections are an opportunity go over the answers to Lab4 and other Labs, or ask questions about other course material.
- Online course evaluations began Monday, March 5, and run through Sunday, March 19 at 11:59pm.
  - Instructors are not able to identify individual responses.
  - Constructive responses help improve future courses.

Overview

- Originally, database systems were tuned to many, small, simple queries (OLTP).
- Many applications use fewer, more time-consuming, *analytic* queries (OLAP).
- New architectures were developed to handle analytic queries efficiently.

Data Warehouses

- One common approach to data integration of multiple data sources
  - Copy sources into a single DB (*warehouse*), and try to keep data reasonably up-to-date.
  - Methods:
    - Periodic reconstruction of the warehouse, perhaps overnight
    - Periodic incremental update of warehouses
    - “Continuous” incremental update of warehouse
- Warehouses are frequently used for analytic queries.
  - Alternative approach: Leave data in separate data sources, and execute “mediated” query across those sources
  - Advantages and disadvantages of Warehouse vs. Mediation?

OLTP

- Most database operations involve *On-Line Transaction Processing* (OTLP).
  - Short, simple, frequent queries and/or modifications, each involving a relatively small number of tuples.
  - Examples: Answering queries from a Web interface, sales at cash registers, selling airline tickets.

OLAP

- *On-Line Analytic Processing* (OLAP, or "analytic") queries are different from OLTP.
  - Few, but complex queries which may take minutes/hours to execute.
  - Databases may be quite large—terabytes is common, but warehouses may have petabytes, exabytes or more.
  - Sometimes, queries do not require having a fully up-to-date database.
    - Why/when is it okay to use data that is not absolutely current, or data that is incomplete?

From Bytes to Yottabytes

Multiples of bytes					V · T · E
SI decimal prefixes		Binary usage	IEC binary prefixes		
Name (Symbol)	Value		Name (Symbol)	Value	
kilobyte (kB)	10^3	2^{10}	kibibyte (KiB)	2^{10}	
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}	
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}	
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}	
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}	
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}	
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}	
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}	
See also: Multiples of bits · Orders of magnitude of data					

OLAP Examples

1. Amazon analyzes purchases by its customers to come up with a personalized screen listing products that are likely of interest to the customer.
2. Analysts at Wal-Mart look for items whose sales in some region are increasing.
   - Send trucks to move merchandise between stores.
3. Google identified advertising "segments" (categories) of population, and displays advertisements based on individual's segment, as well as personal history.
4. Summary reports of product sales by Consumer Goods companies may be created monthly/weekly/daily/hourly …
   - Automatically report trends and anomalies and react to them

Common Architecture-1

1-Before Cloud Computing:

- Database systems at store branches handle OLTP.
- Local store databases are copied to a data Warehouse overnight
  - … or perhaps warehouse is incrementally updated quickly, with only the changed data copied.
- Analysts use the Warehouse for OLAP.
- Older data may be archived.
  - But even “cold” data is kept for a long time, possibly forever. (Why?)

Common Architecture-2

2-With Cloud Computing:

- Data systems for store branches may be stored in a cloud, using database schema suitable for OLTP.
  - May be in one or more databases.
  - Current OLTP data may be used for decision support.
- Local store databases are copied to a data Warehouse overnight
  - … or perhaps warehouse is incrementally updated quickly, with only the changed data copied.
- Analysts use the Warehouse for OLAP.
- Older data may be archived.

Some Modern Architectures

There are modern system approaches that combine:

- Streaming data (new events such as sensor data and stock quotes)
- OLTP transactions reading and writing data, such as banking transactions or order entry
- OLAP analytics, which we'll discuss in this lecture
  - Data Science analytics, such as customer categorization

Some of these approaches involved multiple copies of data, e.g., row data format for OLTP and column data format for OLAP.

- Some systems handle OLTP and OLAP using one copy of data

Star Schemas

- A *Star Schema* is a common organization for data in a Warehouse. A Star Schema consists of Dimension Tables and a Fact Table.

1. *Dimension Tables*: Smaller, largely static (unchanging) information describing the data that's in the Facts.
   - Examples: Product, Customer, Store
2. *Fact Table*: A very large accumulation of Facts, such as Sales
   - A Fact gives the Sales of a specific product to a specific customer in a specific store on a specific date.
   - The key for a Fact Table consists of values from its Dimension Tables (foreign keys).
   - Facts may be "insert-mostly", with some updates.

Example: Star Schema

- Suppose we want to record, in a warehouse, the information about every beer sale:
  - the bar,
  - the brand of beer,
  - the drinker who bought the beer,
  - the day of the purchase, and
  - the price charged

- The Fact Table is a relation:

  Sales(bar, beer, drinker, day, price)

Example -- Continued

- The Dimension Tables provide information about the bar, beer, and drinker "dimensions":

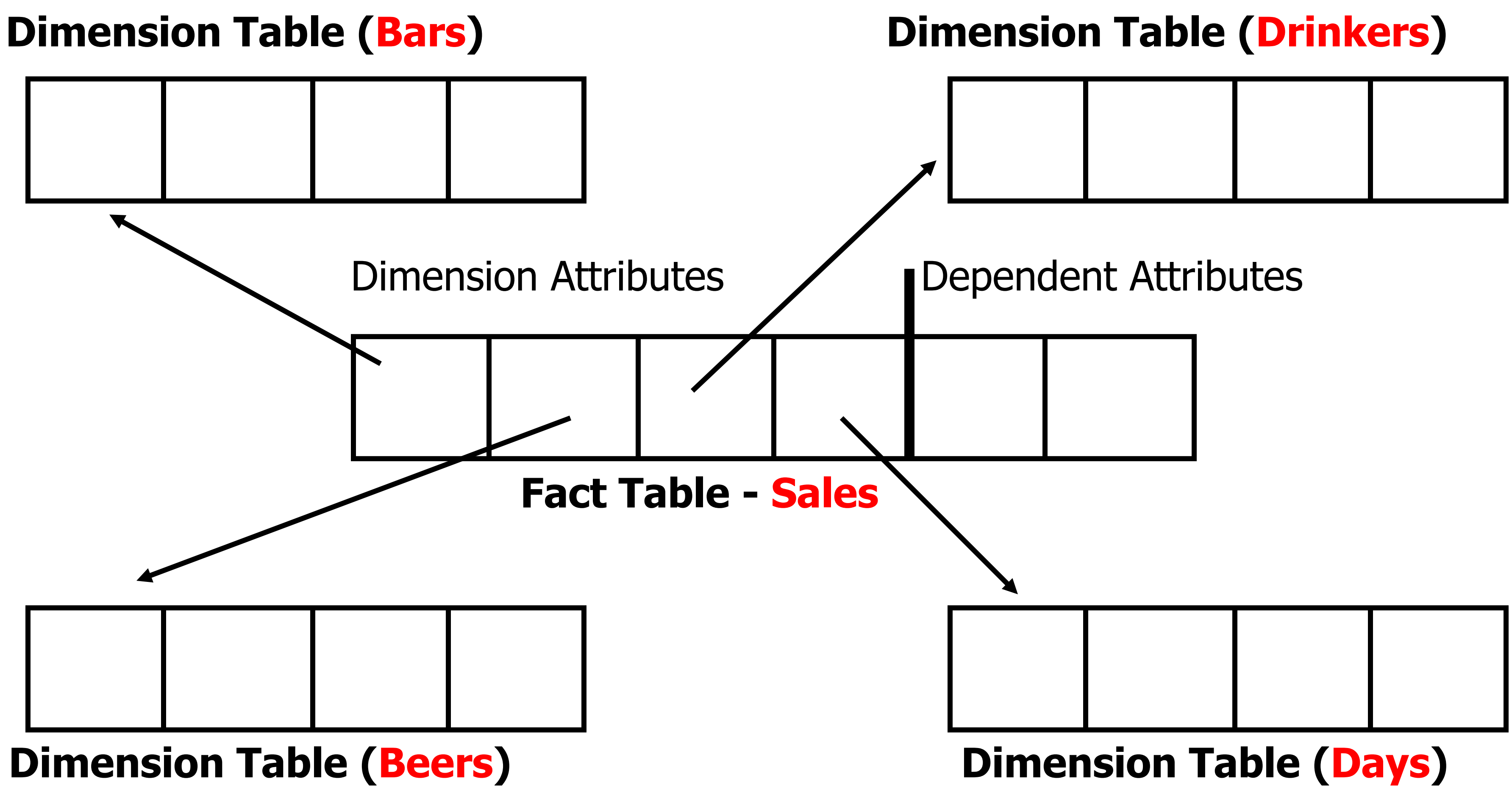
  Bars(bar, addr, license)

  Beers(beer, manf)

  Drinkers(drinker, addr, phone)

  Days(month, daynumber, year)

Visualization – Star Schema



Dimension and Dependent Attributes

- Two classes of Fact Table attributes:
  1. *Dimension Attribute*: the key of a dimension table.
  2. *Dependent Attribute*: a fact value determined by the dimension attributes of the tuple.
     - The sales info (e.g., price, quantity, salesperson) for a specific <u>product</u> to a specific <u>customer</u> at a specific <u>store</u>
     - The price of a specific <u>beer</u> purchase by a specific <u>drinker</u> at a specific <u>bar</u> on a specific <u>day</u>

Dimension and Dependent Attributes and Fact Table

The key of a Fact Table is the combination of the keys of its dimensions

- Values of dimension attributes in any Fact must match dimension attribute values in the Dimension Tables.
- But there don't have to be Facts for every combination of Dimension values.

Example:

- If there's a Fact saying that:

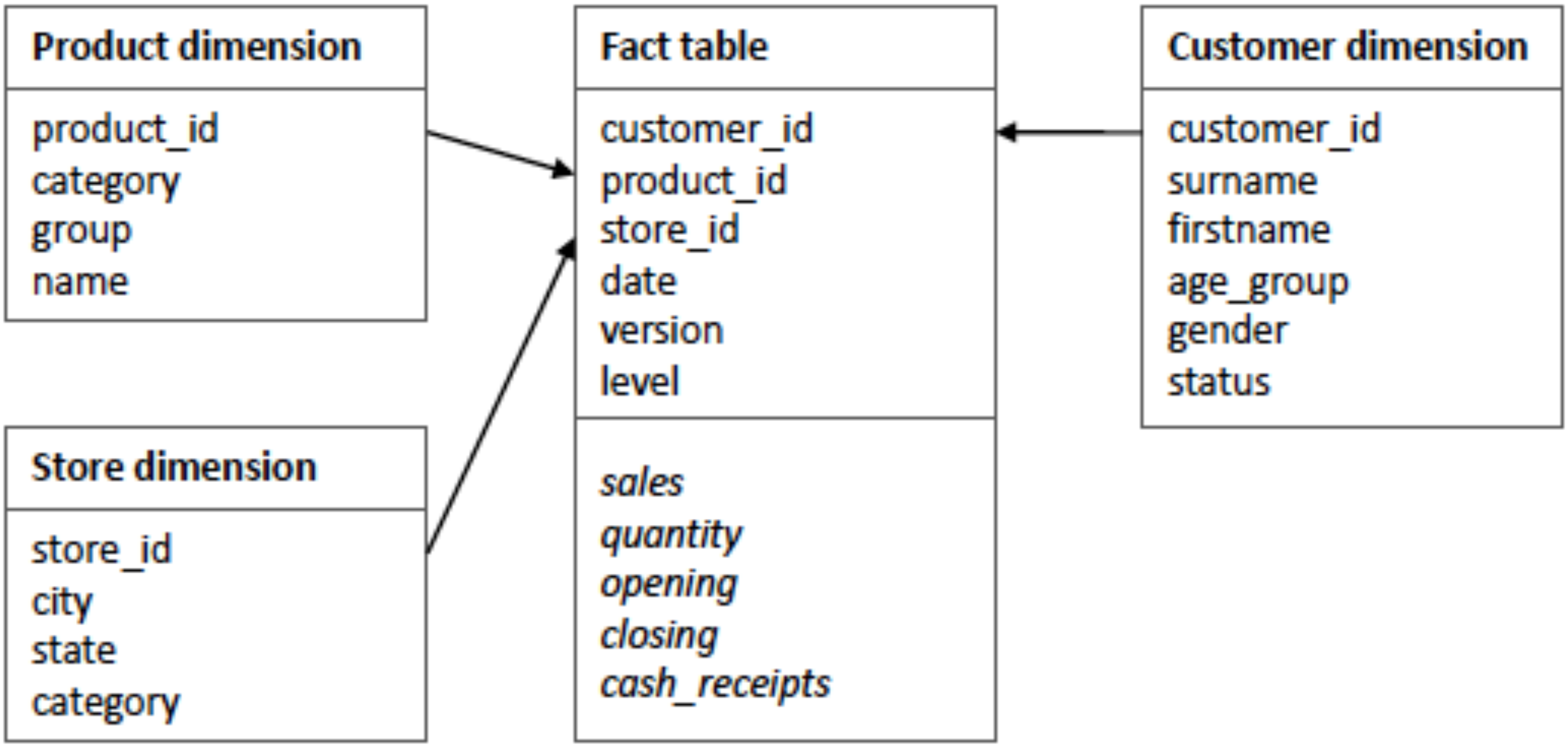
  George bought Bud at Joe's Bar on Jan 23, 2015 (at some price),

  then those values must be in the Dimension Tables.
- But even though those values are in the Dimension Tables, there doesn't have to be a Fact for them.

Example: Dependent Attribute

- price is a dependent attribute in our example Sales relation.
- That attribute is determined by the combination of dimension attributes: bar, beer, drinker and day.
- Time is sometimes treated as a <u>dimension</u> (specific month, day or hour) and sometimes treated as a <u>dependent</u> attribute.
  - For example, if you're recording specific second/msec, you're probably treating time as a dependent attribute, not as a dimension.

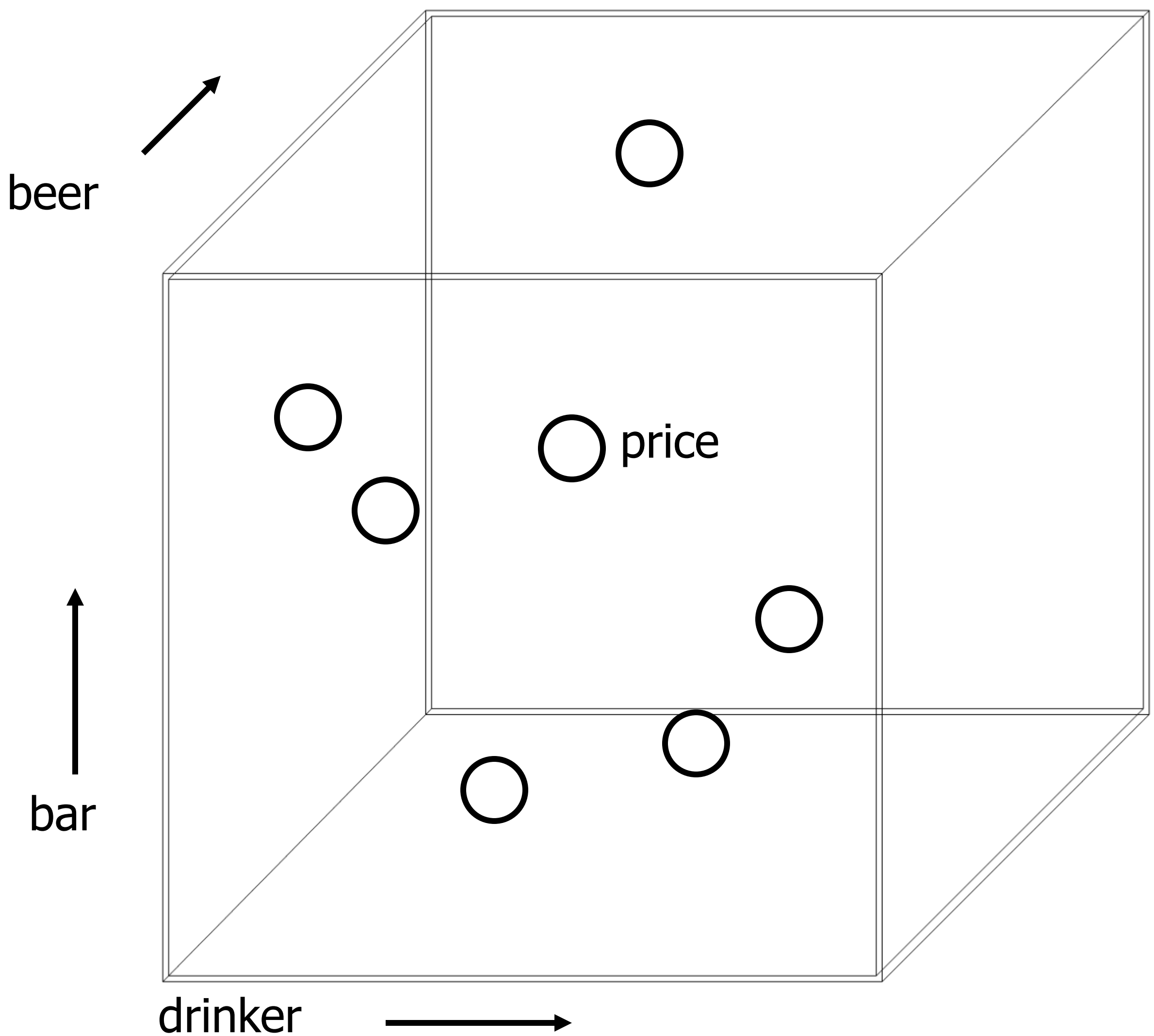
Another OLAP Example



Data Cubes

- Keys of dimension tables are the dimensions of a hypercube.
  - Example: For the beer Sales data, the four dimensions are bar, beer, drinker and day.
- Dependent attributes (e.g., price and quantity) appear as labels for points in the cube.

Visualization -- Data Cubes

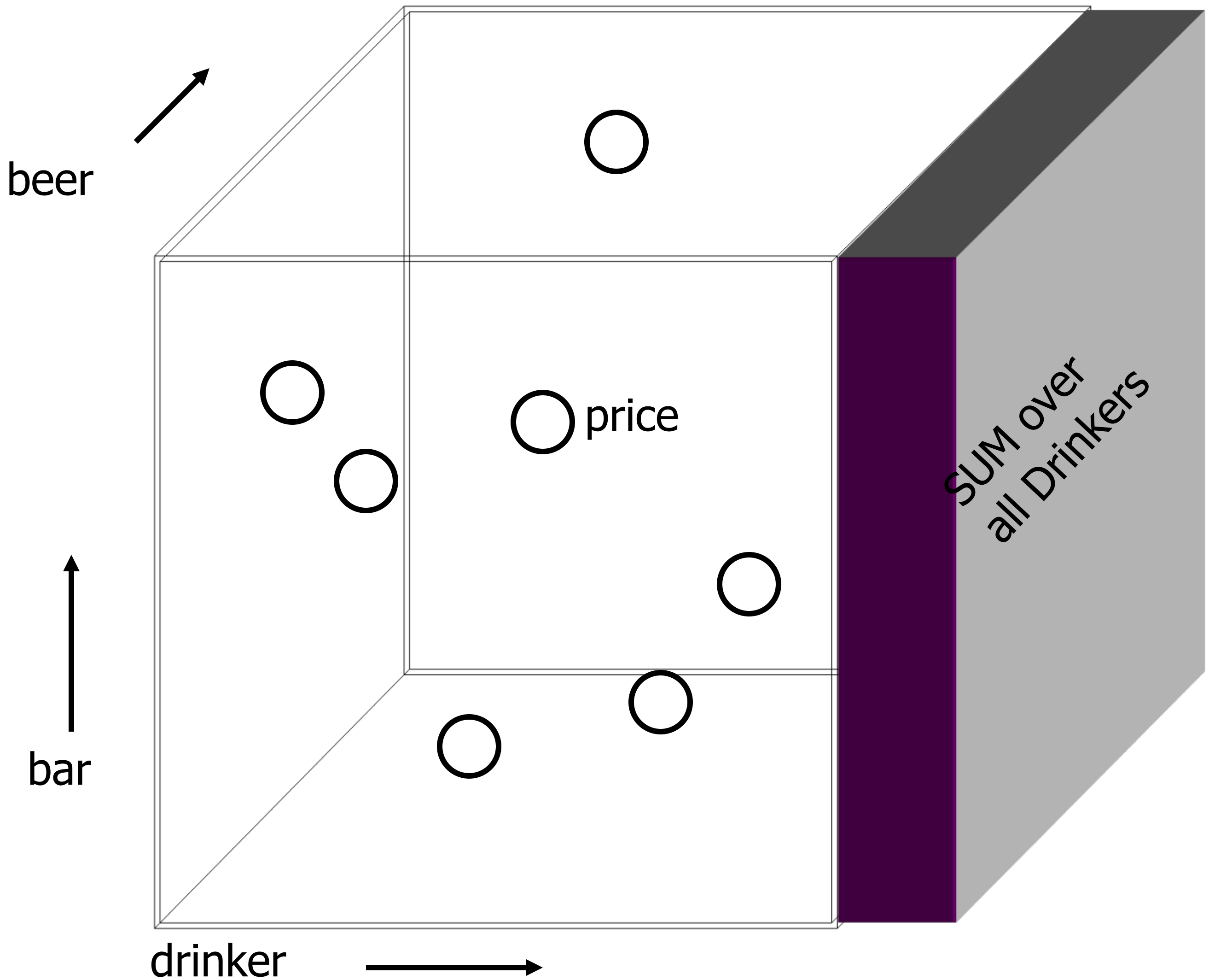


Measures

- The Data Cube also (logically) includes aggregations (e.g., SUM) along the margins of the cube.
- These *measures* (which textbook calls *marginals*) include aggregations over one dimension, two dimensions, …

Visualization --- Data Cube with Aggregation

What's the total sales of each beer sold by each bar?



Sums on the Cube

How can we write the total price for each bar and beer, summing over all drinkers, as a GROUP BY?

```
SELECT bar, beer, SUM(price)
FROM Sales
GROUP BY bar, beer;
```

What does this represent?

```
SELECT drinker, SUM(price)
FROM Sales
GROUP BY drinker;
```

The total spent by each drinker, summing over all bars and beers?

Example: Measures

- Our 3-dimensional Sales cube includes the sum of price over each bar, each beer, and each drinker.
  - Summing over each drinker was first example on previous slide.
  - Could go 4-dimensional and have day as fourth dimension.
- It could also have the sum of price for each drinker over all bar-beer pairs, (see second example on previous slide), all beer-day pairs, …, all bar-drinker-day triples, …
- Question: Do the aggregates have to be stored, and maintained every time a relevant fact is inserted, updated or deleted?

Structure of the Cube

- Think of each dimension as having an additional value *, meaning “everything”.
- A point with one or more *’s in its coordinates aggregates over the dimensions with the *’s.
- Examples:
  - Sales(”Joe’s Bar”, ”Bud”, *, *) holds the Sum (over all drinkers and all days) of the Bud beers consumed at Joe’s Bar.
  - Sales(bar, beer, *, *) holds the Sum (over all drinkers and all days) for any bar and beer.
- Sum isn’t the only “Measure/Marginal”.
  - Average, Count and more complex statistical formulas are sometimes used.

Roll-Up

- *Roll-up* means aggregate along one or more dimensions.
- Example: Given a Fact Table showing how much Bud each drinker consumes at each bar, roll it up into a table giving the total amount of Bud consumed by each drinker.

Drill-Down

- *Drill-down* means "dis-aggregate", that is, break an aggregate into its constituent parts.
- Example: Having determined that Joe's Bar sells very few Anheuser-Busch beers, break down his sales by each particular A-B beer.
- Can be done accurately only when underlying the Fact Table has all the data for each A-B beer.

Example: Roll-Up and Drill-Down

$ of Anheuser-Busch by drinker/bar

	Jim	Bob	Mary
Joe's Bar	45	33	30
Nut-House	50	36	42
Blue Chalk	38	31	40

Roll-up by Bar

$ of A-B / drinker

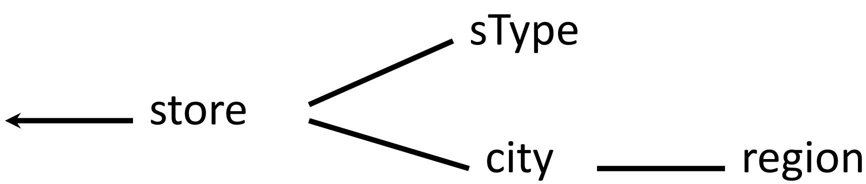
Jim	Bob	Mary
133	100	112

Drill-down by Beer

$ of A-B Beers / drinker

	Jim	Bob	Mary
Bud	40	29	40
M'lob	45	31	37
Bud Light	48	40	35

Dimension Hierarchies



store	storeId	cityId	tId	mgr
	s5	sfo	t1	joe
	s7	sfo	t2	fred
	s9	la	t1	nancy

sType	tId	size	location
	t1	small	downtown
	t2	large	suburbs

city	cityId	pop	regId
	sfo	1M	north
	la	5M	south

region	regId	name
	north	cold region
	south	warm region

- ➔ star schema
- ➔ snowflake schema
- ➔ fact constellations

Aggregates

- Add up amounts for day 1
- In SQL: SELECT sum(amt) FROM SALES
  WHERE date = 1

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

⇨ 81

Aggregates

- Add up amounts by day
- In SQL: SELECT date, sum(amt) FROM SALES
  GROUP BY date

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

ans	date	sum
	1	81
	2	48

Another Example

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

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

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

rollup →

← drill-down

OLAP Aggregates

- Operators: sum, count, max, min, median, avg, …
- “Having” clause
- Using dimension hierarchy
  - Average by region (across stores in region)
  - Maximum by month (across dates in month)

Outer Join Motivation

- Suppose we join relations R and S on some join condition.
- A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- **Outer Join** preserves dangling tuples by padding them with NULL.
  - Assumes that attributes allow NULL.

Reminder: Join (Inner Join)

```
SELECT * FROM R, S WHERE R.B = S.B;
SELECT * FROM R JOIN S ON R.B = S.B;
SELECT * FROM R INNER JOIN S ON R.B=S.B;
```

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples, (4,5) and (6,7), are dangling.

Example: Outer Join

SELECT * FROM R OUTER JOIN S WHERE R.B = S.B;

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

R OUTER JOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Outer Joins

R OUTER JOIN S is the core part of an Outer Join expression.

It can be modified by:

1. Optional ON <condition> after JOIN.
2. Optional LEFT, RIGHT, or FULL before OUTER.
   - LEFT means pad dangling tuples of R only.
   - RIGHT means pad dangling tuples of S only.
   - FULL means pad both; this choice is the default.
     - OUTER JOIN means FULL OUTER JOIN

Left and Right Outer Join

What is the result of the following?

SELECT * FROM R **LEFT** OUTER JOIN S WHERE R.B = S.B;

What is the result of the following?

SELECT * FROM R **RIGHT** OUTER JOIN S WHERE R.B = S.B;

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

Examples: Left /Right Outer Join

SELECT *
FROM Movies LEFT OUTER JOIN StarsIn
ON title = movieTitle AND year = movieYear

This gives Movie tuples with any star who StarsIn that movie,
and NULL-padded Movie tuples for which there's no star who StarsIn that movie,
but won't include StarsIn tuples where stars doesn't star in any movie in Movies.

SELECT *
FROM Movies RIGHT OUTER JOIN StarsIn
ON title = movieTitle AND year = movieYear

This gives StarsIn tuples where the listed movie is in Movies,
and NULL-padded StarsIn tuples for which movie listed isn't in Movies,
but won't include movies for which there's no star that's in StarsIn.

OLAP and OUTER JOIN

Joining a set of Dimensions

… (not necessarily all of the Dimensions)

… and then taking LEFT OUTER JOIN of result with Fact Table
… will give you entries for **every** combination of Dimensions,
… **not just the ones** that have entries in the Fact Table.

There may not be any Facts for a given Day, but you'd like that Day to show up in your report.

- Example: Summing up Sales amount across all Stores and Products and Days, that "missing" Day's total Sales amount should be 0.
- You can get that by using Outer Join. (Well, actually you get NULL.)

How do you change NULL value to 0?

- One common way to do this is with the **Coalesce** function.
- COALESCE(x, 0) has value x if x isn't NULL, and value 0 if x is NULL.

Data Mining

- *Data mining* is a popular term for queries that summarize big data sets in useful ways.
- Examples:

1. Clustering all Web pages by topic.
2. Finding characteristics of fraudulent credit-card use.

Market-Basket Data

- An important form of mining from relational data involves *market baskets* = sets of "items" that are purchased together as a customer leaves a store.
- Summary of basket data is *frequent itemsets* = sets of items that often appear together in baskets.

Example: Market Baskets

- If people often buy hamburger and ketchup together, the store can:
  1. Put hamburger and ketchup near each other and put potato chips between.
  2. Run a sale on hamburger and raise the price of ketchup.
- **A Priori** is one simple algorithm for finding frequent itemsets, but we won't discuss that.
  - There are Data Mining and Machine Learning courses at UCSC covering Data Mining problems and algorithms.

Finding Frequent Pairs

- The simplest case is when we only want to find "frequent pairs" of items.
- Assume data is in a relation Baskets(basket, item).
- The *support threshold* *s* is the minimum number of baskets in which a pair of items appears before we are interested in the pair.

Frequent Pairs in SQL

```
SELECT b1.item, b2.item
FROM Baskets b1, Baskets b2
WHERE b1.basket = b2.basket
  AND b1.item < b2.item
GROUP BY b1.item, b2.item
HAVING COUNT(*) >= s;
```

Look for two Basket tuples with the same basket and different items. First item must precede second, so we don't count the same pair twice.

Create a group for each pair of items that appears in at least one basket.

Throw away pairs of items that do not appear at least *s* times.

A-Priori Trick – (1)

- Straightforward implementation of Frequent Pairs involves join of a huge Baskets relation with itself.
- The *a-priori algorithm* speeds the query by recognizing that a pair of items { *i*, *j* } cannot have support *s* unless both { *i* } and { *j* } do.

- Reminder: The *support threshold* *s* is the minimum number of baskets in which a pair of items appears before we are interested in the pair.
  - Item { i } has support s if it appears in at least s baskets.

A-Priori Trick – (2)

- Use a materialized view to hold only information about frequent items.

```
INSERT INTO BasketsOne(basket,
  item)
SELECT * FROM Baskets
WHERE item IN (
  SELECT item FROM Baskets
  GROUP BY item
  HAVING COUNT(*) >= s
);
```

Items that appear in at least *s* baskets.

A-Priori Algorithm

1. Materialize the view BasketsOne.
2. Run the “obvious” query, but on BasketsOne instead of Baskets.
   - Computing BasketsOne is cheap, since it doesn’t involve a join.
   - What is the “obvious” query?

- BasketsOne *probably* has many fewer tuples than Baskets.
  - The running time decreases with the *square* of the number of tuples involved in the join.

Example: A-Priori

- Suppose:
  1. A supermarket sells 10,000 items.
  2. The average basket has 10 items.
  3. The support threshold is 1% of the baskets.
- At most 1/10 of the items can be frequent, i.e., in 1% of the baskets.
  - Why?
- *Probably*, the minority of items in one basket are frequent, hence speedup of BasketOne to BasketOne join, compared to Basket to Basket join.