CSE 180, Database Systems I
Shel Finkelstein, UC Santa Cruz
Lecture 11

Chapter 11: Data Analytics (Online Analytic Processing, OLAP)

Database System Concepts, 7th Ed.

Important Notices

CSE 180 Final Exam is on **Thursday March 19, 4:00 – 7:00pm**, but **it will be given via Canvas, so you must have web access**.

- A Piazza post this weekend will explainihow Canvas will be used for Final.
- **No** early/late Finals. **No** make-up Finals. **No** devices (except to take exam).
- Includes a Multiple Choice Part, and two Long Answers Parts, with time limits.
  - Red Scantron sheets are not required, since you're submitting solution via Canvas.
- Covers entire quarter, with slightly greater emphasis on second half of quarter.
- **During the Final, you may view Lecture slides and Lab Assignment solutions, plus one double-sided 8.5 x 11 sheet (with anything that you want written or printed on it), but you must not do any web searches or receive assistance from anyone else during the Final.**
  - **Your agreement to Academic Integrity terms is required before the Final!**
- Practice Final from Fall 2019 (2 Sections) was posted on Piazza under Resources→Exams on Sunday, March 1.
  - Solution has also been posted on Piazza… but take it yourself first, rather than just reading the solution.
- See Piazza notice @194 about Final, which has reminder about Course Grading.
  - After the class ends, your course score will be determined by your scores on Gradiance, Lab Assignments and Exams.
  - **You won't be able to do any additional work afterwards to improve your grade.**

Important Notices

- Lab3 grades have been unmuted.
  - Deadline for questions about Lab3 is **Thursday, March 12**.
- Lab4 was due Sunday, March 8, by 11:59pm.
  - Solution has been posted on Piazza.
- Gradiance #5 is due **Saturday, March 14** by 11:59pm.
- Attend Lectures, Lab Sections, Office Hours and LSS Tutoring.
  - Most sessions will be held via Zoom; see Piazza notices.
  - My Office Hours on Wed, March 11 will be 3:00-5:00pm, held by Hangout with students who request time via email.
  - See Piazza notices about LSS Tutoring with Jeshwanth Bheemanpally.
- Winter 2020 Student Experience of Teaching Surveys - SETs are open.
  - SETs close on Sunday March 15 at 11:59pm.
  - Instructors **are not** able to identify individual responses.
  - Constructive responses help improve future courses.

Chapter 11: Data Analytics

- Overview
- Data Warehousing
- Online Analytical Processing
- Data Mining

Overview

- **Data analytics**: the processing of data to infer patterns, correlations, or models for prediction
- Primarily used to make business decisions
  - Per individual customer
    - E.g., what product to suggest for purchase
  - Across all customers
    - E.g., what products to manufacture/stock, in what quantity
- Critical for businesses today

Overview (continued)

- Common steps in data analytics
  - Gather data from multiple sources into one location
    - Data warehouses also integrated data into common schema
    - Data often needs to be **extracted** from source formats, **transformed** to common schema, and **loaded** into the data warehouse
      - Can be done as **ETL (extract-transform-load)**, or **ELT (extract-load-transform)**
  - Generate aggregates and reports summarizing data
    - Dashboards showing graphical charts/reports
    - **Online analytical processing (OLAP) systems** allow interactive querying
    - Statistical analysis, using tools such as R, SAS, and SPSS
      - Including extensions for parallel processing of big data
  - Build **predictive models** and use the models for decision making

Overview (continued)

- Predictive models are widely used today.
  - E.g., use customer profile features (e.g. income, age, gender, education, employment) and past history of a customer to predict likelihood of default on loan
    - and use prediction to make loan decision
  - E.g., use past history of sales (by season) to predict future sales
    - And use it to decide what/how much to produce/stock
    - And to target customers
- Other examples of business decisions:
  - What items to stock?
  - What insurance premium to charge?
  - To whom to send advertisements?

Overview (continued)

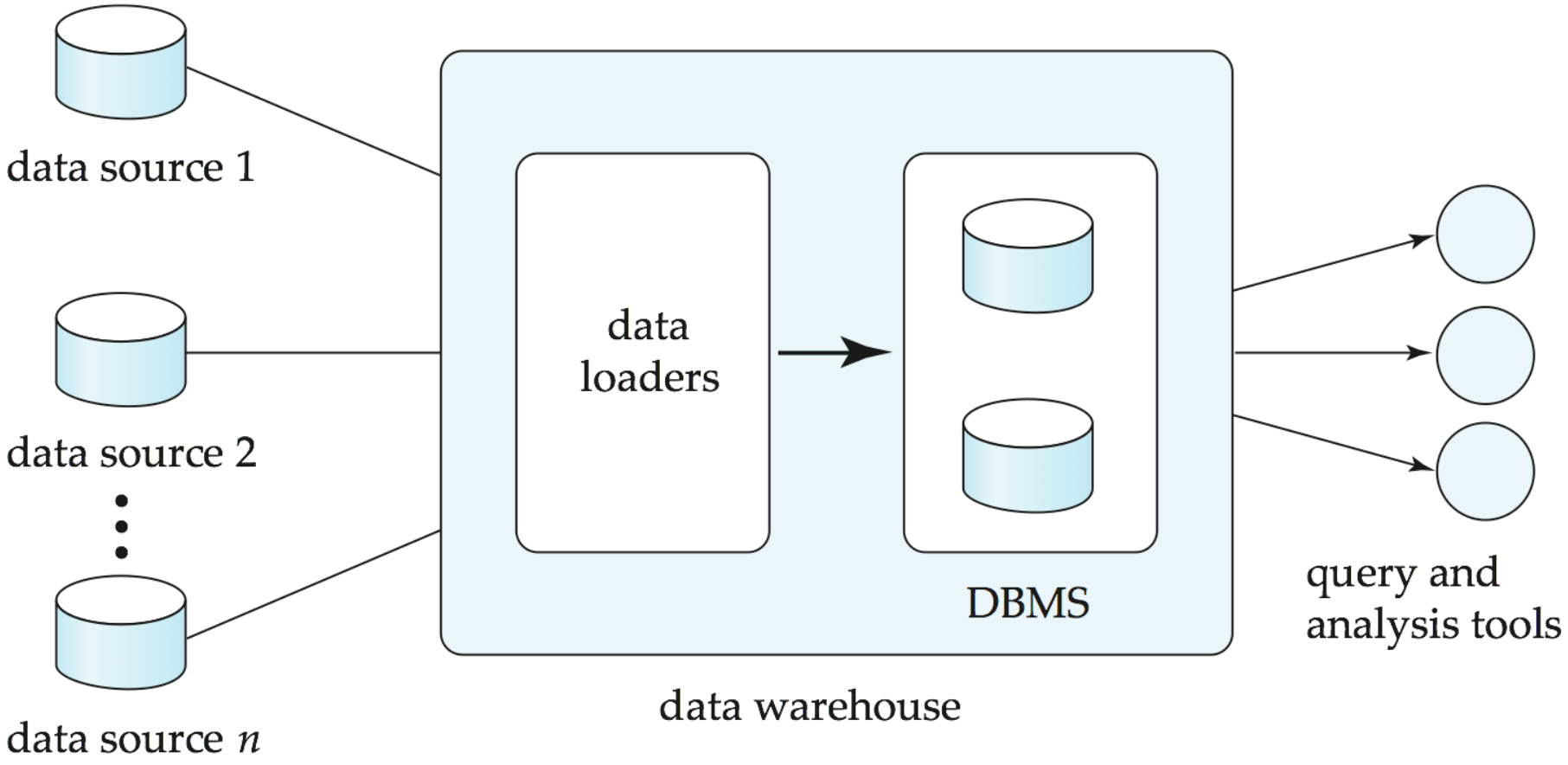
- **Machine Learning** techniques are key to finding patterns in data and making predictions
- **Data Mining** extends .techniques developed by machine-learning communities to run them on very large datasets.
- The term **Business Intelligence (BI)** is synonym for data analytics.
- The term **Decision Support** focuses on reporting and aggregation.

DATA WAREHOUSING

Data Warehousing

- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
  - Greatly simplifies querying, permits study of historical trends
  - Shifts decision support query load away from transaction processing systems

Data Warehousing



Design Issues

- *When and how to gather data*
  - **Source-driven architecture**: data sources transmit new information to warehouse
    - either continuously or periodically (e.g., at night)
  - **Destination-driven architecture:** warehouse periodically requests new information from data sources
  - **Synchronous** vs **asynchronous replication**
    - Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is often too expensive
    - Usually OK to have slightly out-of-date data at warehouse
    - Data/updates are periodically downloaded form online transaction processing (OLTP) systems.
- *What schema to use*
  - Schema integration

More Warehouse Design Issues

- **Data transformation** and **data cleansing**
  - E.g., correct mistakes in addresses (misspellings, zip code errors)
  - Merge address lists from different sources and purge duplicates
- *How to propagate updates*
  - Warehouse schema may be a (materialized) view of schema from data sources
    - View maintenance
- *What data to summarize*
  - Raw data may be too large to store on-line
  - Aggregate values (totals/subtotals) often suffice
  - Queries on raw data can often be transformed by query optimizer to use aggregate values

Multidimensional Data Schema

Data in Warehouses can be divided into Fact and Dimension tables.

- **Fact tables** are large.
  - E.g, *sales*(*item_id, store_id, customer_id, date, number, price*)
- **Dimension tables** are relatively small.
  - They provide additional information about items, stores, customers, and dates.
- A Fact table is similar to an array, whose subscripts are the keys of its Dimension tables.
  - A Fact tuple can only appear for a “subscript” if there are matching values in the Dimension tables.
  - But a Fact tuple doesn’t appear if there’s nothing to say, that is, if an item wasn’t purchased at a store by a customer on a particular date.
  - Most of the ”subscripts” don’t have corresponding Fact tuple.

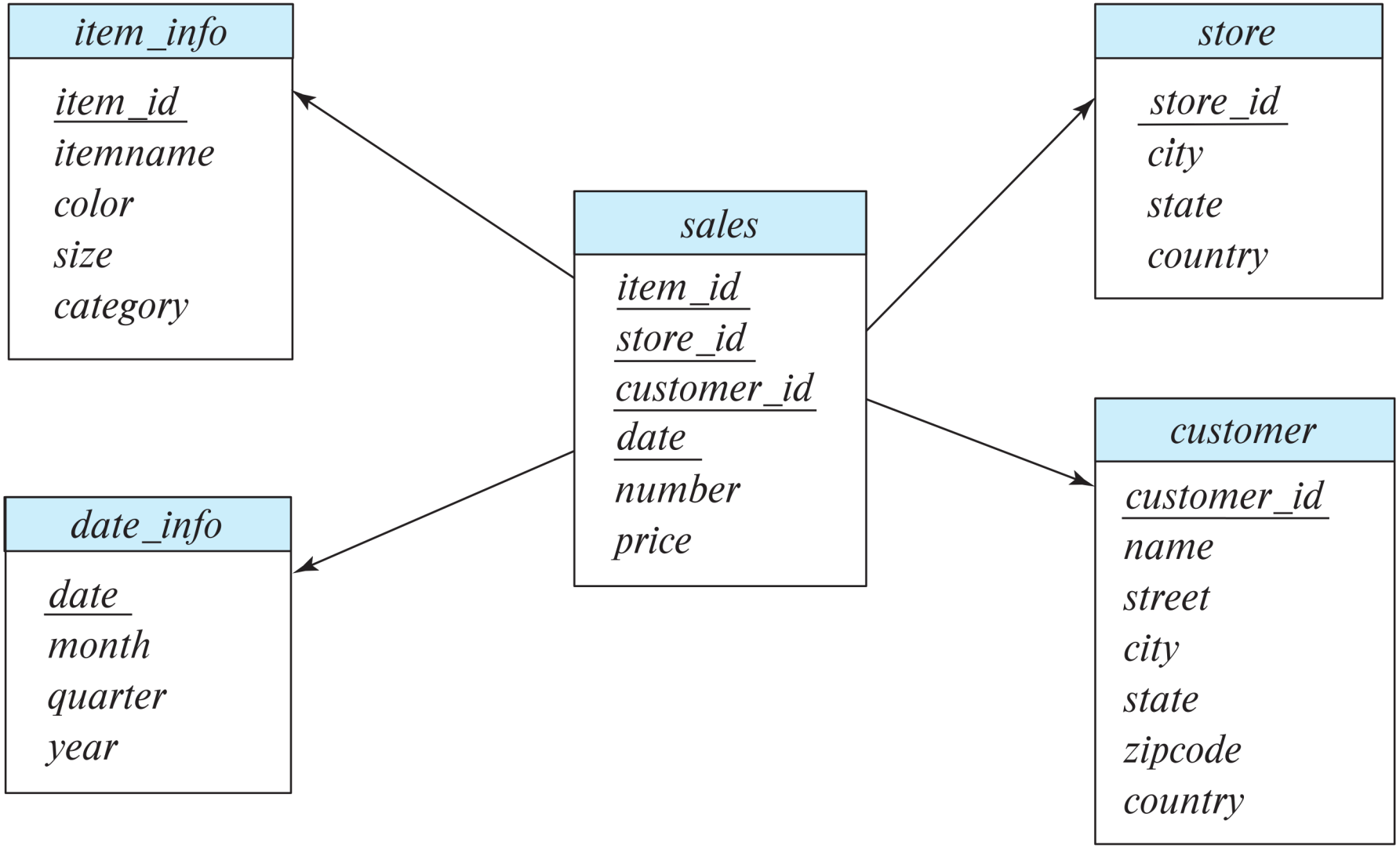
Fact Table Attributes

Attributes of Fact tables usually are either Dimension or Measure attributes.

sales(*item_id, store_id, customer_id, date*, *number, price*)

- **Dimension attributes**
  - For example *item_id, store_id* and *customer_id* and *date* are Dimension attributes of the *sales* relation
  - Dimension attributes in Fact tables are Foreign Keys referring to the Dimension tables.
  - The Primary key of the Fact table combines the keys of its Dimension tables
- **Measure attributes**
  - Measure attributes measure some value, and they can be aggregated.
    - For example, *number* and *price* attributes of the *sales* relation

Sales Schema



Multidimensional Data

- This is called a **Star Schema**
  - There may be multiple Fact tables that have different dimensions.
  - A **Snowflake Schema** has multiple levels of dimension tables.
- In Analytic queries, typically:
  - A Fact table is joined with some Dimension tables to get more information about the dimensions
    - E.g., Get city and state for each store, and then …
  - A GROUP BY is performed on some Dimension table attributes
    - E.g., Grouping by store and date, and then …
  - Measure attributes of the Fact table are aggregated
    - E.g., Calculating the sum of number*price for each store and date (over all customers on all items)

Database Support for Data Warehouses

- Data in warehouses is usually append only, not updated.
  - Can avoid "concurrency control" overheads from conflicts.
- Data warehouses often use **column-oriented storage**.
  - E.g., a sequence of *sales* tuples is stored as follows:
    - Values of item_id attribute are stored as an array.
    - Values of store_id attribute are stored as an array.
    - And so on
  - Arrays are compressed, reducing storage, IO and memory costs significantly.
  - Queries can fetch only attributes that they care about, reducing IO and memory cost.
- Data warehouses often use parallel storage and query processing infrastructure
  - Distributed file systems, Map-Reduce, Hive, proprietary products, …
- Column-oriented storage is now frequently use for Decision Support, not just in Data Warehouses.

Alternative to Data Warehouses: Data Lakes

- Another approach is not to transform data to a common schema
  - **Data Lakes** are repositories which allow data to be stored in multiple formats, without schema integration.
  - Data can be stored rapidly, in near "raw" form.
    - … but there's much more effort during querying to clean and transform data appropriately to answer queries.
  - Having all data in Data Lake (stored in cloud) makes it access all of an enterprise's data (in one place).
    - But there are still issues about representation, integration, etc.

OLAP

Data Analysis and OLAP

- **Online Analytical Processing (OLAP)**
  - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay.)
- We will use the following relation to illustrate OLAP concepts
  - clothingS*ales* (*item_name*, *color*, *clothes_size*, *quantity*)

A clothingSales Relation Instance

item_name	*color*	*clothes_size*	*quantity*
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
pants	dark	small	14
pants	dark	medium	6
pants	dark	large	0
pants	pastel	small	1
pants	pastel	medium	0
pants	pastel	large	1
pants	white	small	3
pants	white	medium	0
pants	white	large	2
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
skirt	dark	small	2
skirt	dark	medium	5
...	...	...	...
...	...	...	...

Aggregation and GROUP BY

What does each of these queries do?

- **SELECT** *item_name, color,* **SUM**(quantity)
  FROM *clothingSales*
  GROUP BY *item_name, color*;

- **SELECT** *clothes_size,* **SUM**(quantity)
  FROM *clothingSales*
  GROUP BY *clothes_size*;

- **SELECT SUM**(quantity)
  FROM *clothingSales;*

Cross Tabulation of *clothingSales* by *item_name* and *color*

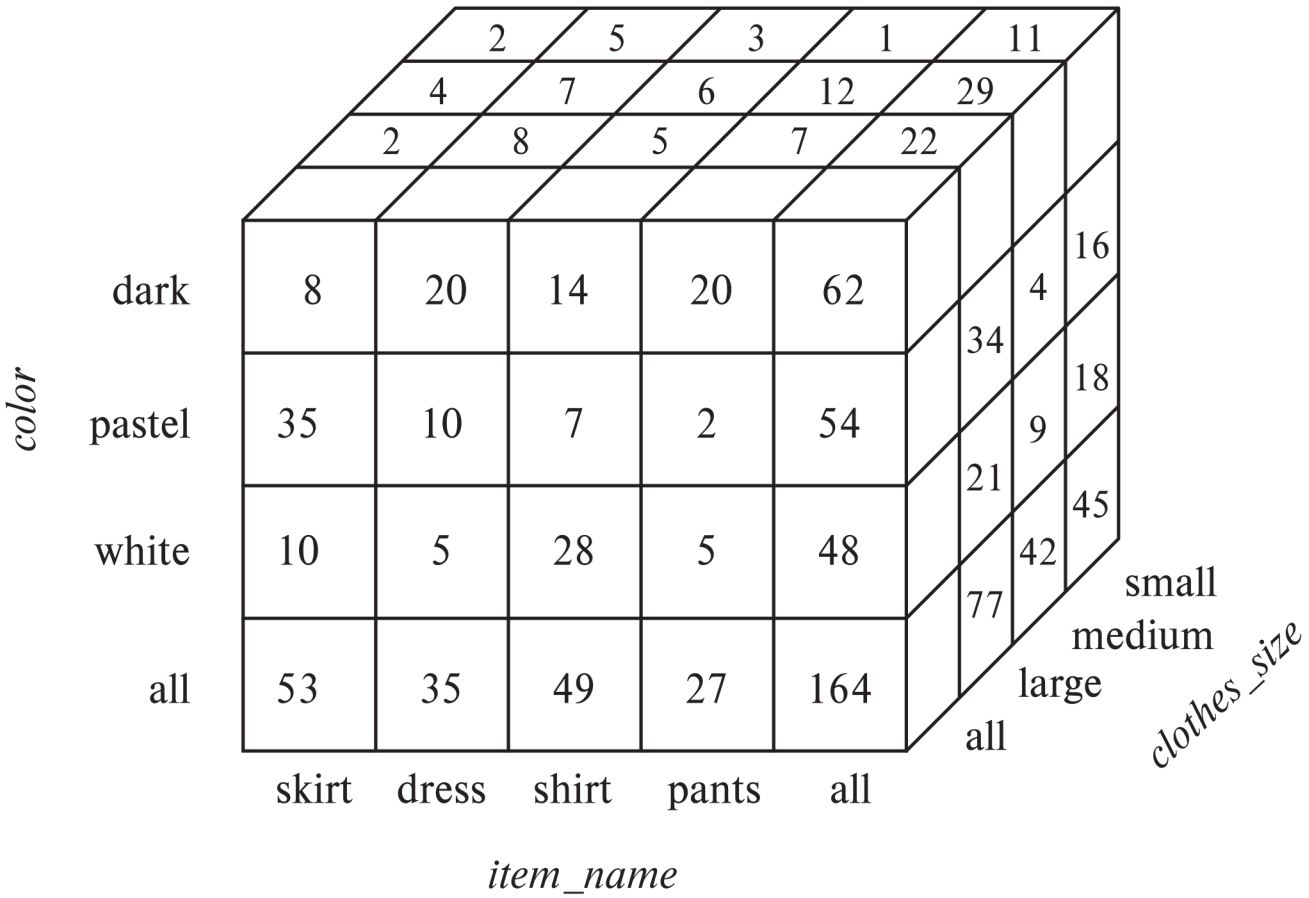
clothes_size **all**

item_name \ *color*	dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
  - Values for one of the dimension attributes (item_name) form the row headers.
  - Values for a second dimension attribute (color) form the column headers.
  - Other dimension attributes (clothes_size) are listed in a menu at the top.
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have n dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube



Online Analytical Processing Operations

- **Pivoting:** Changing the dimensions used in a cross-tab.
  - E.g., Moving clothes_size to column names.
- **Slicing:** Creating a cross-tab for fixed values only.
  - E.g., Fixing color to white and clothes_size to small.
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** Moving from finer-granularity data to a coarser granularity.
  - E.g., Aggregating away an attribute.
  - E.g., Moving from aggregates by day to aggregates by month or year.
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data.
  - Involves estimates if you don't have the original data.

Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
- We entered **all** on this slide to represent aggregation.
- The SQL standard actually uses **NULL** to represent aggregation.
  - Could cause confusion with regular uses of **NULL** values.
- COALESCE can convert NULL to other values, e.g., COALESCE(color, 'all');
  - If color isn't **NULL**, leaves it unchanged.
  - If color is **NULL**, 'all" appears.

item_name	*color*	*clothes_size*	*quantity*
skirt	dark	**all**	8
skirt	pastel	**all**	35
skirt	white	**all**	10
skirt	**all**	**all**	53
dress	dark	**all**	20
dress	pastel	**all**	10
dress	white	**all**	5
dress	**all**	**all**	35
shirt	dark	**all**	14
shirt	pastel	**all**	7
shirt	white	**all**	28
shirt	**all**	**all**	49
pants	dark	**all**	20
pants	pastel	**all**	2
pants	white	**all**	5
pants	**all**	**all**	27
all	dark	**all**	62
all	pastel	**all**	54
all	white	**all**	48
all	**all**	**all**	164

OLAP IN SQL

Pivot Operation

- **SELECT** *
  FROM *clothingSales*
  PIVOT (**SUM**(*quantity*)
  FOR *color* **in** ('dark','pastel','white'))
  ORDER BY *item_name*;

- Table shows data from clothingSales instance with colors as columns.
  - 2 dark small shirts
  - 4 pastel small shirts
  - 17 white small shirts

item_name	*clothes_size*	*dark*	*pastel*	*white*
dress	small	2	4	2
dress	medium	6	3	3
dress	large	12	3	0
pants	small	14	1	3
pants	medium	6	0	0
pants	large	0	1	2
shirt	small	2	4	17
shirt	medium	6	1	1
shirt	large	6	2	10
skirt	small	2	11	2
skirt	medium	5	9	5
skirt	large	1	15	3

Cube Operation

- The **CUBE** operation computes **UNION** of **GROUP BY**'s on every subset of the specified attributes
- E.g., consider the query

  SELECT *item_name, color, size,* **SUM**(*number*)
  FROM *sales*
  GROUP BY CUBE(*item_name, color, size*)

  This computes the union of eight different groupings of the *sales* relation:

  { (*item_name, color, size*), (*item_name, color*),
  (*item_name, size*), (*color, size*),
  (*item_name*), (*color*),
  (*size*), ( ) }

  where ( ) denotes an empty **GROUP BY** list.
- For each grouping, the result contains the **NULL** value for attributes not present in the grouping.

Online Analytical Processing Operations

- Relational representation of cross-tab that we saw earlier on Slide with title “Representation of Cross-tabs” (but with **NULL** in place of **all**), can be computed by:

```
SELECT item_name, color, SUM(number)
FROM sales
GROUP BY CUBE( item_name, color );
```

Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
- We entered **all** on this slide to represent aggregation.
- The SQL standard actually uses **NULL** to represent aggregation.
  - Could cause confusion with regular uses of **NULL** values.
- COALESCE can convert NULL to other values, e.g., COALESCE(color, 'all');
  - If color isn't **NULL**, leaves it unchanged.
  - If color is **NULL**, 'all" appears.

item_name	*color*	*clothes_size*	*quantity*
skirt	dark	**all**	8
skirt	pastel	**all**	35
skirt	white	**all**	10
skirt	**all**	**all**	53
dress	dark	**all**	20
dress	pastel	**all**	10
dress	white	**all**	5
dress	**all**	**all**	35
shirt	dark	**all**	14
shirt	pastel	**all**	7
shirt	white	**all**	28
shirt	**all**	**all**	49
pants	dark	**all**	20
pants	pastel	**all**	2
pants	white	**all**	5
pants	**all**	**all**	27
all	dark	**all**	62
all	pastel	**all**	54
all	white	**all**	48
all	**all**	**all**	164

Extended Aggregation (continued)

- The **ROLLUP** construct generates UNION on every **prefix** of the specified list of attributes
- **SELECT** *item_name*, *color*, *size*, **SUM**(*number*)
  FROM *sales*
  GROUP BY ROLLUP (*item_name, color, size*)

  Generates union of four different groupings:

  { (*item_name, color, size*), (*item_name, color*), (*item_name*), ( ) }

  with **NULL** used so that all results have same set of attributes.

OLAP Implementation

- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **Multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **Relational OLAP (ROLAP)** systems.
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **Hybrid OLAP (HOLAP)** systems.

OLAP Implementation (continued)

- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
  - Space and time requirements for doing so can be very high.
    - There can be 2^n combinations of **GROUP BY** attributes, where n is number of attributes.
  - Sometimes some aggregates are precomputed, and other aggregates are computed on-demand from precomputed aggregates
    - System can compute SUM on (*item_name, color*) from SUM on (*item_name, color, size*) by summing across all size values.
- *Many modern systems suggest not storing any aggregates, avoiding need to update aggregates when data changes.*
  - *Data is stored by column, instead of by row, and aggregation operations are very fast for column-oriented representations.*
  - *Systems may store data twice, in row-oriented and column-oriented forms, or convert data after it is entered, or choose one representation*
    - *Physical-independence hides this from applications.*

Reporting and Visualization

- **Reporting tools** help create formatted reports with tabular/graphical representation of data
  - E.g., SQL Server reporting services, Crystal Reports
- **Data Visualization** tools help create interactive visualization of data
  - E.g., Tableau, FusionChart, plotly, Datawrapper, Google Charts, etc.
  - Frontend typically based on HTML+JavaScript

Acme Supply Company, Inc.
Quarterly Sales Report

Period: Jan. 1 to March 31, 2009

Region	Category	Sales	Subtotal
North	Computer Hardware	1,000,000	
	Computer Software	500,000	
	All categories		1,500,000
South	Computer Hardware	200,000	
	Computer Software	400,000	
	All categories		600,000
		Total Sales	2,100,000

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 Fact tuples for a given date value, but you'd like that date value to show up in your report.

- Example: Summing up the sales quantity across all items, customers and stores, that "missing" date's total quantity 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**.

MovieTheater createView Example

For each showing (in Showings) of that movie, there may be ticket tuples (in Tickets) for that movie's movieID. The computedEarnings of a movie can be calculated by adding up ticketPrice for all the Tickets tuples that correspond to Showings of that movie.

Create a view called earningsView that has 2 attributes, movieID and computedEarnings This view should have a tuple for each movieID that gives the computedEarnings for that movieID.

If there's a movieID for which there are no tickets, then there still should be a tuple for that movieID in earningsView, and that tuple's computedEarnings should be 0.

MovieTheater createView Example without Outer Join

```
( SELECT S.movieID, SUM(T.ticketPrice) as computedEarnings
        FROM Showings S, Tickets T
        WHERE S.theaterID = T.theaterID
          AND S.showingDate = T.showingDate
          AND S.startTime = T.startTime
          AND S.movieID IS NOT NULL
        GROUP BY S.movieID )

   UNION

( SELECT M.movieID, 0 as computedEarnings
        FROM Movies M
        WHERE NOT EXISTS ( SELECT * FROM Showings S, Tickets T
                           WHERE S.theaterID = T.theaterID
                             AND S.showingDate = T.showingDate
                             AND S.startTime = T.startTime
                             AND S.movieID IS NOT NULL
                             AND M.movieID = S.movieID ) );
```

MovieTheater createView Example with Outer Join

```
CREATE VIEW earningsView AS
    SELECT M.movieID, COALESCE( CE.computedEarnings, 0 )
    FROM Movies M LEFT OUTER JOIN
            ( SELECT S.movieID, SUM(T.ticketPrice) AS computedEarnings
              FROM Showings S, Tickets T
              WHERE S.theaterID = T.theaterID
                  AND S.showingDate = T.showingDate
                  AND S.startTime = T.startTime
                  AND S.movieID IS NOT NULL
             GROUP BY S.movieID )  CE
    WHERE M.movieID = CE.movieID;
```