On-Line Analytical 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)

CMPS 182 Final Exam is on **Monday June 10, 4:00 – 7:00pm**, in our usual classroom.

- **No** early/late Finals, **no** make-up Finals.
- **No** devices.
- Includes a Multiple Choice Section and a Longer Answers Section.
  - Bring Red Scantron sheets (ParSCORE form number f-1712) sold at Bookstore, and #2 pencils for Multiple Choice Section.
    - Ink and #3 pencils don't work.
- Covers entire quarter, with slightly greater emphasis on second half of quarter.
- You may bring in one double-sided 8.5 by 11 sheet, with anything that you can read unassisted printed or written on both sides of the paper.
  - **No sharing** of sheets is permitted.
  - Include name on top right of sheet. Sheets will be collected with Finals.
- You **must** show your UCSC ID at end of Final.
- Practice Final from Spring 2017 (2 Sections) is on Piazza under Resources-->Exams
  - Answers have also been posted (one file) in the same location..

Important Notices

Gradiance #5 was assigned on Tuesday, May 28, and is due on
Friday, June 7 by 11:59pm.

- You should now have enough information to complete this Gradiance Assignment.
- Some of the questions may be difficult.

Might have time to do a very brief review of one topic on Friday, June 7.

- Probably will discuss BCNF and 3NF question from Practice Final.

Spring 2019 Student Experience of Teaching Surveys - SETs are now open, and SETs close on Sunday June 9 at 11:59pm.

- SETs is the new term for campus-wide student course evaluations.
- Instructors **are not** able to identify individual responses.
- Constructive responses help improve future courses.

See Small Group Tutoring website for LSS Tutoring with Chandler Hawkins.

Overview

- Originally, database systems were tuned to many, small, simple queries (OLTP).
- Many applications use fewer, more time-consuming, *analytical* queries (OLAP).
- New architectures were developed to handle analytical 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 analytical queries.
  - Alternative approach: Leave data in separate data sources, and execute "mediated" query across those sources
  - Advantages and disadvantages of Warehouse vs. Mediation?

OLTP

- Database operations we've discussed before 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 Analytical Processing* (OLAP, or "analytic") queries are different from OLTP.
  - Fewer but more 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

Some Modern Architectures

There are modern system approaches that combine capabilities for:

- OLTP transactions, such as Banking transactions and Order Entry
- OLAP analytics, which we'll discuss in this lecture
  - Data Science analytics, such as customer categorization
- Streaming data (new events, such as sensor data and stock quotes)

Some of these approaches involved multiple copies of data for availability and for performance.

- Some systems use row-based representation for OLTP and column-based representation for OLAP.

Data is often stored in the Cloud, often in files, rather than in databases.

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 that we want to record 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)

Example: Star Schema (modified)

- Suppose that we want to record 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

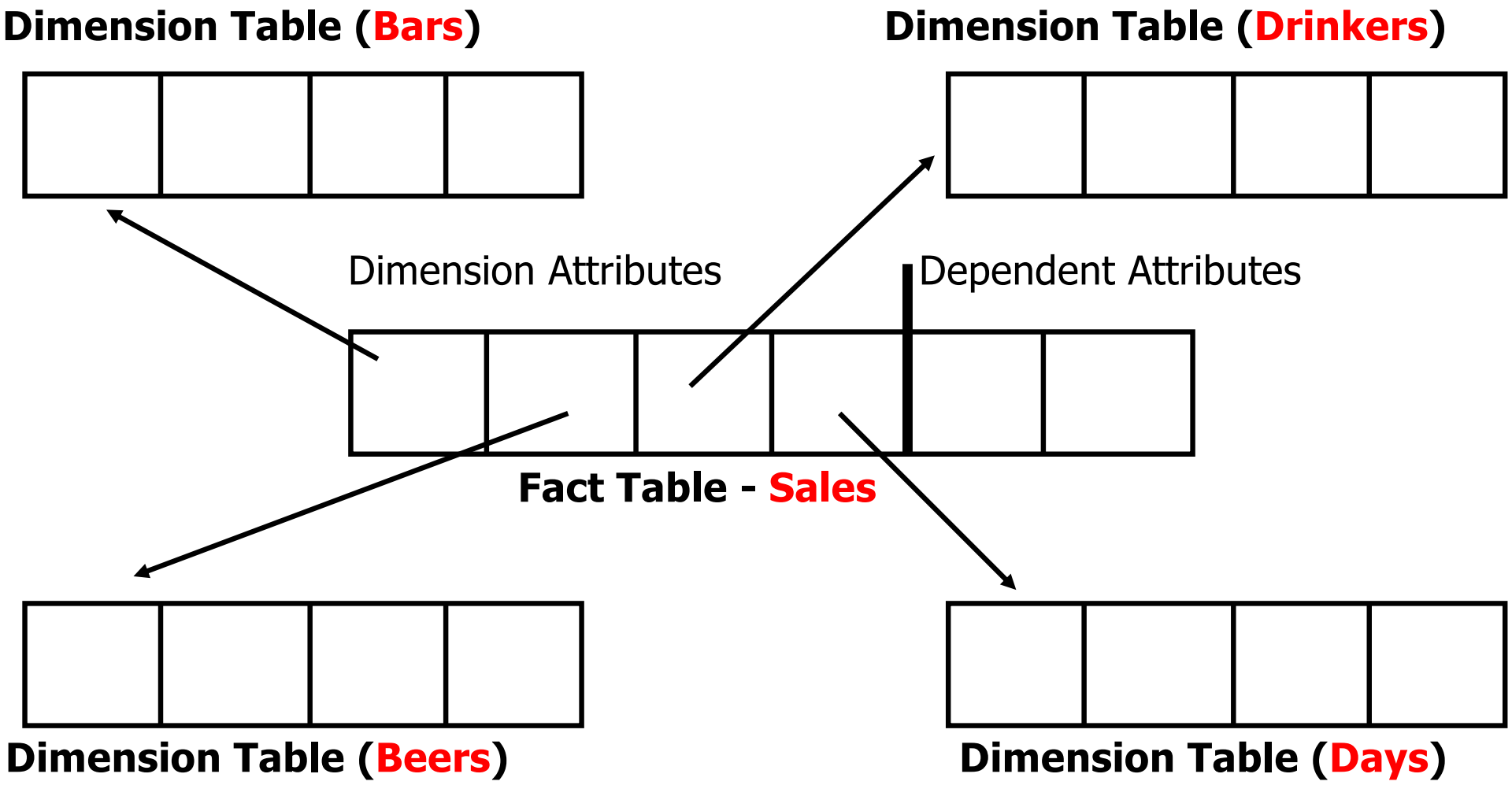
- The Fact Table is a relation:

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

  - Since primary key of Days is month, daynumber, year, should be:

    Sales(bar, beer, drinker, month, daynumber, year, price)

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 product to a specific customer at a specific store
     - The price of a specific beer purchase by a specific drinker at a specific bar on a specific day

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:

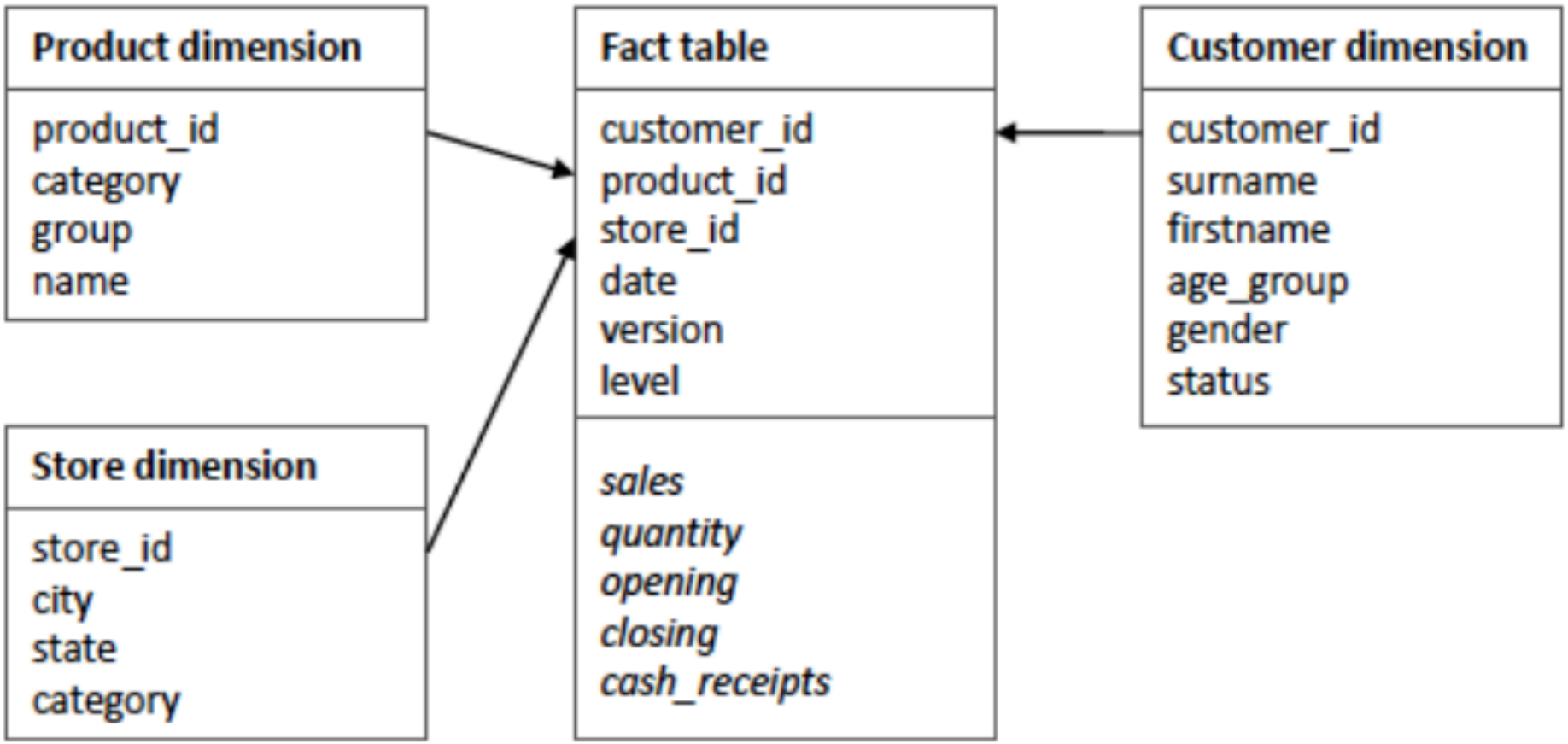
  George bought Bud at Joe's Bar on Jan 23, 2015 (at some price),

  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 dimension (specific month, day or hour) and sometimes treated as a dependent attribute.
  - For example, if you're recording specific second/msec, you're probably treating time as a dependent attribute, not as a dimension.

Another (Familiar) OLAP Example

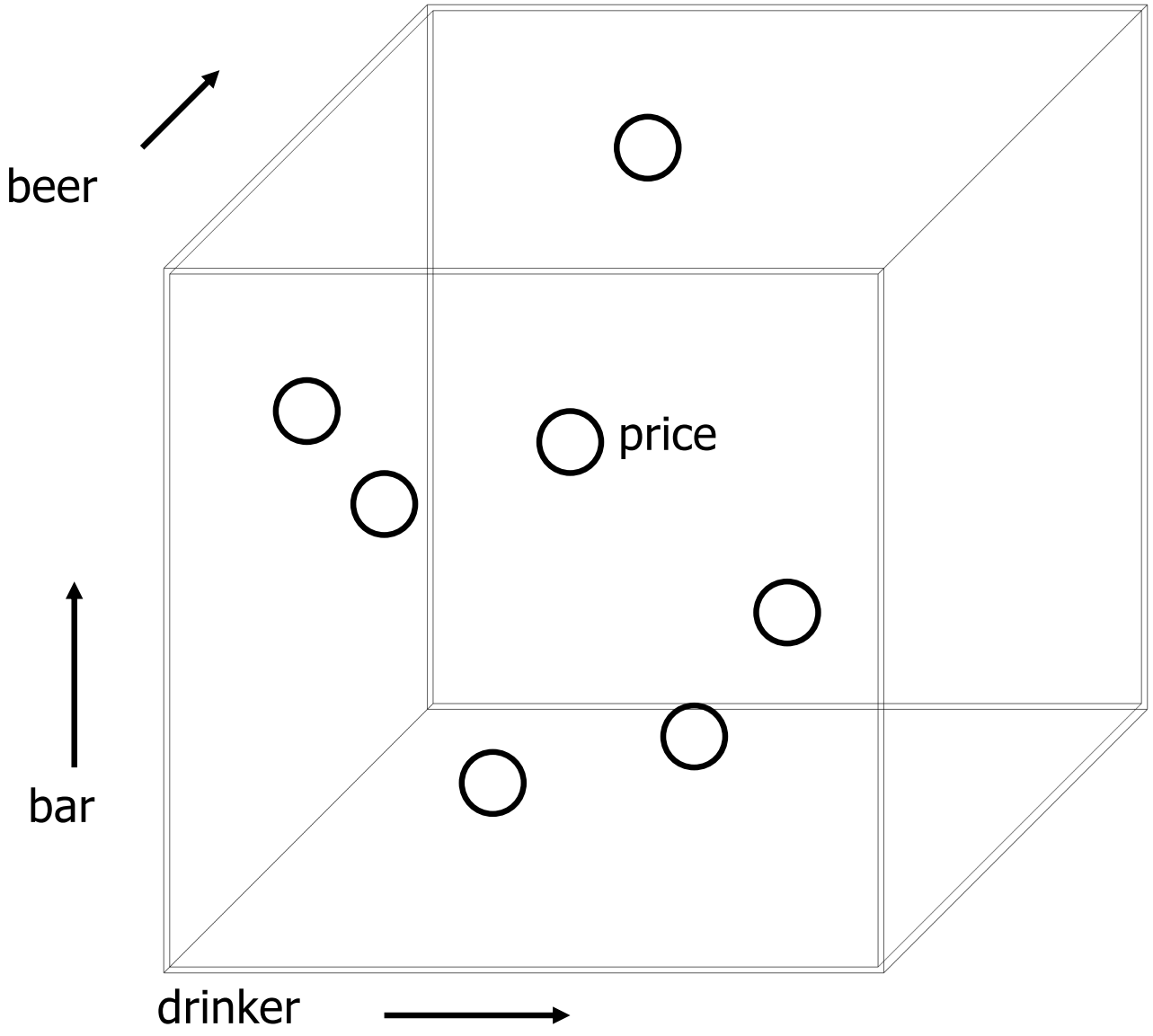


Data Cubes

- 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

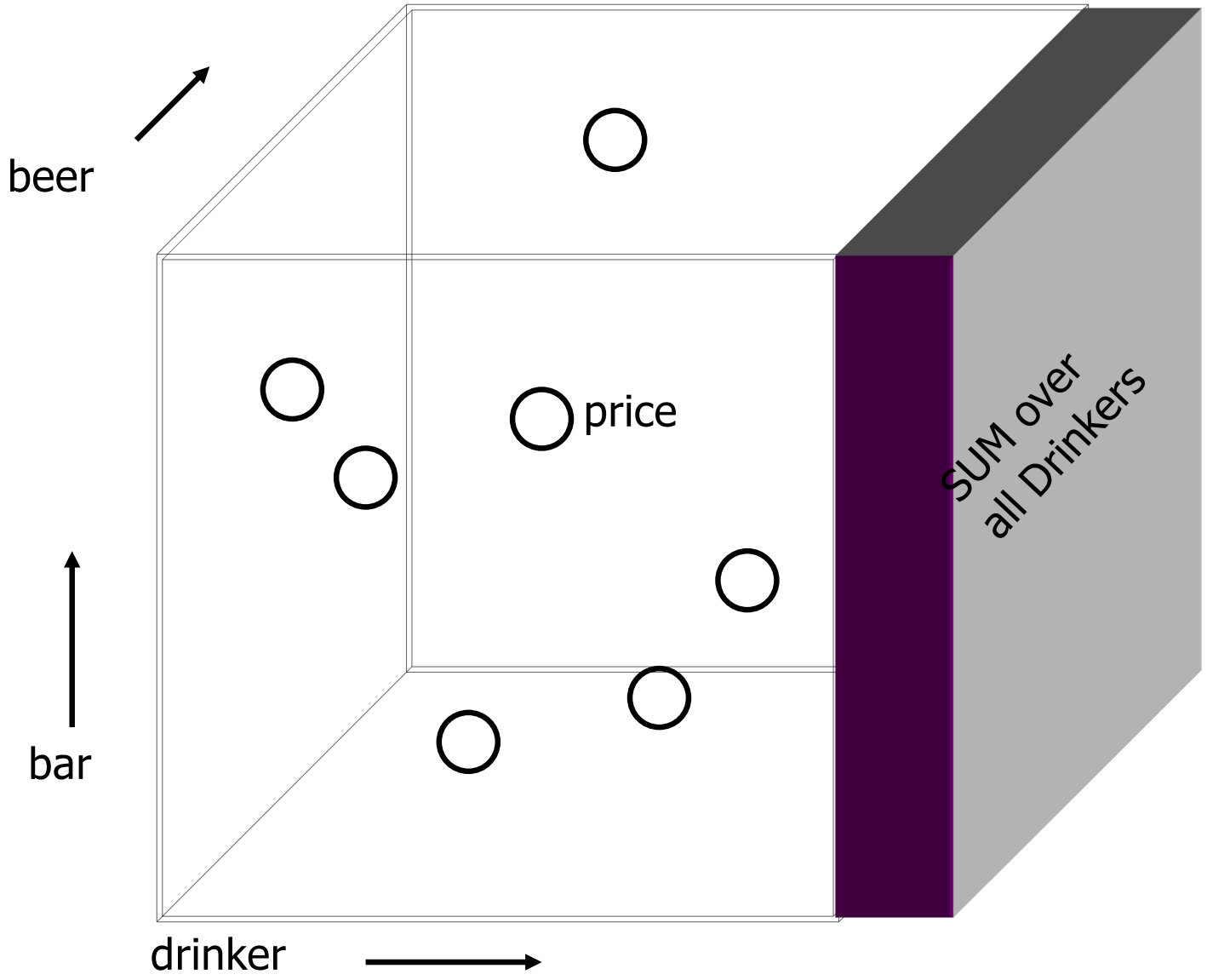


Measures

- 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.
- Drill-Down can be done accurately only when we also have the underlying Fact Table, which has full data for every A-B beer.
  - Doing Drill-Down approximately still has value; how can we do that?

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

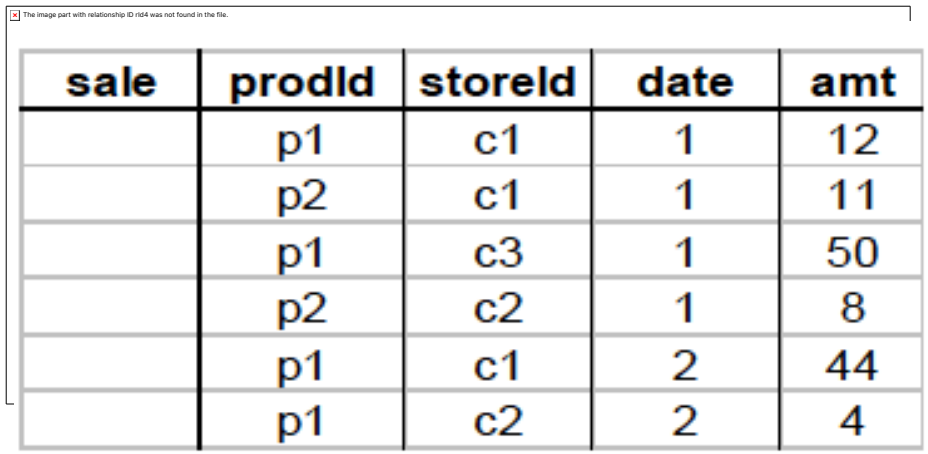
Drill-down by Beer

$ of A-B Beers / drinker

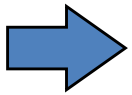
	Jim	Bob	Mary
Bud	40	29	40
M'lob	45	31	37
Bud Light	48	40	35

Aggregates and Roll-Up (1)

- Add up the 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



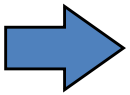
81

Aggregates and Roll-Up (2)

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

The picture can't be displayed.

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

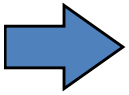


ans	date	sum
	1	81
	2	48

Roll-Up and Drill-Down

- 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

Roll-Up

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

Note: We should be showing R.B and S.B separately in these slides.

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

Taking the Cartesian Product of the Dimension Table Keys …

… and then taking LEFT OUTER JOIN of that with the 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.

Sales Records Example without Outer Join

Sales(productID, customerID, storeID, dayDate, paidPrice, quantity)
*The cost of a sale in Sales is paidPrice * quantity.*

Find the storeID, storeName and the total cost for the Sales for every store, **including for the stores that have no Sales.**

```
SELECT st.storeID, st.storeName, SUM(sal.paidPrice*sal.quantity)
FROM Stores st, Sales sal
WHERE st.storeID = sal.storeID
GROUP BY st.storeID, st.storeName

UNION

SELECT st.storeID, st.storeName, 0
FROM Stores st
WHERE NOT EXISTS
        ( SELECT * FROM Sales sal
          WHERE st.storeID = sal.StoreID );
```

Sales Records Example with Outer Join

Sales(productID, customerID, storeID, dayDate, paidPrice, quantity)

*The cost of a sale in Sales is paidPrice * quantity.*

Find the storeID, storeName and the total cost for the Sales for every store, **including for the stores that have no Sales.**

```
SELECT st.storeID, st.storeName,
                    SUM( COALESCE( sal.paidPrice, 0) * COALESCE( sal.quantity, 0 )  )
FROM Stores st LEFT OUTER JOIN Sales sal
WHERE st.storeID = sal.storeID
GROUP BY st.storeID, st.storeName;
```