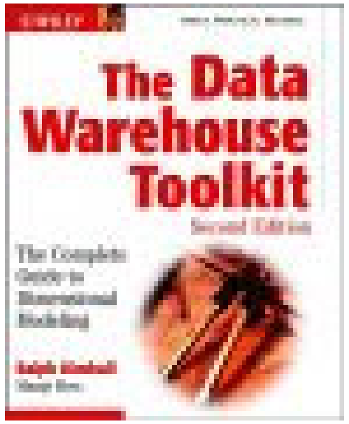
Data Warehousing & Dimension Modeling

TAN Kian-Lee
(tankl@comp.nus.edu.sg)
Department of Computer Science, NUS

Course information


The Data Warehouse Toolkit
Second Edition
The Complete Guide to Dimensional Modeling
Ralph Kimball
Margy Ross

- Reference text
  - “The Data Warehouse Toolkit” by Ralph Kimball and Margy Ross
  - Chapters 1, 2 and part of 4.
- Course project
  - Group-based (stay in the same group as your XML project)
  - Tentatively due Saturday 8pm (Week 13)
- Test 2 (Week 13)
  - Whatever is covered in lecture
- Acknowledgement: Lecture materials borrowed/adapted from Brian Babcock from Stanford

Two Distinct Workload in Databases: OLTP vs. OLAP

- OLTP (Operational): On-Line Transaction Processing
  - Many short transactions (queries + updates)
  - Examples:
    - Update account balance
    - Enroll in course
    - Add book to shopping cart
  - Queries touch small amounts of data (one record or a few records)
  - Updates are frequent
  - Concurrency is biggest performance concern
  - Data must be up-to-date, consistent at all times

database

- OLAP (Analytical): On-Line Analytical Processing
  - Long transactions, complex queries
  - Examples:
    - Report total sales for each department in each month
    - Identify top-selling books
    - Count classes with fewer than 10 students
  - Queries touch large amounts of data
  - Updates are infrequent
  - Individual queries can require lots of resources
  - Operating on static "snapshot" of data usually OK
  - Approximate answers may also be OK

data warehousing

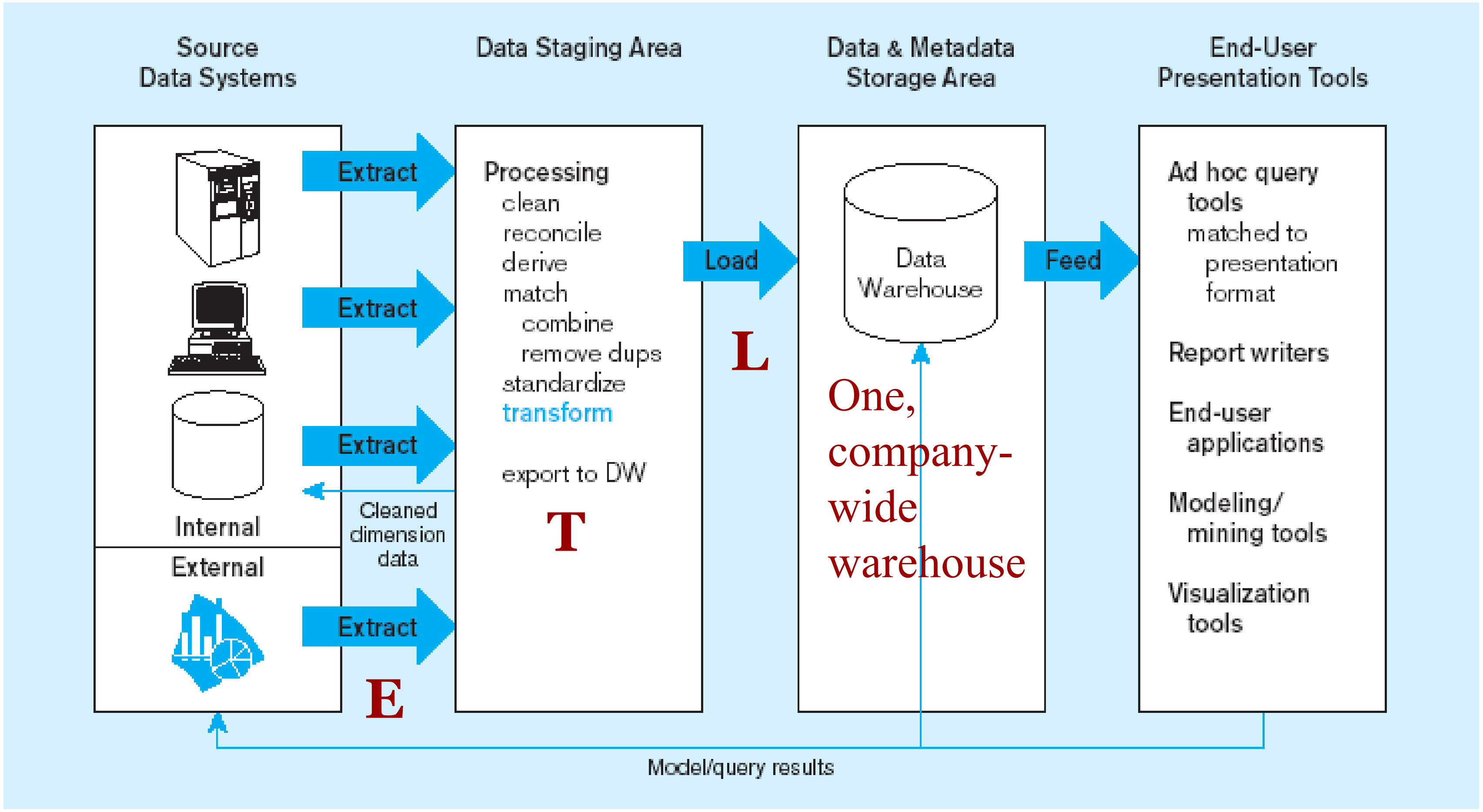
Why OLAP & OLTP don’t mix

- Different performance requirements
  - OLAP can “crowd out” OLTP transactions
    - Transactions are slow → unhappy users
  - Example:
    - Analysis query asks for sum of all sales
    - Acquires lock on sales table for consistency
    - New sales transaction is blocked
- Different data modeling requirements
  - Complex models (many tables) vs Simplicity (ease of understanding)
  - Normalized vs De-normalized schemas
  - Standardized/limited workload vs ad-hoc queries
- Analysis queries require data from many sources
  - Single sources vs integration of data from multiple sources
  - Current point-in-time vs identification of long-term patterns over historical data (e.g., changes in behavior over time)

Data Warehouses

- Doing OLTP and OLAP in the same database system is often impractical
- Solution: Build a “data warehouse”
  - Copy data from various OLTP systems
    - ETL tools – Extract, Transform, Load
  - Optimize data organization, system tuning for OLAP
  - Transactions aren’t slowed by big analysis queries
  - Periodically refresh the data in the warehouse

Generic two-level data warehousing architecture

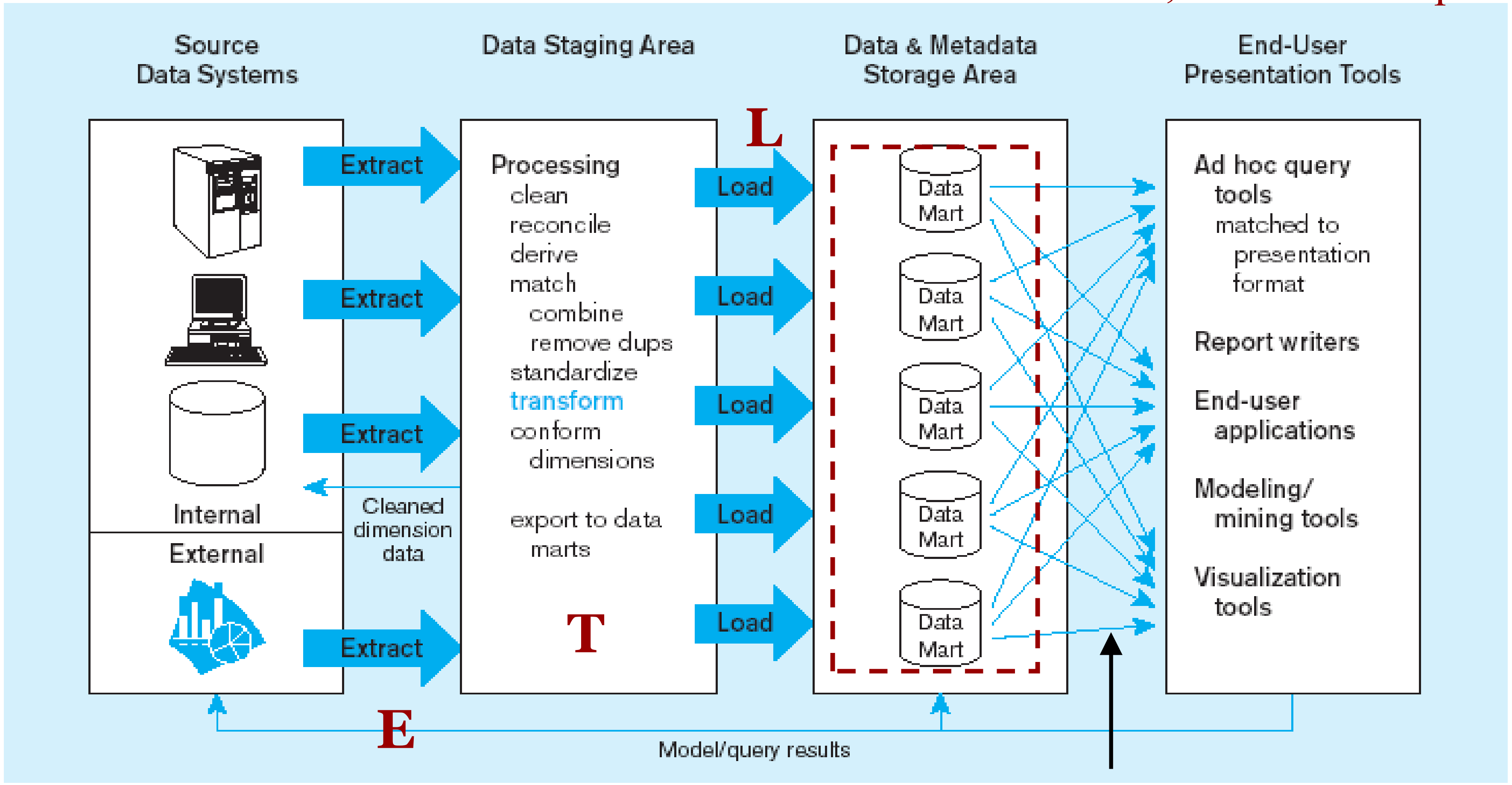


Periodic extraction ➔ data is not completely current in warehouse

Independent data mart data warehousing architecture

Data marts:

Mini-warehouses, limited in scope

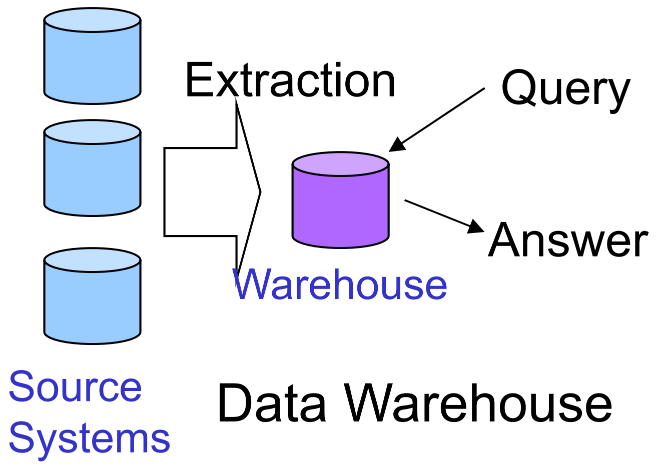


Separate ETL for each *independent* data mart

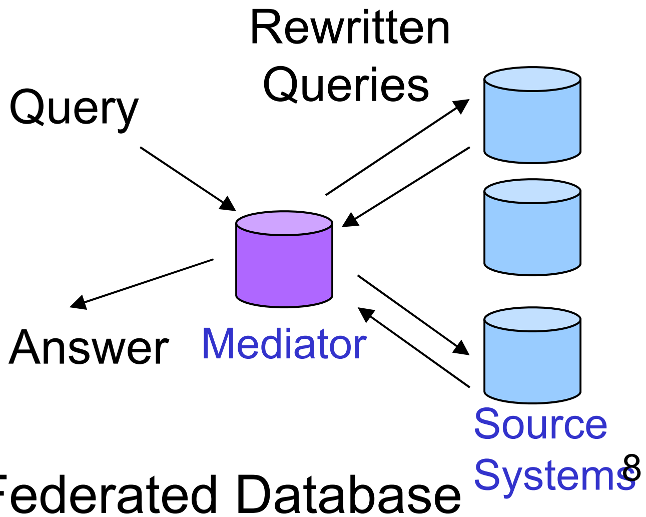
Data access complexity due to *multiple* data marts

Federated Databases

- An alternative to data warehouses
- Data warehouse
  - Create a copy of all the data
  - Execute queries against the copy
- Federated database
  - Pull data from source systems as needed to answer queries
- "lazy" vs. "eager" data integration



Data Warehouse



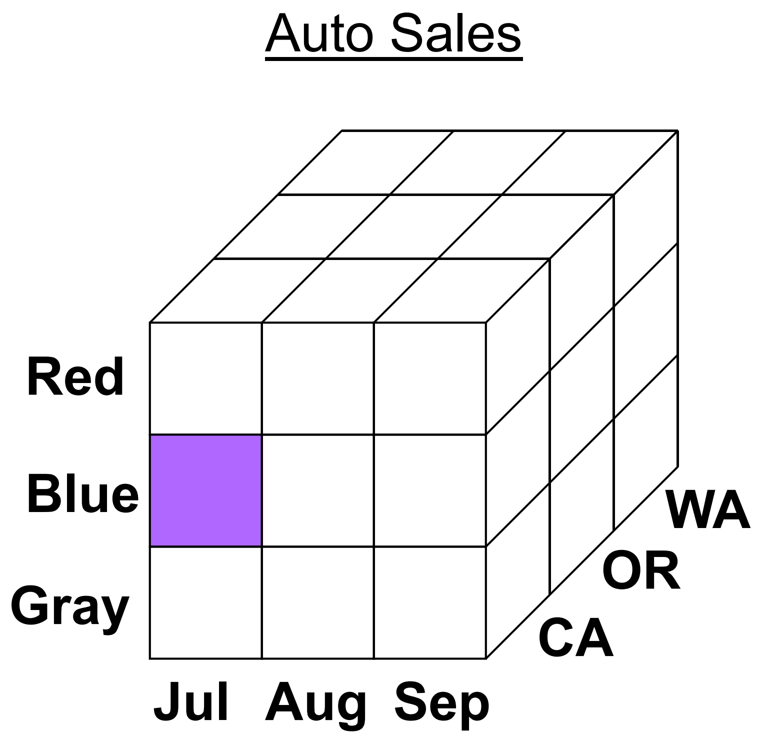
Federated Database

Warehouses vs. Federation

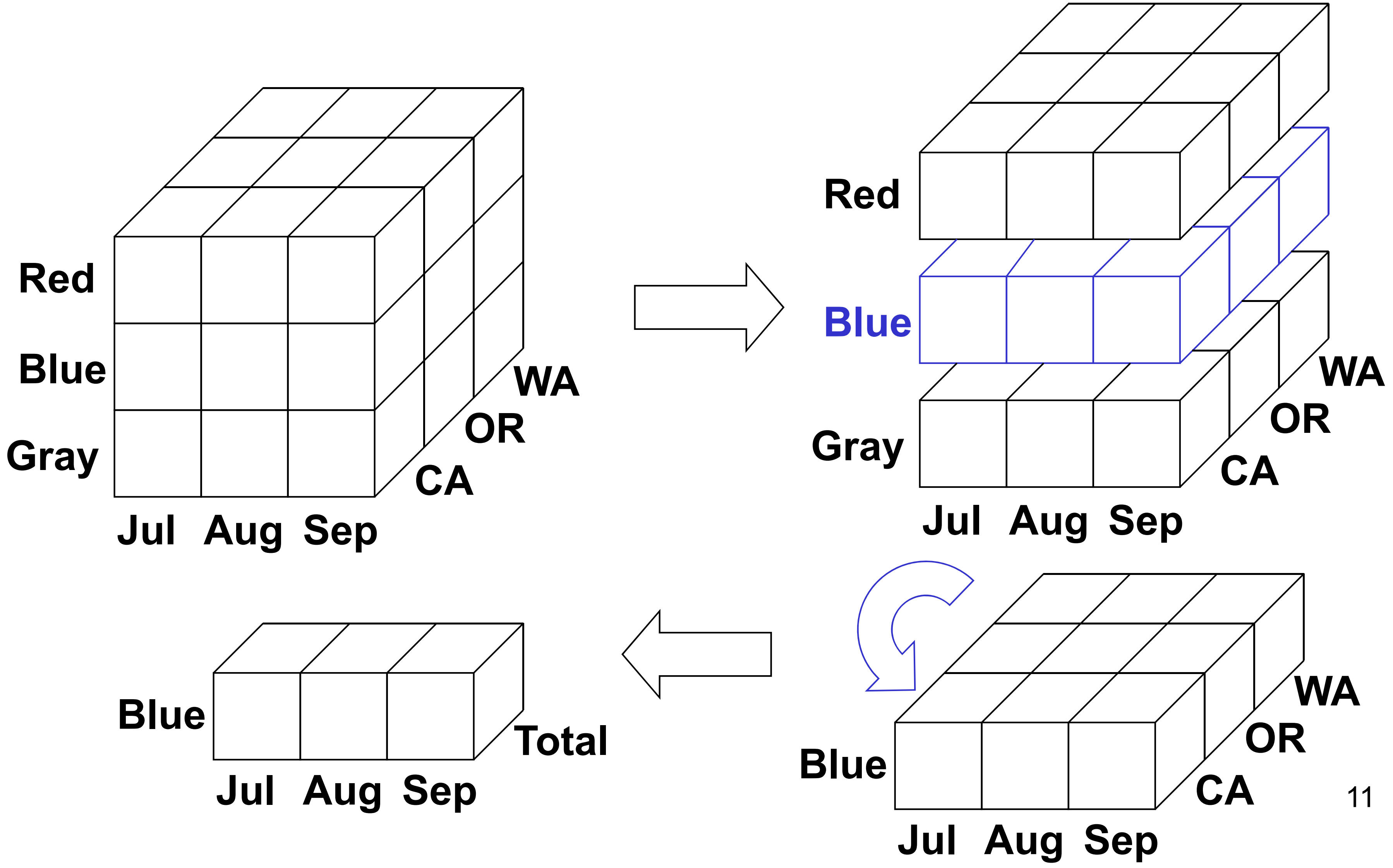
- Advantages of federated databases:
  - No redundant copying of data
  - Queries see “real-time” view of evolving data
  - More flexible security policy
- Disadvantages of federated databases:
  - Analysis queries place extra load on transactional systems
  - Query optimization is hard to do well
  - Historical data may not be available
  - Complex “wrappers” needed to mediate between analysis server and source systems
- Data warehouses are much more common in practice
  - Better performance
  - Lower complexity
  - Slightly out-of-date data is acceptable

Data Cube

- Axes of the cube represent attributes of the data records
  - Generally discrete-valued / categorical
  - e.g. color, month, state
  - Called dimensions
- Cells hold aggregated measurements
  - e.g. total $ sales, number of autos sold
  - Called facts
- Real data cubes have >> 3 dimensions



Slicing and Dicing



Sparsity

- Imagine a data warehouse for a retail store.
- Suppose dimensions are: Customer, Product, Store, Day
- If there are 100,000 customers, 10,000 products, 1,000 stores, and 1,000 days…
- …data cube has 1,000,000,000,000,000 cells!
- Fortunately, most cells are empty.
- A given store doesn’t sell every product on every day.
- A given customer has never visited most of the stores.
- A given customer has never purchased most products.
- Multi-dimensional arrays are not an efficient way to store sparse data.

Querying the Data Cube

- Cross-tabulation
  - "Cross-tab" for short
  - Report data grouped by 2 dimensions
  - Aggregate across other dimensions
  - Include subtotals
- Operations on a cross-tab
  - Roll up (further aggregation)
  - Drill down (less aggregation)

Number of Autos Sold

	CA	OR	WA	Total
Jul	45	33	30	108
Aug	50	36	42	128
Sep	38	31	40	109
Total	133	100	112	345

Roll Up and Drill Down

Number of Autos Sold

	CA	OR	WA	Total
Jul	45	33	30	108
Aug	50	36	42	128
Sep	38	31	40	109
Total	133	100	112	345

Roll up
by Month

Number of Autos Sold

CA	OR	WA	Total
133	100	112	345

Drill down
by Color

Number of Autos Sold

	CA	OR	WA	Total
Red	40	29	40	109
Blue	45	31	37	113
Gray	48	40	35	123
Total	133	100	112	345

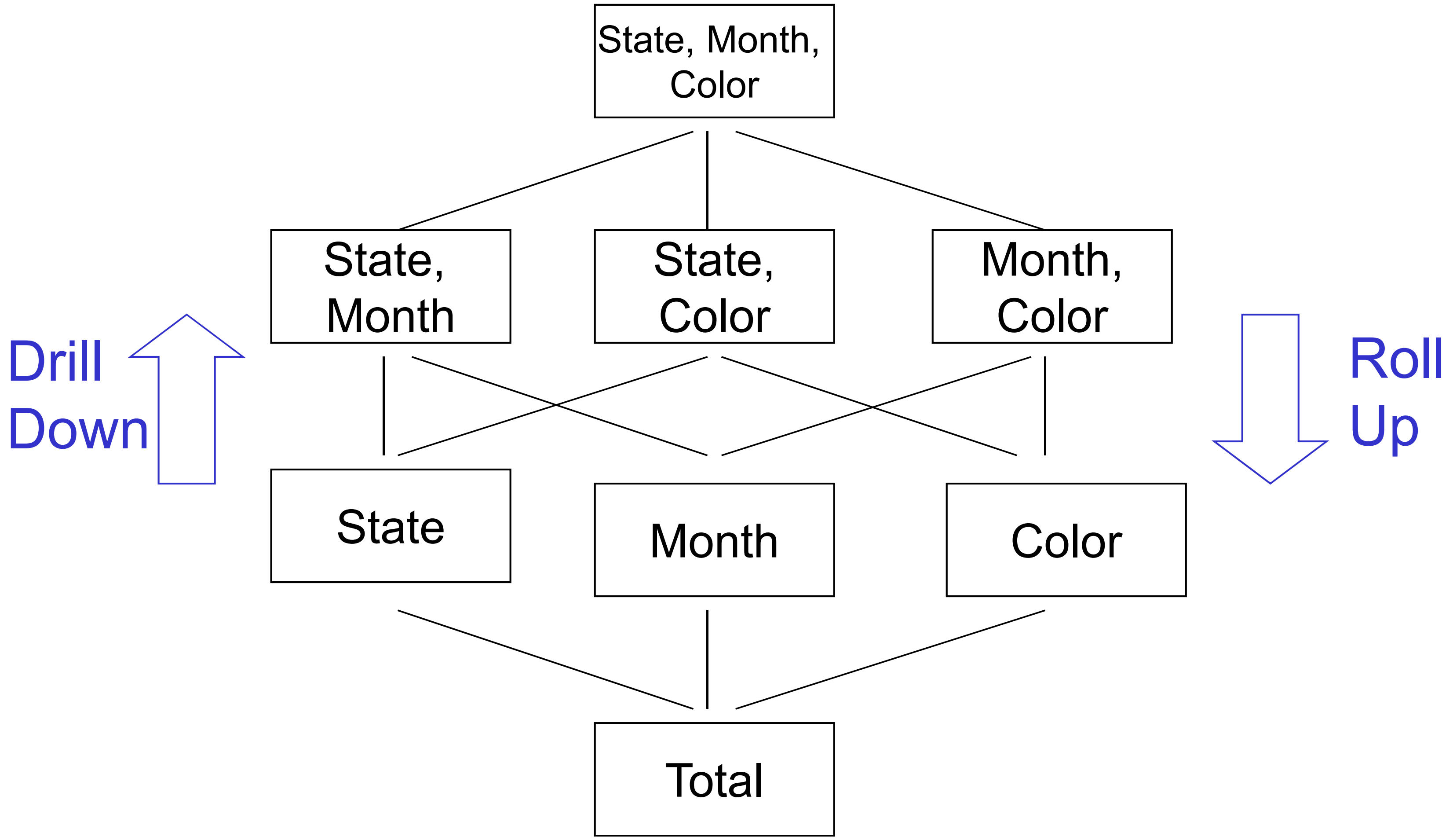
“Standard” Data Cube Query

- Measurements
  - Which fact(s) should be reported?
- Filters
  - What slice(s) of the cube should be used?
- Grouping attributes
  - How finely should the cube be diced?
  - Each dimension is either:
    - (a) A grouping attribute
    - (b) Aggregated over (“Rolled up” into a single total)
  - n dimensions $\rightarrow 2^n$ sets of grouping attributes
- Aggregation = projection to a lower-dimensional subspace

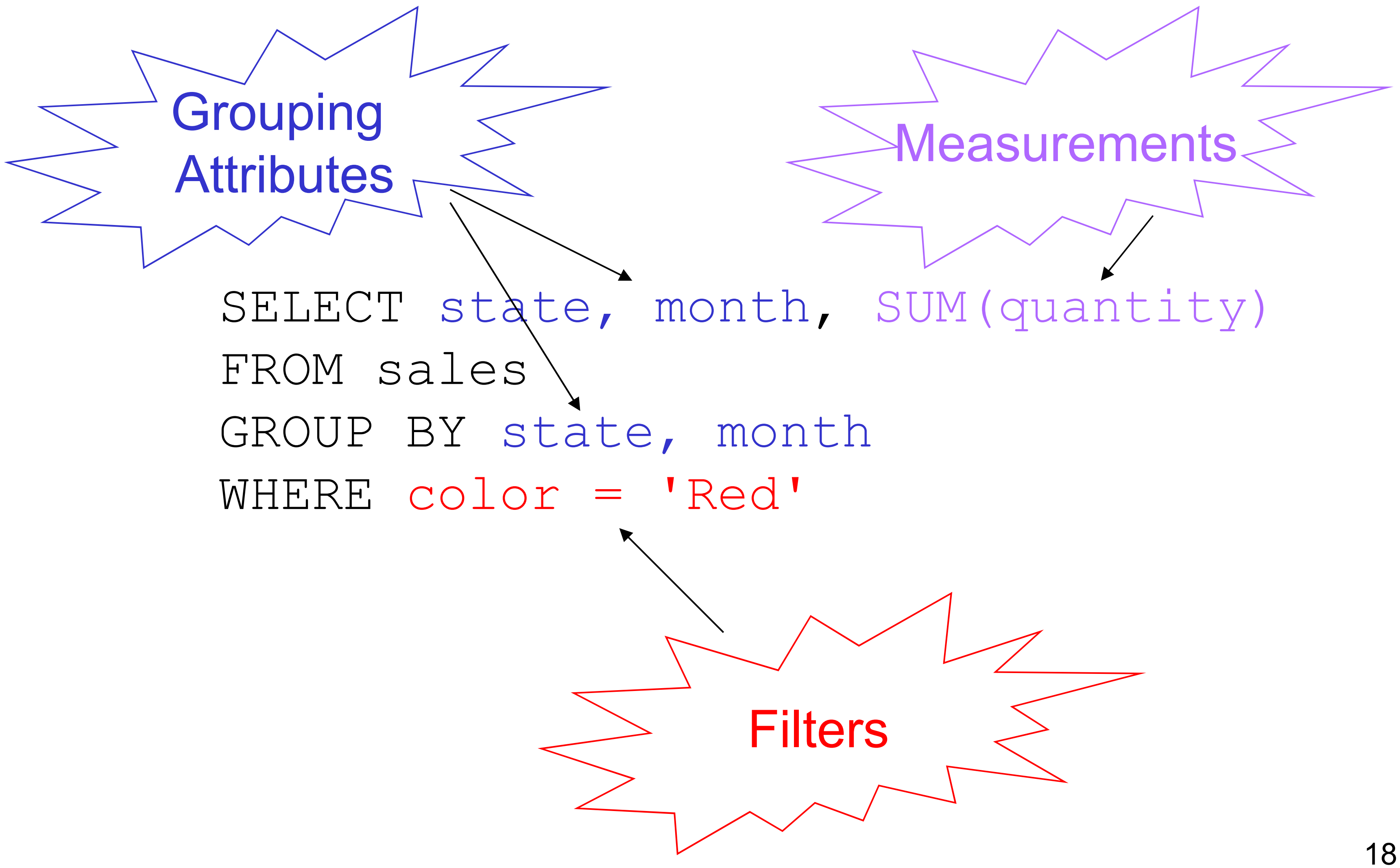
Full Data Cube with Subtotals

- Pre-computation of aggregates → fast answers to OLAP queries
  - Materialized views
- Ideally, pre-compute all 2^n types of subtotals
  - This does not handle filters!
- Otherwise, perform aggregation as needed
- Coarser-grained totals can be computed from finer-grained totals
  - But not the other way around

Data Cube Lattice



Creating a Cross-tab with SQL



```
SELECT state, month, SUM(quantity)
FROM sales
GROUP BY state, month
WHERE color = 'Red'
```

What about the totals?

- SQL aggregation query with GROUP BY does not produce subtotals, totals
- Our cross-tab report is incomplete.

```
State          Month          SUM
CA             Jul            45
CA             Aug            50
CA             Sep            38
OR             Jul            33
OR             Aug            36
OR             Sep            31
WA             Jul            30
WA             Aug            42
WA             Sep            40
```

Number of Autos Sold

	CA	OR	WA	Total
Jul	45	33	30	?
Aug	50	36	42	?
Sep	38	31	40	?
Total	?	?	?	?

SQL99 supports OLAP extensions

- OLAP extensions added to SQL 99
  - CUBE, ROLLUP

```
SELECT state, month, SUM(quantity)
FROM sales
GROUP BY CUBE(state, month)
WHERE color = 'Red'
```

Results of the CUBE query

	CA	OR	WA	Total
Jul	45	33	30	108
Aug	50	36	42	128
Sep	38	31	40	109
Total	133	100	112	345

State	Month	SUM(quantity)
CA	Jul	45
CA	Aug	50
CA	Sep	38
CA	NULL	133
OR	Jul	33
OR	Aug	36
OR	Sep	31
OR	NULL	100
WA	Jul	30
WA	Aug	42
WA	Sep	40
WA	NULL	112
NULL	Jul	108
NULL	Aug	128
NULL	Sep	109
NULL	NULL	345

Notice the use of NULL for totals

Subtotals at all levels

ROLLUP vs. CUBE

- CUBE computes entire lattice
- ROLLUP computes one path through lattice
  - Order of GROUP BY list matters
  - Groups by all prefixes of the GROUP BY list

GROUP BY ROLLUP(A,B,C)

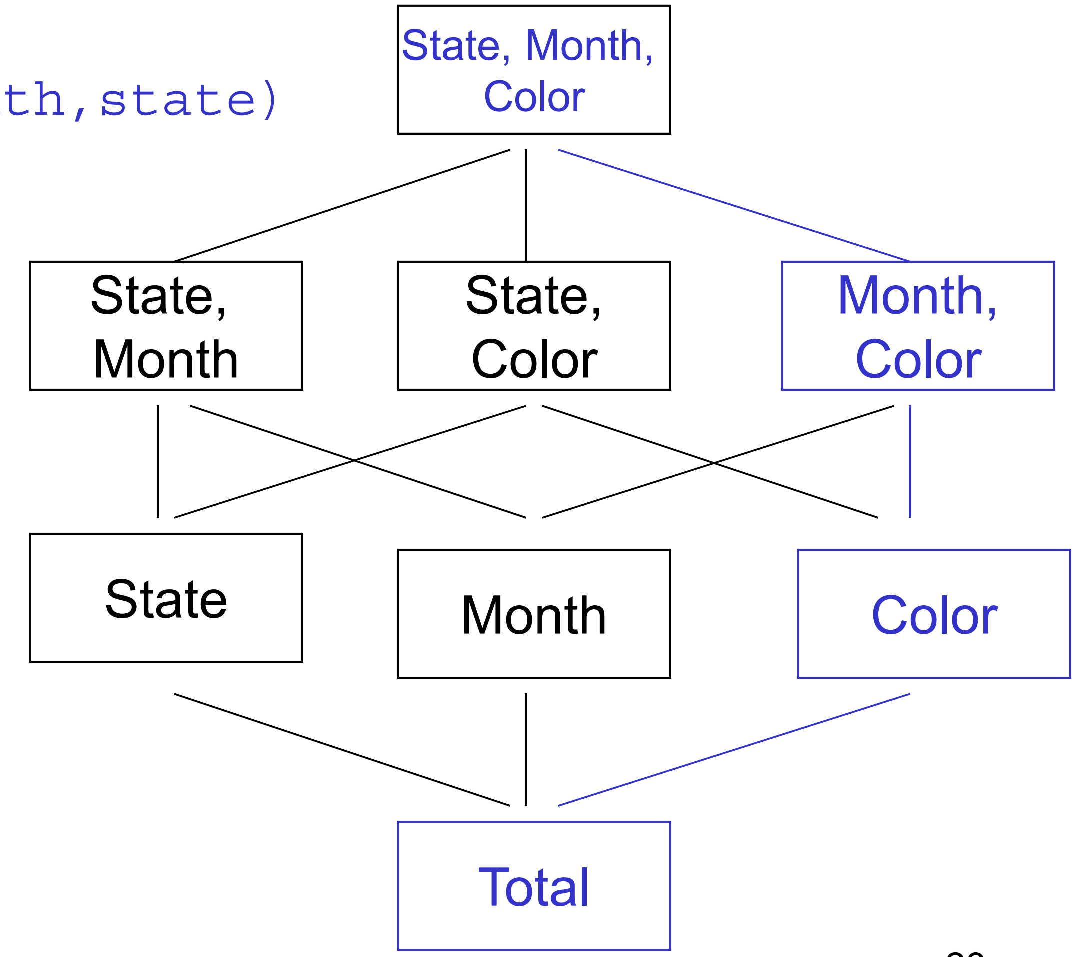
- A,B,C
- (A,B) subtotals
- (A) subtotals
- Total

GROUP BY CUBE(A,B,C)

- A,B,C
- Subtotals for the following: (A,B), (A,C), (B,C), (A), (B), (C)
- Total

ROLLUP example

```
SELECT color, month, state, SUM(quantity)
FROM sales
GROUP BY ROLLUP(color,month,state)
```



Quiz

- Suppose we have a table Facts(d1,d2,d3,x) and consider the following three 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
  ```

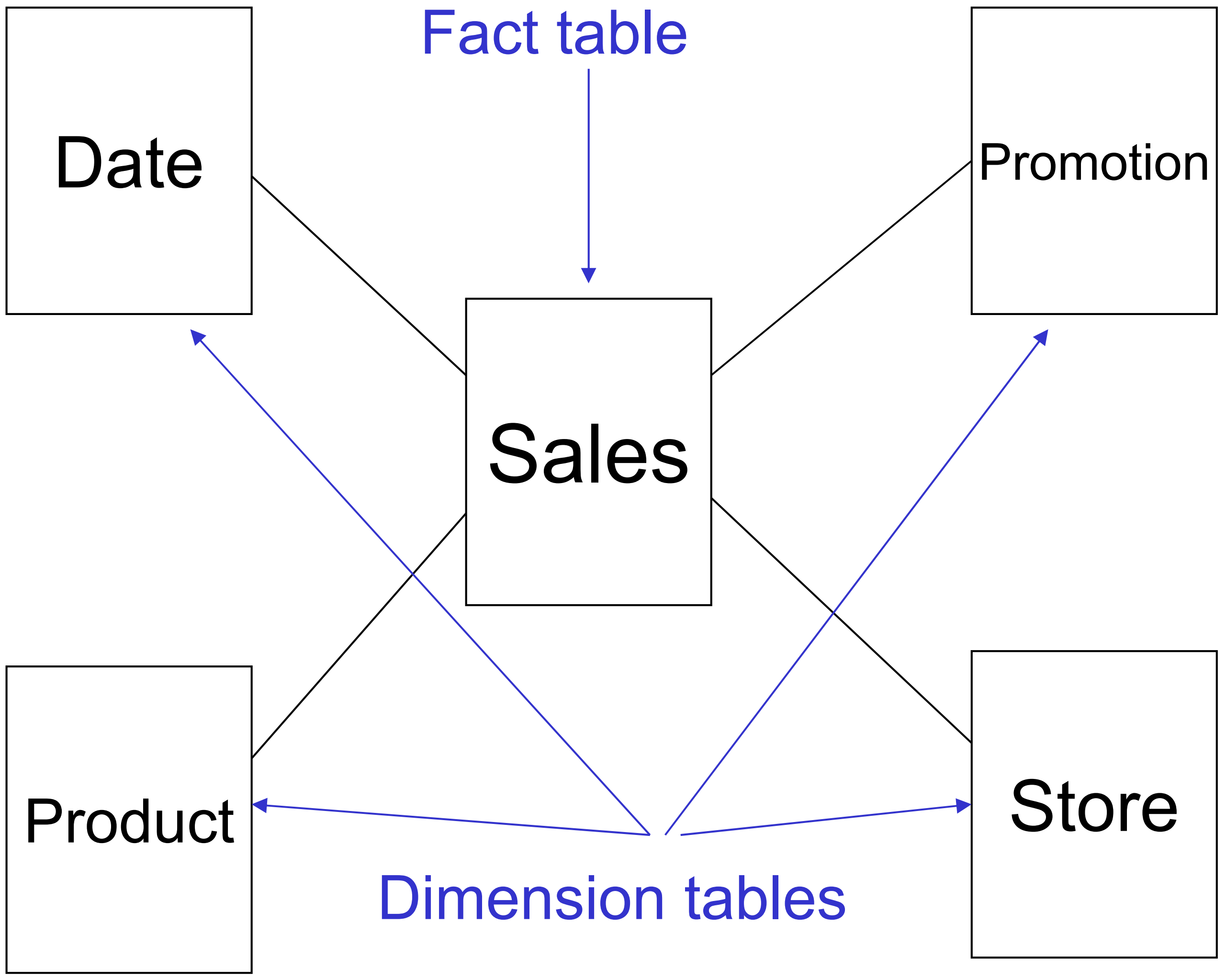
  Q3:
  ```
  SELECT d1,d2,d3,SUM(x)
  FROM Facts
  GROUP BY d1,d2,d3 WITH ROLLUP
  ```

  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 Facts. The number of tuples in the result of each of the three queries above can be specified as a formula over n1,n2,n3. A tuple (a,b,c,d,e,f) is interpreted as follows: when n1=a, n2=b, and n3=c, then the result sizes of queries Q1,Q2, and Q3 are d, e, and f respectively. What is the tuple like if n1=n2=n3=2? What about n1= 5, n2=4, n3=3?

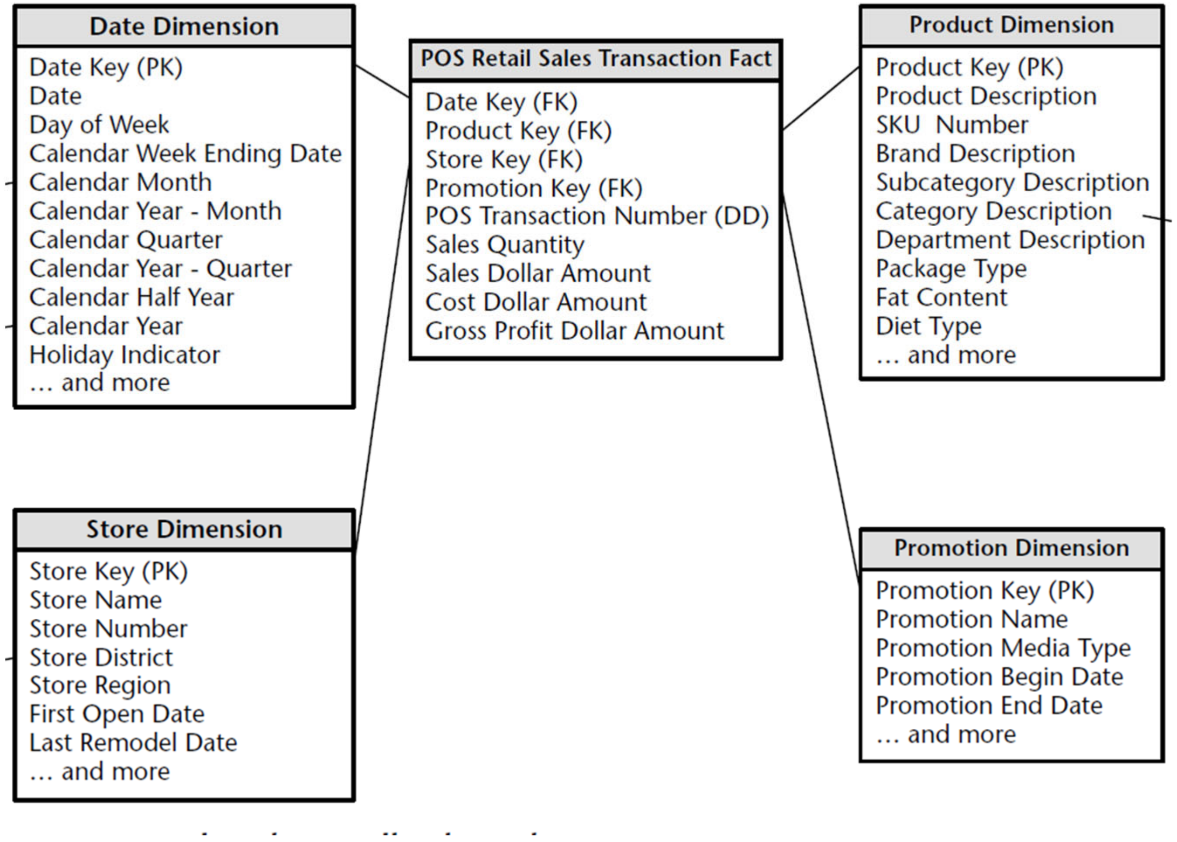
Logical vs Physical Design

- Simplicity
  - Users should understand the design
  - Data model should match users' conceptual model
  - Queries should be easy and intuitive to write
- Expressiveness
  - Include enough information to answer all important queries
  - Include all relevant data (without irrelevant data)
- Performance
  - An efficient physical design should be possible

Logical design: Star Schema



Example



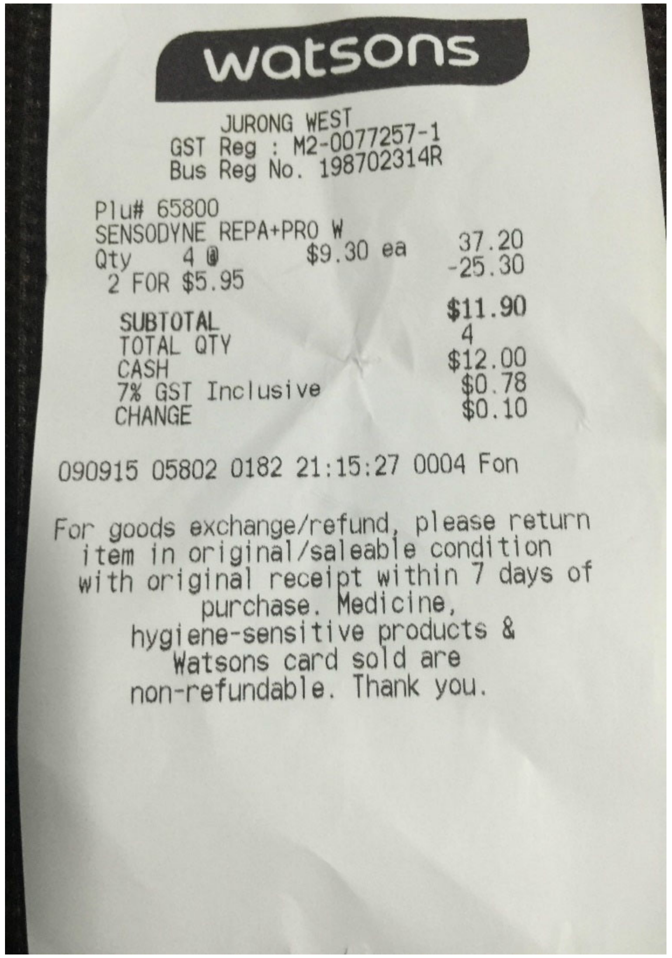
Date Dimension
Date Key (PK)
Date
Day of Week
Calendar Week Ending Date
Calendar Month
Calendar Year - Month
Calendar Quarter
Calendar Year - Quarter
Calendar Half Year
Calendar Year
Holiday Indicator
… and more

POS Retail Sales Transaction Fact
Date Key (FK)
Product Key (FK)
Store Key (FK)
Promotion Key (FK)
POS Transaction Number (DD)
Sales Quantity
Sales Dollar Amount
Cost Dollar Amount
Gross Profit Dollar Amount

Product Dimension
Product Key (PK)
Product Description
SKU Number
Brand Description
Subcategory Description
Category Description
Department Description
Package Type
Fat Content
Diet Type
… and more

Store Dimension
Store Key (PK)
Store Name
Store Number
Store District
Store Region
First Open Date
Last Remodel Date
… and more

Promotion Dimension
Promotion Key (PK)
Promotion Name
Promotion Media Type
Promotion Begin Date
Promotion End Date
… and more



```
watsons
        JURONG WEST
   GST Reg : M2-0077257-1
   Bus Reg No. 198702314R

Plu# 65800
SENSODYNE REPA+PRO W
Qty     4 @       $9.30 ea     37.20
 2 FOR $5.95                  -25.30
  SUBTOTAL                    $11.90
  TOTAL QTY                        4
  CASH                        $12.00
  7% GST Inclusive             $0.78
  CHANGE                       $0.10

090915 05802 0182 21:15:27 0004 Fon

For goods exchange/refund, please return
 item in original/saleable condition
 with original receipt within 7 days of
       purchase. Medicine,
   hygiene-sensitive products &
       Watsons card sold are
    non-refundable. Thank you.
```

Fact Tables

- Each fact table contains measurements about a process of interest.
- Each fact row contains two things:
  - Numerical measure columns
  - Foreign keys to dimension tables
  - That’s all!
- Properties of fact tables:
  - Very big
    - Often millions or billions of rows
  - Narrow
    - Small number of columns
  - Changes often
    - New events in the world → new rows in the fact table
    - Typically append-only
- Uses of fact tables:
  - Measurements are aggregated fact columns.

Dimension Tables

- Each one corresponds to a real-world object or concept.
  - Examples: Customer, Product, Date, Employee, Region, Store, Promotion, Vendor, Partner, Account, Department
- Properties of dimension tables:
  - Contain many descriptive columns
    - Dimension tables are wide (dozens of columns)
  - Generally don’t have too many rows
    - At least in comparison to the fact tables
    - Usually < 1 million rows
  - Contents are relatively static
    - Almost like a lookup table
- Uses of dimension tables
  - Filters are based on dimension attributes
  - Grouping columns are dimension attributes
  - Fact tables are referenced through dimensions

Surrogate Keys

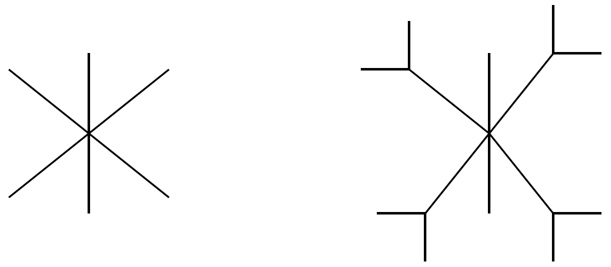
- Primary keys of dimension tables should be surrogate keys, not natural keys
  - Natural key: A key that is meaningful to users
  - Surrogate key: A meaningless integer key that is assigned by the data warehouse
  - Keys or codes generated by operational systems = natural keys (avoid using these as keys!)
    - E.g. Account number, UPC code, Social Security Number
  - Syntactic vs. semantic

Benefits of Surrogate Keys

- Data warehouse insulated from changes to operational systems
- Easy to integrate data from multiple systems
  - What if there’s a merger or acquisition?
- Narrow dimension keys → Thinner fact table → Better performance
  - This can actually make a big performance difference.
- Better handling of exceptional cases
  - For example, what if the value is unknown or TBD?
  - Using NULL is a poor option
    - Three-valued logic is not intuitive to users
    - They will get their queries wrong
    - Join performance will suffer
  - Better: Explicit dimension rows for “Unknown”, “TBD”, “N/A”, etc.
- Avoids tempting query writers to assume implicit semantics
  - Example: WHERE date_key < '01/01/2004'
  - Will facts with unknown date be included?

Snowflake Schema

- Dimension tables are not in normal form
  - Redundant information about hierarchies
- Normalizing dimension tables leads to snowflake schema
  - Avoid redundancy → some storage savings



- Snowflaking not recommended in most cases
  - More tables = more complex design
  - More tables → more joins → slower queries
  - Space consumed by dimensions is small compared to facts
  - Exception: Really big dimension tables
    - In some warehouses, customer dimension is really large
    - We'll return to this next week.

Slowly Changing Dimensions

- Compared to fact tables, contents of dimension tables are relatively stable.
  - New sales transactions occur constantly.
  - New products are introduced rarely.
  - New stores are opened very rarely.
- Attribute values for existing dimension rows do occasionally change over time
  - Customer moves to a new address
  - Grouping of stores into districts, regions changes due to corporate re-org
- How to handle gradual changes to dimensions?
  - Option 1: Overwrite history
  - Option 2: Preserve history

Overwriting History

- Simplest option: update the dimension table
  - "Type 1" slowly changing dimension
- Example:
  - Product size incorrectly recorded as "8 oz" instead of "18 oz" due to clerical error
  - Error is detected and fixed in operational system
  - Error should also be corrected in data warehouse
    - Update row in dimension table
    - Update pre-computed aggregates
- Updating dimension table rewrites history
  - Brian lived in WI in 1993
  - Later, Brian moved to CA
  - Suppose we update the customer dimension table
  - Query: "Total sales to WI customers in 1993?"
  - Sales to Brian are incorrectly omitted from the query answer

Preserving History

- Accurate historical reporting is usually important in a data warehouse
- How can we capture changes while preserving history?
- Answer: Create a new dimension row
  - Old fact table rows point to the old row
  - New fact table rows point to the new row
  - “Type 2” slowly changing dimension

Customer Dimension

Cust_key	Name	Sex	State	YOB
457	Brian	Male	WI	1976
…	…	…	…	…
784	Brian	Male	CA	1976

Old dimension row

New dimension row

Slowly Changing Dim. Example

Customer Dimension

Cust_key	**Name**	**Sex**	**State**	**YOB**
457	Brian	Male	WI	1976
…	…	…	…	…
784	Brian	Male	CA	1976

Sales Fact

Cust_key	**…**	**Quantity**
…	…	…
457	…	5
…	…	…
784	…	4

Existing fact rows use old dimension row $\Rightarrow$

New fact rows use new dimension row $\Rightarrow$

Pros and Cons

- Type 1: Overwrite existing value
  - + Simple to implement
- Type 2: Add a new dimension row
  - + Accurate historical reporting
  - + Pre-computed aggregates unaffected
  - - Dimension table grows over time
- Type 2 SCD requires surrogate keys
  - – Store mapping from operational key to most current surrogate key in data staging area
- To report on Brian’s activity over time, constrain on natural key
  - – WHERE name = ‘Brian’

Choosing Type 1 vs. Type 2

- Both choices are commonly used
- Easy to “mix and match”
  - Preserve history for some attributes
  - Overwrite history for other attributes
- Questions to ask:
  - Will queries want to use the original attribute value or the new attribute value?
    - In most cases preserving history is desirable
  - Does the performance impact of additional dimension rows outweigh the benefit of preserving history?
    - Some fields like “customer phone number” are not very useful for reporting
    - Adding extra rows to preserve phone number history may not be worth it

Hybrid SCD Solutions

- Suppose we want to be able to report using either old or new values
  - Mostly useful for corporate reorganizations!
  - Example: Sales districts are re-drawn annually
- Solution: Create two dimension columns
- Approach #1: “Previous District” and “Current District”
  - Allows reporting using either the old or the new scheme
  - Whenever district assignments change, all “Current District” values are moved to “Previous District”
  - “Type 3” Slowly Changing Dimension
- Approach #2: “Historical District” and “Current District”
  - Allows reports with the original scheme or the current scheme
  - When district assignment changes, do two things:
    - Create a new dimension row with “Historical District” = new value
    - Overwrite relevant dim. rows to set “Current District” = new value

The Customer Dimension

- Customer dimensions can be very wide
  - Often dozens or even hundreds of attributes
  - Contact information (name, address, phone, e-mail)
  - Demographics (age, ethnicity, gender, education, profession, income, household size, etc.)
  - Psychographics (interests, values, beliefs, attitudes)
  - Dates (birthday, first purchase, last purchase, online reg. date)
  - Behavioral scores (RFM, churn propensity, etc.)
- Data available from many sources
  - Information provided directly by customers
  - Prospect lists acquired from partners or vendors
  - Syndicated data
    - Market research
    - Customs data
  - Data derived from warehouse analysis

Very Large Dimensions

- Customer dimensions can be very wide
  - Dozens or hundreds of attributes
- Customer dimensions can be very large
  - Tens of millions of rows in some warehouses
  - Sometimes includes prospects as well as actual customers
- Size can lead to performance challenges
  - One case when performance concerns can trump simplicity
  - Can we reduce width of dimension table?
  - Can we reduce number of rows caused by preserving history for slowly changing dimension?

Outrigger Tables

- Limited normalization of large dimension table to save space
- Identify attribute sets with these properties:
  - Highly correlated
  - Low in cardinality (compared to # of customers)
  - Change in unison
- Example:
  - External data provider computes demographic data for each county
  - 100 demographic attributes are provided
  - Updates are supplied every six months
- Follow these steps for each attribute set:
  - Create a separate "outrigger dimension" for each attribute set
  - Remove the attributes from the customer dimension
  - Replace with a foreign key to the outrigger table
  - No foreign key from fact row to outrigger
    - Outrigger attributes indirectly associated with facts via customer dim.

Outrigger Example

Customer Dimension

Cust_id	FName	LName	Zip	Demo_id
1	Brian	Babcock	94403	34
2	Rajeev	Motwani	94303	12
3	Leland	Stanford	94305	12

County Demographics Outrigger

Demo_id	County	AvgInc	HHold Size	…
12	Santa Clara	78000	2.3	…
…	…	…	…	…
34	San Mateo	67000	2.5	…

Outrigger Tables

- Advantages:
  - Space savings
    - Customer dimension table becomes narrower
    - Outrigger table has relatively few rows
    - One copy per county vs. one copy per customer
- Disadvantages:
  - Additional tables introduced
    - Accessing outrigger attributes requires an extra join
    - Users must remember which attributes are in outrigger vs. main customer dimension
      - Creating a view can solve this problem

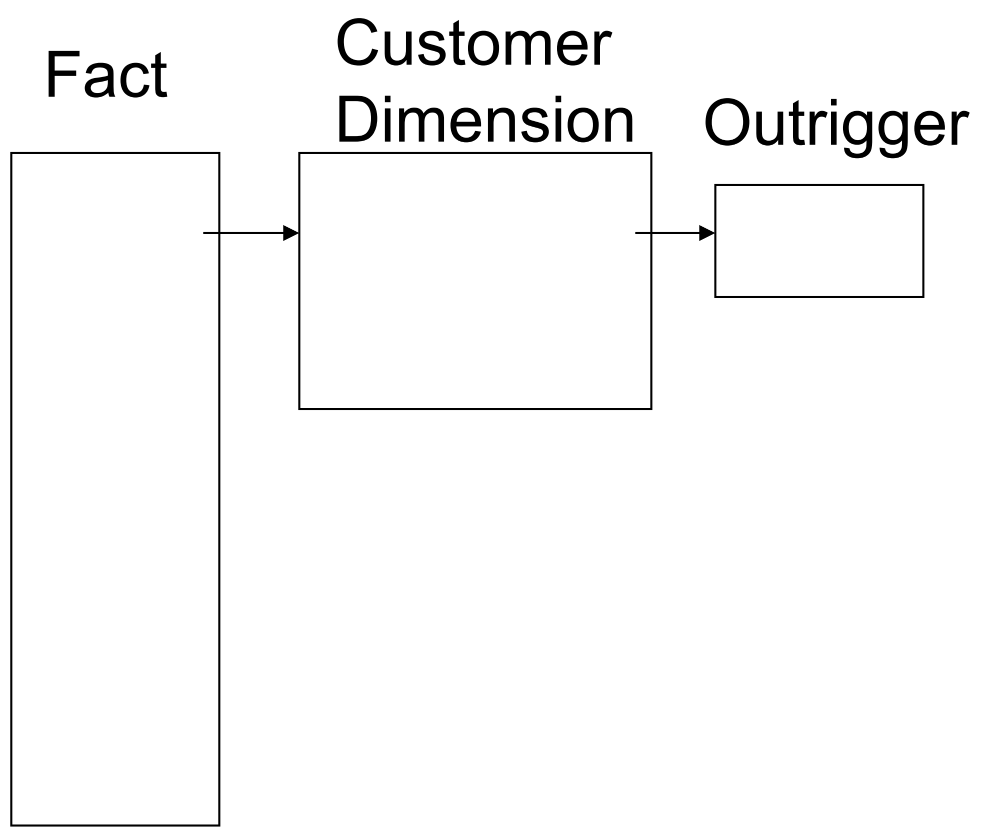
Mini-Dimensions

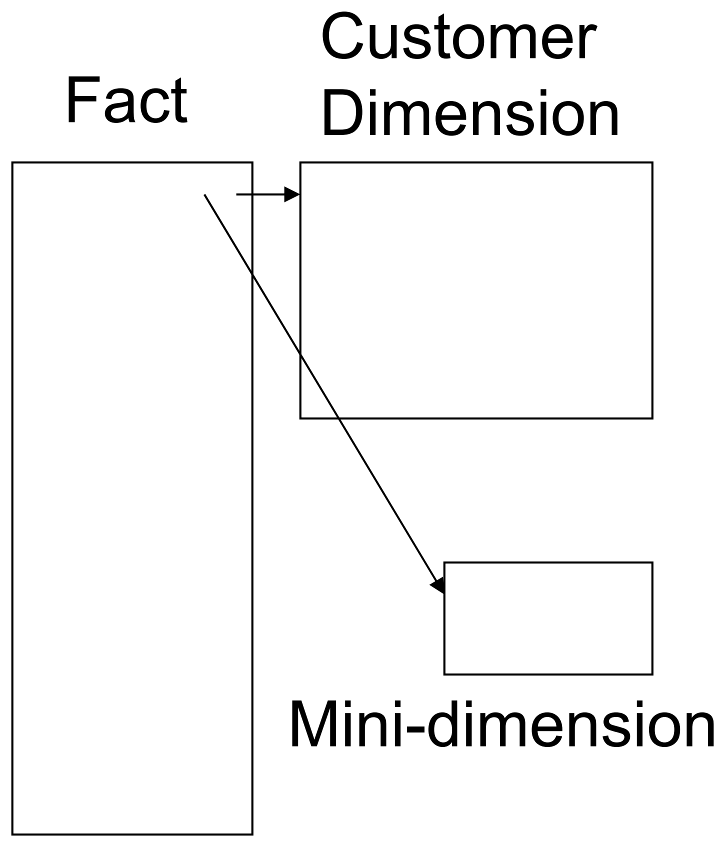
- Some attributes change relatively frequently
  - Behavior-based scores
  - Certain demographic attributes
    - Age, Income, Marital Status, # of children
- How to preserve history without row explosion?
- Some attributes are queried relatively frequently
  - Queries using huge customer dimension are slowed
- How to improve query performance?
- Create a mini-dimension:
  - Remove frequently-changing or frequently-queried attributes from the customer dimension
  - Add them to a separate mini-dimension table instead
  - Discretize mini-dimension attributes to reduce cardinality
    - Group continuously-valued attributes into buckets or bands
    - Example: Age < 20, Age 20-29, Age 30-39, Age 40-49, Age 50+
  - Include foreign keys to both customer dimension & mini-dimension in fact table

Mini-Dimensions

- Advantages:
  - History preserved without space blow-up
    - Fact table captures historical record of attribute values
    - Mini-dimension has small number of rows
      - # of unique combinations of mini-dimension attributes is small
      - Consequence of discretization
      - Limit number of attributes in a single mini-dimension!
  - Improved performance for queries that use mini-dimension
    - At least for those queries that can avoid the main customer dimension
- Disadvantages:
  - Fact table width increases
    - Due to increased number of dimension foreign keys
  - Information lost due to discretization
    - Less detail is available
    - Impractical to change bucket / band boundaries
  - Additional tables introduced
    - Users must remember which attributes are in mini-dimension vs. main customer dimension

Outrigger vs. Mini-Dimension





Outrigger vs. Mini-Dimension

- Mini-dimension approach
  - Explicit link between fact table and mini-dimension
  - No explicit link between customer dimension and mini-dimension
  - Difficult to express queries that group past customer behavior based on current demographic values
    - Implicit association via fact table
- Outrigger approach
  - No explicit link between fact table and outrigger
  - Explicit link between customer dimension and outrigger
  - Associating facts with outrigger requires join through customer dimension
  - Preserving history for rapidly-changing attributes leads to customer dimension blow-up
- Hybrid approach
  - Separate some attributes into their own mini-dimension
  - Add foreign key to mini-dimension to both fact table and customer dimension
  - Customer dimension foreign key updated using “overwrite history” semantics
  - Queries based on historically accurate attribute values use fact table foreign key
  - Queries based on latest attribute values use customer dimension foreign key
  - Greater expressive power, and greater risk of confusing users!

More Outriggers / Mini-Dims

- Lots of information about some customers, little info about others
  - A common scenario
- Example: web site browsing behavior
  - Web User dimension (= Customer dimension)
  - Unregistered users
    - User identity tracked over time via cookies
    - Limited information available
      - First active date, Latest active date, Behavioral attributes
      - Possibly ZIP code through IP lookup
  - Registered users
    - Lots of data provided by user during registration
  - Many more unregistered users than registered users
  - Most attribute values are unknown for unregistered users
- Split registered user attributes into a separate table
  - Either an outrigger or a mini-dimension
  - For unregistered users, point to special “Unregistered” row

Summary

- Two database activities – OLTP and OLAP – are very different, and have different demands on performance
- Data warehouse has been developed to deal with OLAP
- Dimension modeling offers simplicity and ease of use, and improved performance