CPSC 304 – March 20, 2018 Administrative Notes

- Reminder: Project
  - Main project deliverables due March 29
  - Demos the following week (sign ups to come)
- Reminder: Final exam Tuesday, April 17
  - Section 201 A-LEU: Swing 122
  - Section 201 LI-Z: Swing 121
  - Section 202 A-LY: Swing 222
  - Section 202 MAG-Z: Swing 221

Now where were we...

- We'd almost finished SQL, but first we were discussing some special joins that you can put in the FROM clause…

Natural Join

- The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.
- Natural Join : Guidelines
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don’t use ON clause in a natural join.

```
SELECT *
FROM student s natural join enrolled e
```

- Natural join of tables with no pairs of identically named columns will return the cross product of the two tables.

```
SELECT *
FROM student s natural join class c
```

More fun with joins

- What happens if I execute query:

```
SELECT *
FROM student s, enrolled e
WHERE s.snum = e.snum
```

- To get *all* students, you need an *outer join*
- There are several special joins declared in the *FROM* clause:
  - Inner join – default: only include matches
  - Left outer join – include all tuples from left hand relation
  - Right outer join – include all tuples from right hand relation
  - Full outer join – include all tuples from both relations
- Orthogonal: can have natural join (as in relational algebra)

Example: SELECT *

FROM Student S NATURAL LEFT OUTER JOIN Enrolled E

More fun with joins examples

R

A	B
1	2
3	3

S

B	C
2	4
4	6

Natural Inner Join

A	B	C
1	2	4

Natural Left outer Join

A	B	C
1	2	4
3	3	Null

Natural Right outer Join

A	B	C
1	2	4
Null	4	6

Natural outer Join

A	B	C
1	2	4
3	3	Null
Null	4	6

Outer join (without the Natural) will use the key word on for specifying The condition of the join.

Outer join not implemented in MYSQL
Outer join is implemented in Oracle

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

R(A,B) S(B,C,D)

A	B	B	C	D
1	2	2	4	6
3	4	4	6	8
5	6	4	7	9

Start clickerouter.sql

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

A is correct

R(A,B) S(B,C,D)

A	B
1	2
3	4
5	6

B	C	D
2	4	6
4	6	8
4	7	9

A	B	B	C	D
3	4	2	4	6
5	6	4	6	8
1	2	NULL	NULL	NULL
NULL	NULL	4	7	9

Database Manipulation Insertion redux

- Can insert a single tuple using:

  INSERT INTO Student
  VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)

- or

  INSERT INTO Student (sid, name, address, phone, major)
  VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)

- Add a tuple to student with null address and phone:

  INSERT INTO Student (sid, name, address, phone, major)
  VALUES (33388, 'Chan', null, null, CPSC)

Database Manipulation
Insertion redux (cont)

- Can add values selected from another table
- Enroll student 51135593 into every class taught by faculty 90873519

```
INSERT  INTO  Enrolled
SELECT  51135593, name
FROM Class
WHERE fid = 90873519
```

The SELECT-FROM-WHERE statement is fully evaluated before any of its results are inserted or deleted.

Database Manipulation
Deletion

- Note that only whole tuples are deleted.
- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE   FROM  Student
WHERE   name = 'Smith'
```

Database Manipulation Updates

- Increase the age of all students by 2 (should not be more than 100)
- Need to write two updates:

```
UPDATE  Student
SET     age = 100
WHERE   age >= 98

UPDATE  Student
SET age = age + 2
WHERE age < 98
```

- Is the order important?

Integrity Constraints (Review)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*:
  - domain constraints,
  - primary key constraints,
  - foreign key constraints,
  - general constraints

General Constraints: Check

- We can specify constraints over a single table using table constraints, which have the form

Check conditional-expression

```
CREATE TABLE  Student
   ( snum  INTEGER,
     sname  CHAR(32),
     major  CHAR(32),
     standing CHAR(2)
     age  REAL,
     PRIMARY KEY  (snum),
     CHECK  ( age >= 10
              AND age < 100 );
```

Check constraints are checked when tuples are inserted or modified

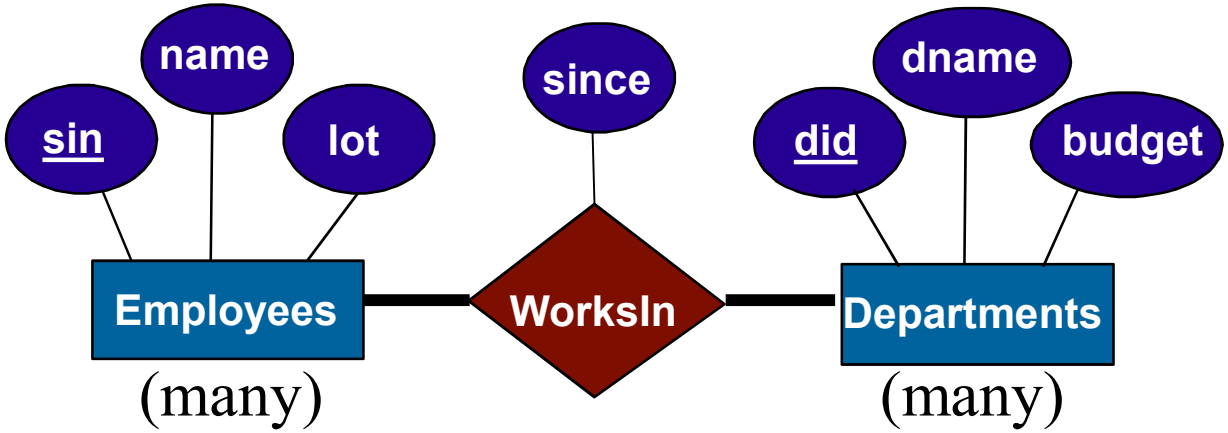
General Constraints: Check

- Constraints can be named
- Can use subqueries to express constraint
- Table constraints are associated with a single table, although the conditional expression in the check clause can refer to other tables

```
CREATE TABLE Enrolled
    ( snum    INTEGER,
      cname   CHAR(32),
      PRIMARY KEY (snum, cname),
      CONSTRAINT noR15
      CHECK (`R15' <>
                ( SELECT c.room
                  FROM    class c
                  WHERE c.name=cname)));
```

No one can be enrolled in a class, which is held in R15

Constraints over Multiple Relations: Remember this one?



- We couldn't express "every employee works in a department and every department has some employee in it"?
- Neither foreign-key nor not-null constraints in Works_In can do that.
- Assertions to the rescue!

Constraints Over Multiple Relations

- Cannot be defined in one table.
- Are defined as ASSERTIONs which are not associated with any table
- Example: *Every MovieStar needs to star in at least one Movie*

```
CREATE ASSERTION  totalEmployment
CHECK
( NOT EXISTS ((SELECT StarID FROM MovieStar)
                    EXCEPT
                   (StarID FROM StarsIn)));
```

Constraints Over Multiple Relations

- Example: Write an assertion to enforce every student to be registered in at least one course.

```
CREATE ASSERTION  Checkregistry
CHECK
( NOT EXISTS ((SELECT snum FROM student)
                        EXCEPT
                       (SELECT snum FROM enrolled)));
```

Triggers

- Trigger : a procedure that starts automatically if specified changes occur to the DBMS
- Active Database: a database with triggers
- A trigger has three parts:
  1. Event (activates the trigger)
  2. Condition (tests whether the trigger should run)
  3. Action (procedure executed when trigger runs)

Useful for project
Not tested on exams

- Database vendors did not wait for trigger standards! So trigger format depends on the DBMS
- NOTE: triggers may cause cascading effects.

Good way to shoot yourself in the foot

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngStudentUpdate
   AFTER INSERT ON Student
REFERENCING NEW TABLE NewStudent
FOR EACH STATEMENT
   INSERT INTO
      YoungStudent(snum, sname, major, standing, age)
      SELECT  snum, sname, major, standing, age
      FROM    NewStudent N
      WHERE  N.age <= 18;
```

event

newly inserted tuples

apply once per statement

action

Can be either before or after

That's nice. But how do we code with SQL?

- Direct SQL is rarely used: usually, SQL is embedded in some application code.
- We need some method to reference SQL statements.
- But: there is an *impedance mismatch* problem.
  - Structures in databases <> structures in programming languages
- Many things can be explained with the impedance mismatch.

The Impedance Mismatch Problem

The host language manipulates variables, values, pointers SQL manipulates relations.

There is no construct in the host language for manipulating relations. See https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

Why not use only one language?

- Forgetting SQL: “we can quickly dispense with this idea” [Ullman & Widom, pg. 363].
- SQL cannot do everything that the host language can do.

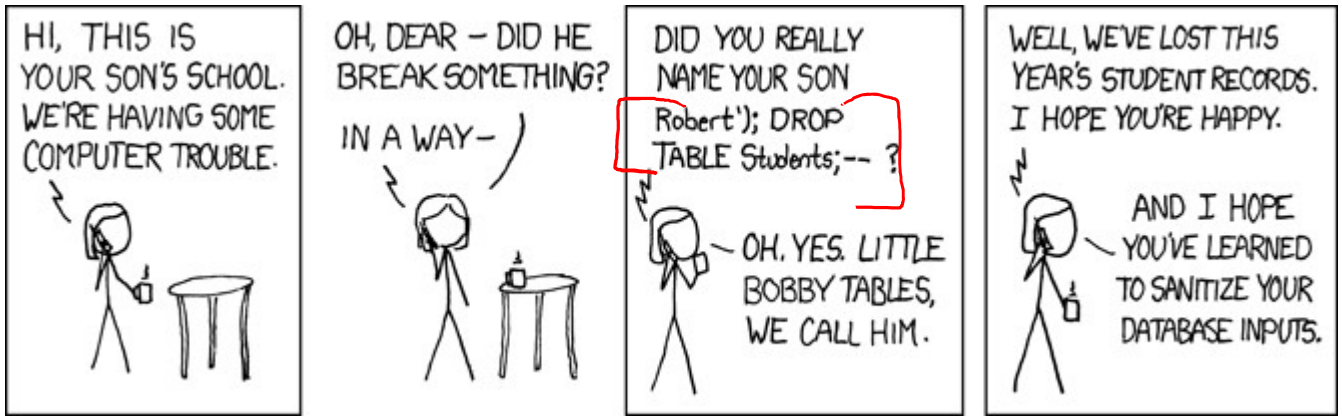
Database APIs

Rather than modify compiler, add library with database calls (API)

- Special standardized interface: procedures/objects
- Passes SQL strings from language, presents result sets in a language-friendly way – solves that impedance mismatch
- Microsoft's *ODBC* is a C/C++ standard on Windows
- Sun's *JDBC* a Java equivalent
- API's are DBMS-neutral
  - a "driver" traps the calls and translates them into DBMS-specific code

And now a brief digression

- Have you ever wondered why some websites don't allow special characters?



Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Consists of a data definition, data manipulation and query language.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.

Summary (Cont')

- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints (and triggers)
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time
- APIs such as ODBC and JDBC introduce a layer of abstraction between application and DBMS

Learning Goals Revisited

- Given the schemas of a relation, create SQL queries using: SELECT, FROM, WHERE, EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE, ANY, ALL, DISTINCT, GROUP BY and HAVING.
- Show that there are alternative ways of coding SQL queries to yield the same result. Determine whether or not two SQL queries are equivalent.
- Given a SQL query and table schemas and instances, compute the query result.
- Translate a query between SQL and RA.
- Comment on the relative expressive power of SQL and RA.
- Explain the purpose of NULL values and justify their use. Also describe the difficulties added by having nulls.
- Create and modify table schemas and views in SQL.
- Explain the role and advantages of embedding SQL in application programs.
- Write SQL for a small-to-medium sized programming application that requires database access.
- Identify the pros and cons of using general table constraints (e.g., CONSTRAINT, CHECK) and triggers in databases.

Data Warehousing & OLAP

Text:

Sections 25.1—25.5; 25.7-25.10.

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 3 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 differences between a star schema design and a snowflake design for a data warehouse, including potential tradeoffs in performance.
- Argue for the value of a data cube in terms of:
  - The type of data in the cube (numeric, categorical, temporal, counts, sums)
  - 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.

What We Have Focused on So Far

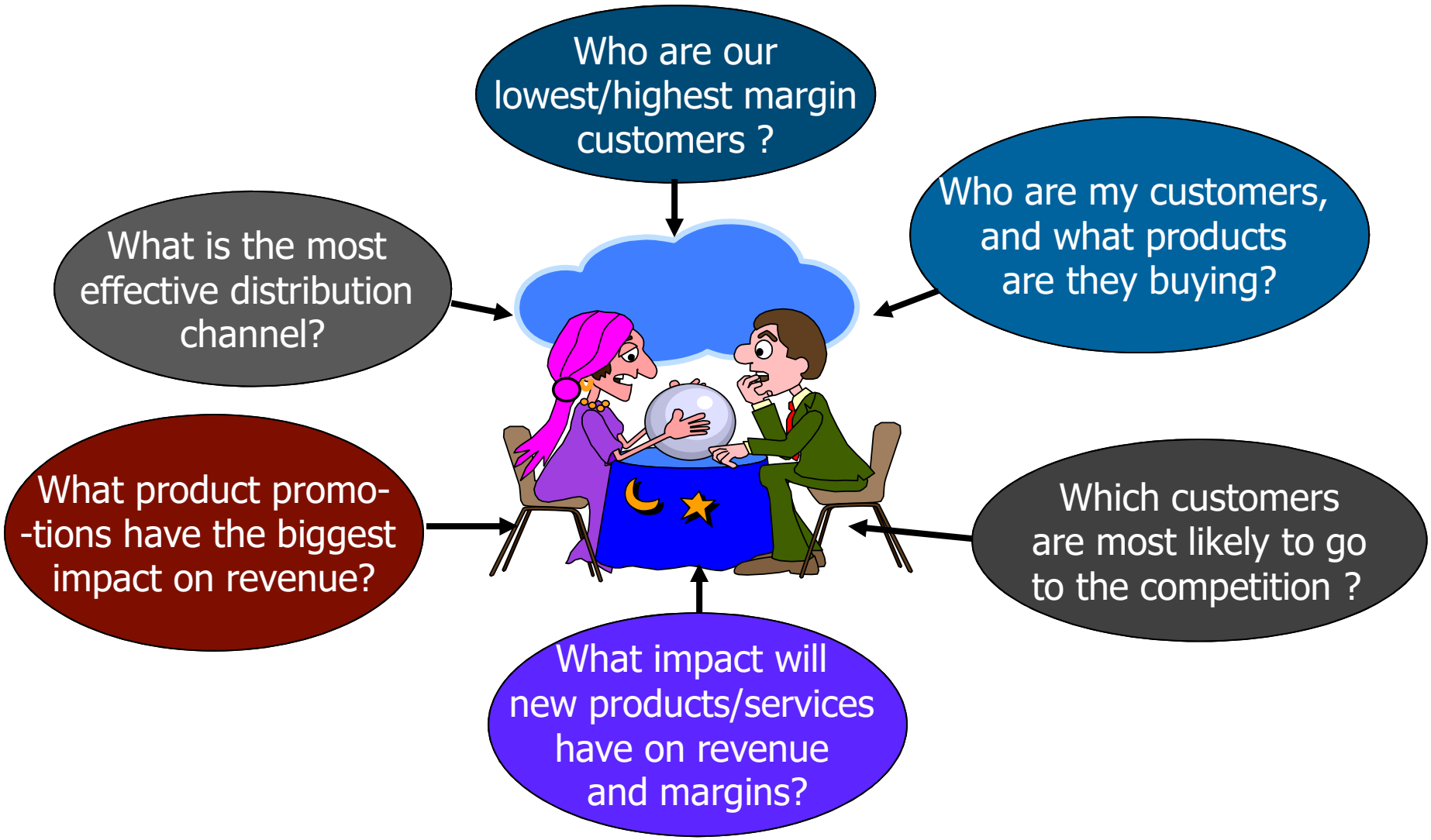
- **OLTP (On-Line Transaction Processing)**
  - Transaction-oriented applications, typically for data entry and retrieval transaction processing.
  - The system responds immediately to user requests.
  - High throughput and insert- or update-intensive database management. These applications are used 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 business 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.

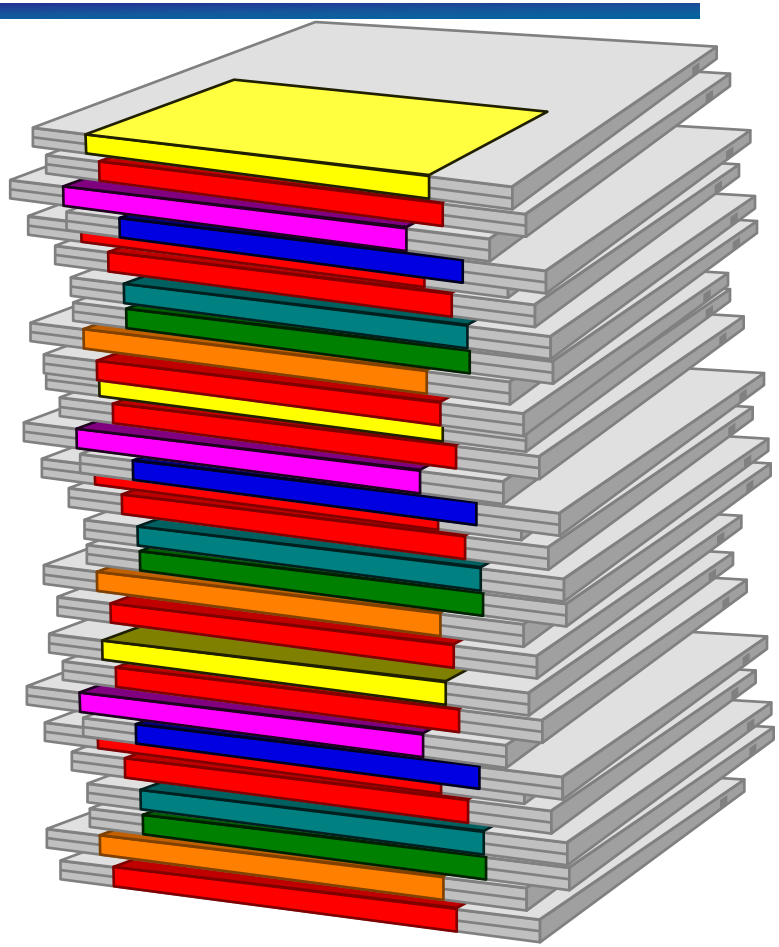
A Producer Wants to Know …



What is Data Warehouse?

- "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."

  —W. H. Inmon



Recognized by many as the father of the data warehouse

Data Warehouses are Integrated

- Constructed by integrating multiple, heterogeneous data sources.
  - relational databases, XML, flat files, on-line transaction records
- Data cleaning and data integration techniques are applied.
  - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources.
    - e.g., Hotel price depends on: currency, various room taxes, whether breakfast or Internet is included, etc.
  - When data from different sources is moved to the warehouse, it is cleaned and converted into a common format.

Typical Data Warehouse Scenario (1/2)

- Consider Canada Safeway's data sources:
  - Operational data from daily purchase transactions, in each store
  - Data about item placement on shelves
  - Supplier data
  - Data about employees, their compensation, etc.
  - Sales/promotion plans
  - Product categories and sub-categories; brands, types; customer demographics; time and date of sale
- Each of the above is essentially an autonomous OLTP database (or set of tables)
  - Local queries; no queries cutting across multiple databases
  - Data must be current at all times
  - Support for concurrency, recovery, and transaction management are a must.

Typical Data Warehouse Scenario (2/2)

- Consider the following use-case queries:
- How does the sale of hamburgers for Feb. 2015 compare with that for Feb. 2014?
- What were the sales of ketchup like last week (when hamburgers and ketchup were placed next to each other) compared to the previous week (when they were far apart)?
- What was the effect of the promotion on ground beef on the sales of hamburger buns and condiments?
- How has the reorganization of the store(s) impacted sales?
  - Be specific here to try to see cause-and-effect—especially with respect to prior periods' sales.
- What was the total sales volume on all frozen food items (not just one item or a small set of items)?

Data Warehouse Integration Challenges

- When getting data from multiple sources, must eliminate mismatches (e.g., different currencies, units, schemas)
- e.g., Shell Canada (Calgary), Shell Oil (Houston), Royal Dutch Shell (Netherlands), etc. may need to deal with data mismatches throughout the multinational organization:
- Exercise: Provide some examples of mismatches that might occur in this example.

Currencies
Decimal places vs. commas
units - metric vs. imperial
dates - ansii standard should rule the world
time zones
currency types
Languages

DW Integration Challenges (cont.)

- e.g., Shell may need to deal with data mismatches throughout the multinational organization:
  - Multiple currencies and dynamic exchange rates
  - Gallons vs. liters; thousands of cubic feet (of gas) vs. cubic meters vs. British Thermal Units (BTUs)
  - Different suppliers, contractors, unions, and business partner relationships
  - Different legal, tax, and royalty structures
  - Local, provincial, federal, and international regulations
  - Different statutory holidays (when reporting holiday sales)
  - Light Sweet Crude (Nigeria) vs. Western Canada Select (Alberta, heavier crude oil)
  - Joint ownership of resources (partners)
  - Retail promotions in its stores; different products

Operational DBMS vs. Data Warehouse summary

- Operational DBMS
  - Day-to-day operations: purchasing, inventory, banking, payroll, manufacturing, registration, accounting, etc.
  - Used to run a business
- Data Warehouse
  - Data analysis and decision making
  - Integrated data spanning long time periods, often augmented with summary information
  - helps to *"optimize"* the business

Why a Separate Data Warehouse?

- High performance for both systems
  - DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery
  - Warehouse—tuned for complex queries, multidimensional views, consolidation
- Different types of queries
  - Extensive use of statistical functions which are poorly supported in DBMS
  - Running queries that involve conditions over time or aggregations over a time period, which are poorly supported in a DBMS
  - Running related queries that are generally written as a collection of independent queries in a DBMS

On-Line Analytical Processing

- Technology used to perform complex analysis of the data in a data warehouse.
  - OLAP enables analysts, managers, and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information.
  - The data has been **E**xtracted **T**ransformed and **L**oaded (ETL) from raw data to reflect the enterprise as understood by the user.
- OLAP queries are, typically:
  - Full of grouping and aggregation
  - Few, but complex queries -- may run for hours

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 students register for courses

  A. OLTP B. OLAP

- UBC wants to figure out how many students are currently taking CPSC 304

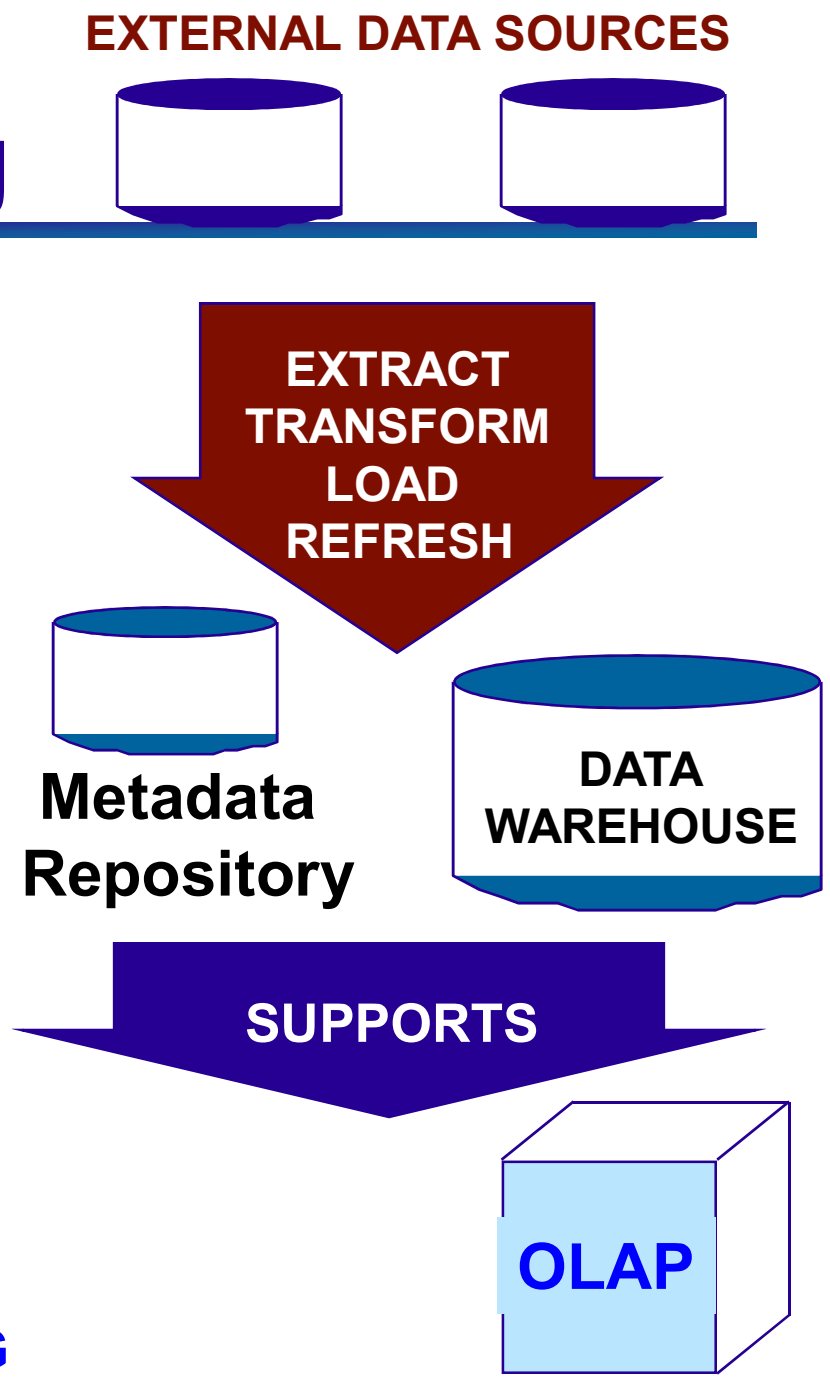
  A. OLTP B. OLAP

- UBC wants to figure out how many students have taken all courses over time and analyze for trends

  A. OLTP B. OLAP

Data Warehousing

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



Business Intelligence: 3 Major Areas

1. Data Warehousing

- Consolidate and integrate operational OLTP databases from many sources into one large, well-organized repository
  - e.g., acquire data from different stores or branches
  - Must handle conflicts in schemas, semantics, platforms, integrity constraints, etc.
  - May need to perform **data cleaning**
- Load data through periodic updates
  - Synchronization and currency considerations
- Maintain an archive of potentially useful historical data including data that is an aggregation or summary, but has been pre-processed.

- Data Warehouses are then used to perform OLAP and Data Mining (DM).

Business Intelligence: 3 Major Areas

2. **OLAP**
   - 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 (business) experts rather than database experts.
3. **Data Mining**
   - Exploratory search for interesting trends (patterns) and anomalies (e.g., outliers, deviations) using more sophisticated algorithms (as opposed to queries).

That is all great, but what are the challenges with data warehousing?

- Semantic Integration: Extract, Transform, Load challenges. We already talked about the Shell example.
- Heterogeneous Sources: Must access data from a variety of source formats and repositories
- DB2, Oracle, SQL Server, Excel, Word, text-based files
- Hundreds of COTS (commercial off-the-shelf software) packages, with export facilities.
- Load, Refresh, and Purge Activities: Must load data, periodically refresh it, and purge old data
- How often?

Data Warehousing Challenges: Extract, Transform, and Load (ETL)

- ETL refers to the processes used to:
  - **Extract** data from homogeneous or heterogeneous data sources
    - Common data-source formats include relational databases, XML, Excel, and flat files.
  - **Transform** the data and store it in a common, standardized format or structure, suitable for querying and analysis.
    - An important function of transformation is *data cleaning*. This operation may take 80% or more of the effort!
  - **Load** the data into the data warehouse

Data Warehousing Challenges: Metadata and approximation

- Metadata Management: Must keep track of source, load time, and other information for all data in the data warehouse.
- Answering Queries Quickly: Approximate answers are often OK.
  - Better to give an approximate answer quickly, than an exact answer *n* minutes later
  - Sampling
  - Snapshot (if the data keeps changing or we want an easily accessible point-in-time result (e.g., end-of-month sales figures))

Data Warehousing Challenges: The need for speed

- Pre-compute and store (materialize) some answers
  - Use a *Data Cube* to store summarized/aggregated data to answer queries, instead of having to go through a much bigger table to find the same answer.
  - The computation is similar in spirit to relational query optimization which is studied in detail in CPSC 404.

OLAP Queries

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

Multidimensional Data Model

- The main relation, which relates dimensions to a measure via foreign keys, is called the ***fact table***.
  - Recall that a FK in one table refers to a candidate key (and most of the time, the primary key) in another table.
  - The fact table has FKs to the dimension tables.
  - These mappings are essential.
- Each dimension can have additional attributes and an associated ***dimension table***.
  - Attributes can be numeric, categorical, temporal, counts, sums
- Fact tables are *much* larger than dimensional tables.
- There can be multiple fact tables.
  - You may wish to have many measures in the same fact table.

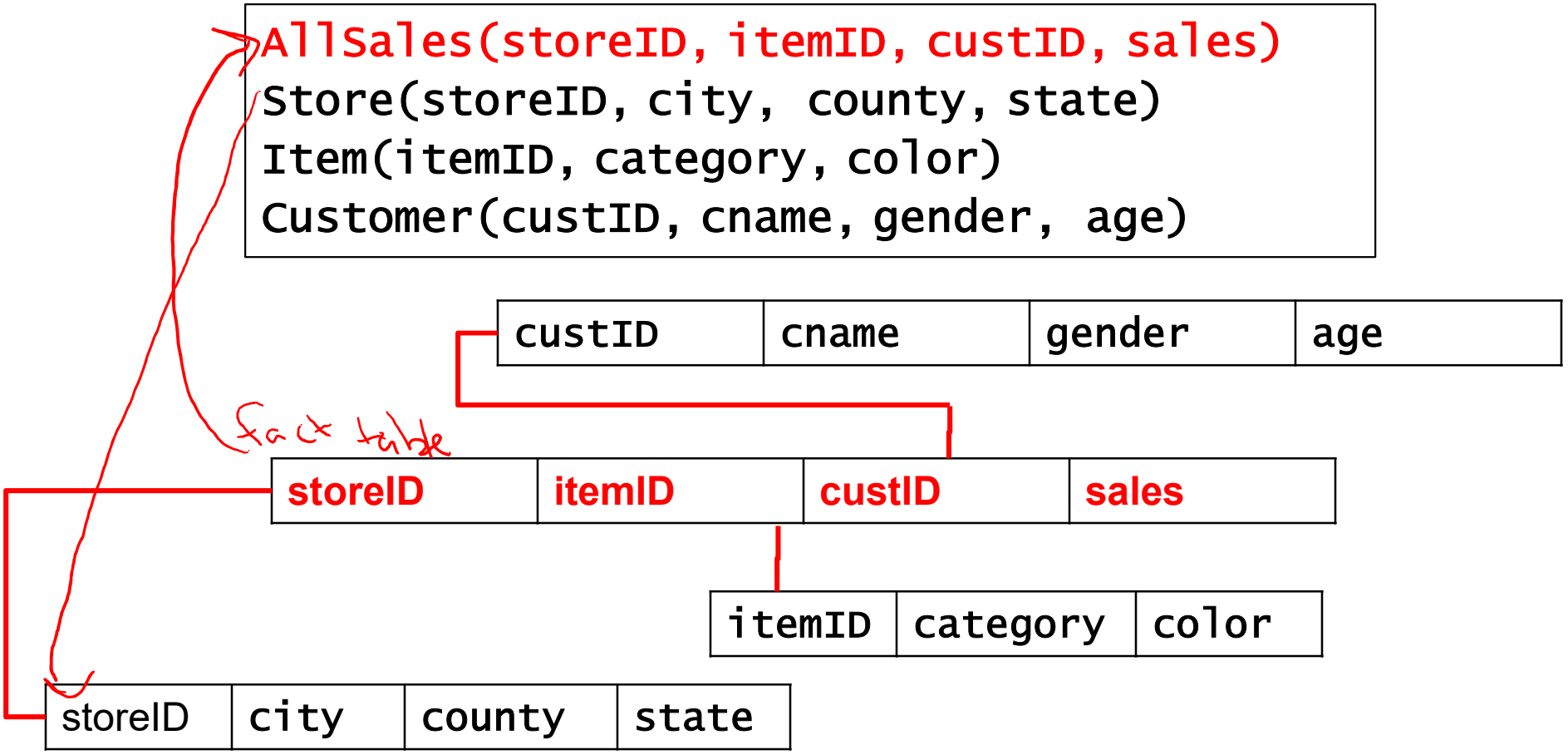
Design Issues

- 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

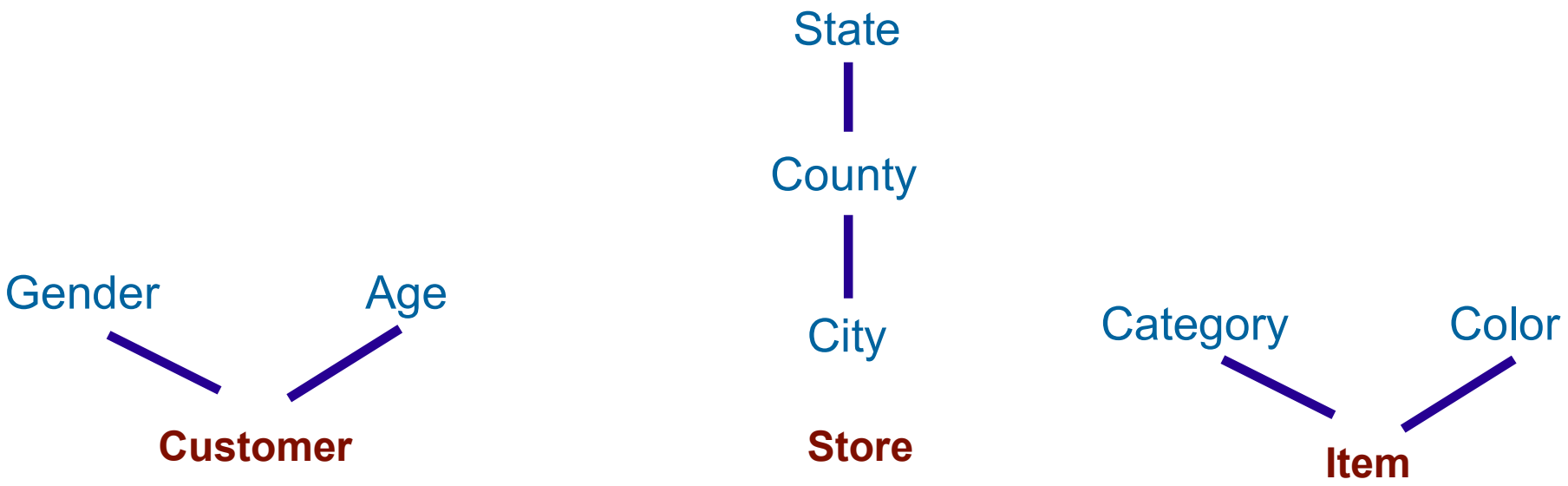
Star Schema – fact table references dimension tables

- Join → Filter → Group → Aggregate



Dimension Hierarchies

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



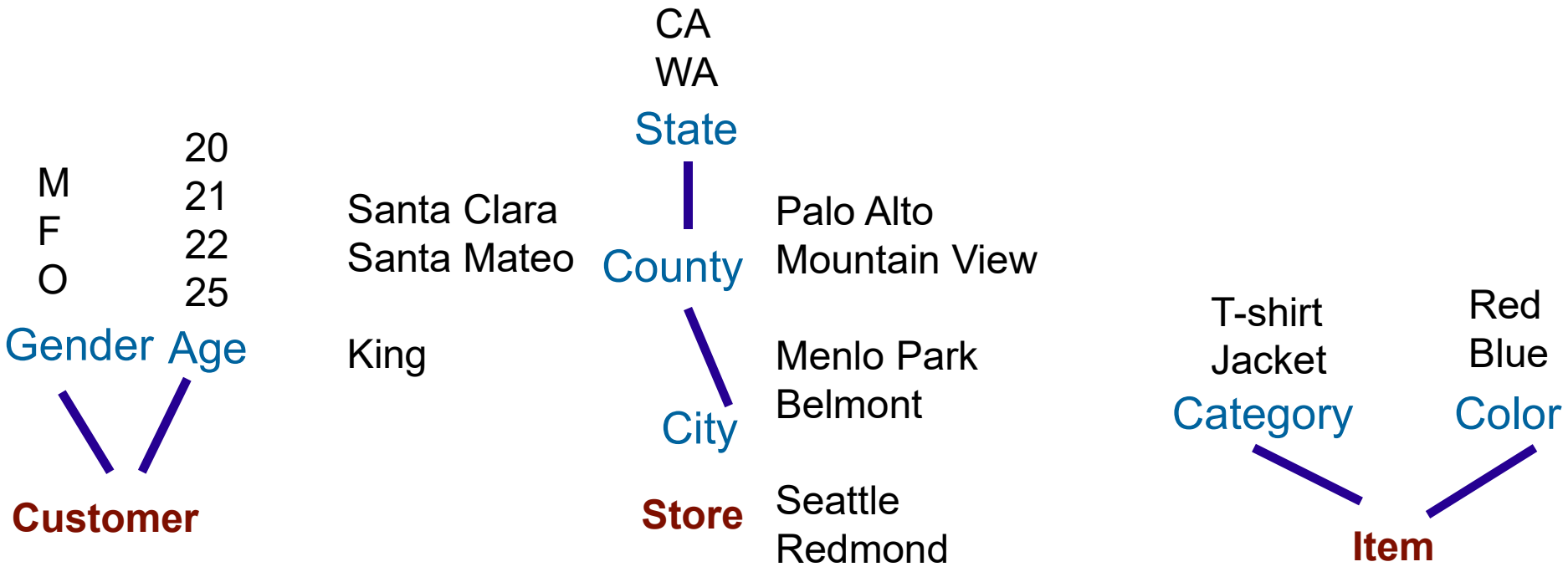
Consider dimensions

Consider building a UBC OLAP application about student data. What are some dimensions along with values in the hierarchy that you might consider (e.g., you may have a "time" dimension, which would consider a hierarchy of times term → Session (W vs. S) → Year → Decade)

Student's age - standing -> decade, undergrad vs. grad
faculties and majors specialization -> major -> faculties
location

Running Example (cont.)

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



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

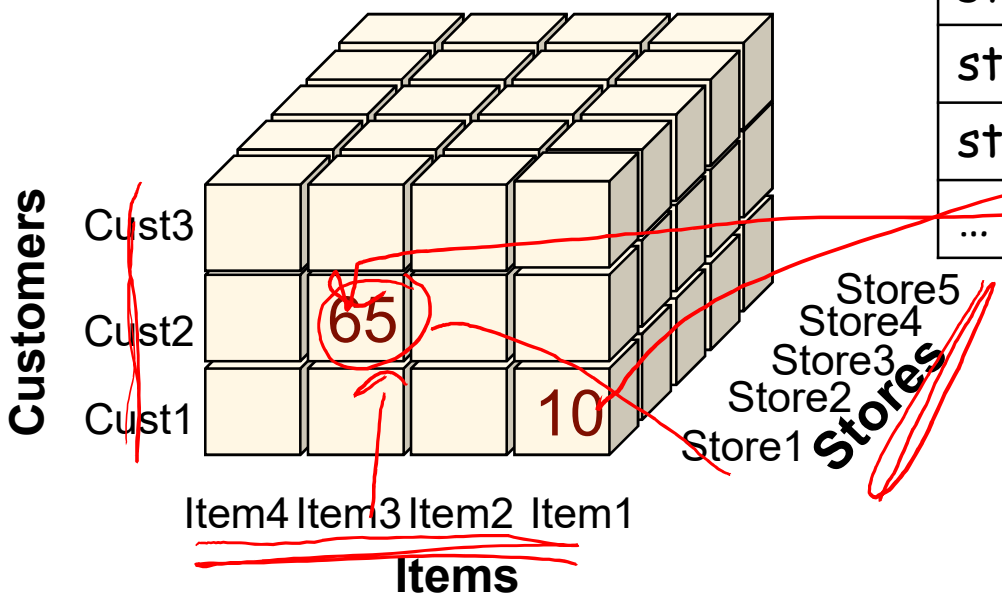
```
SELECT *
FROM    AllSales F, Store S, Item I, Customer C
WHERE  F.storeID = S.storeID and
           F.itemID = I.itemID and
           F.custID = C.custID;
```

- 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.

Full Star Join Summarized

Find total sales by store, item, and customer.

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



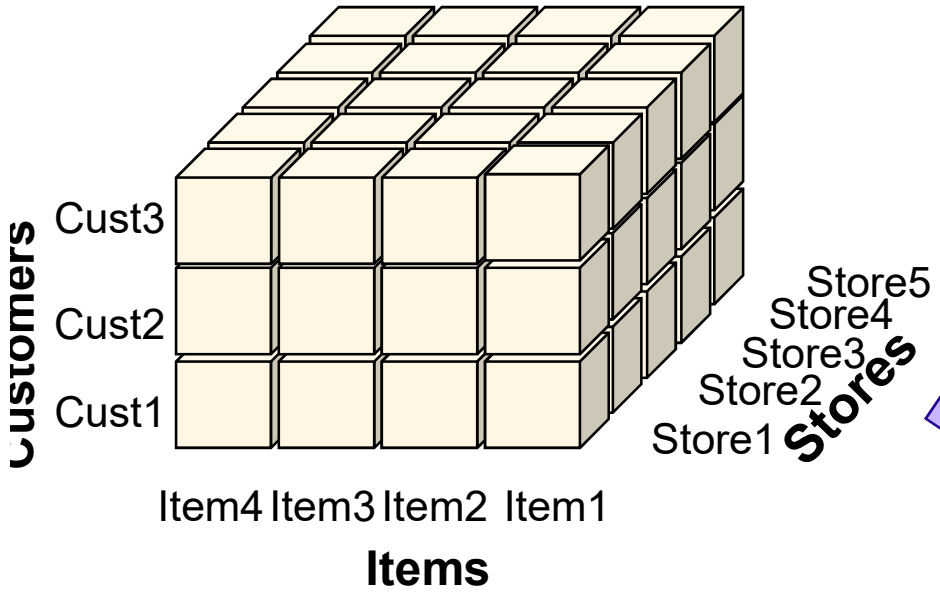
storeID	itemID	custID	Sum (sales)
store1	item1	cust1	10
store1	item3	cust2	65
…	…	…	…

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)

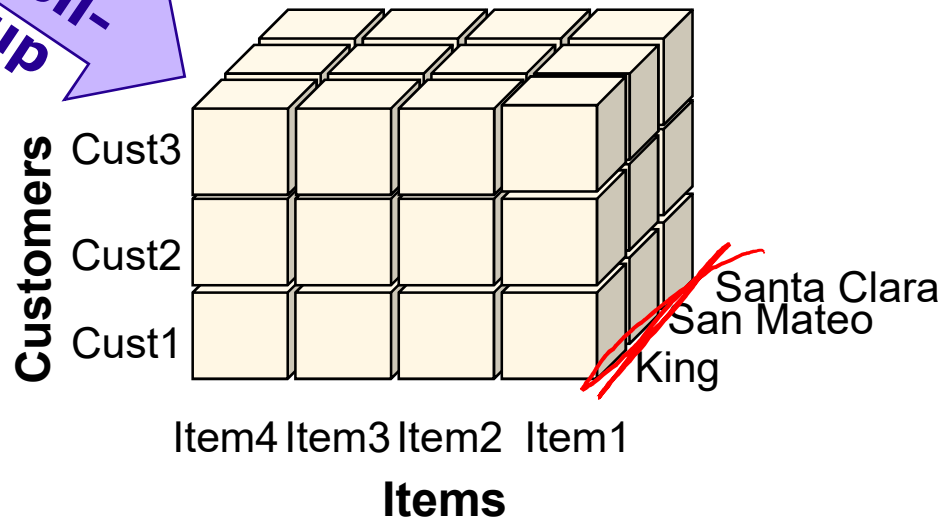
Use Roll-up on total sales by store, item, and customer to find total sales by item and customer for each county.




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

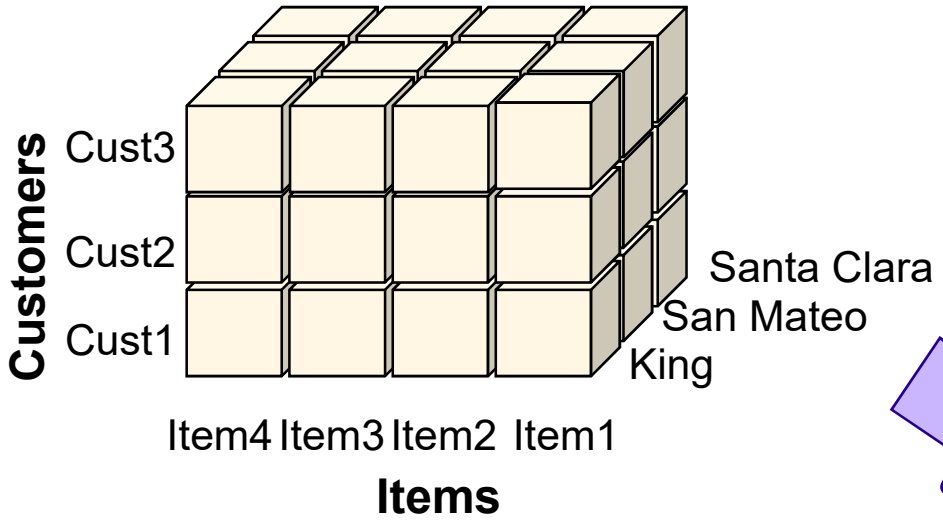
Roll-up

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



Roll-up Example 2 (Hierarchy)

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



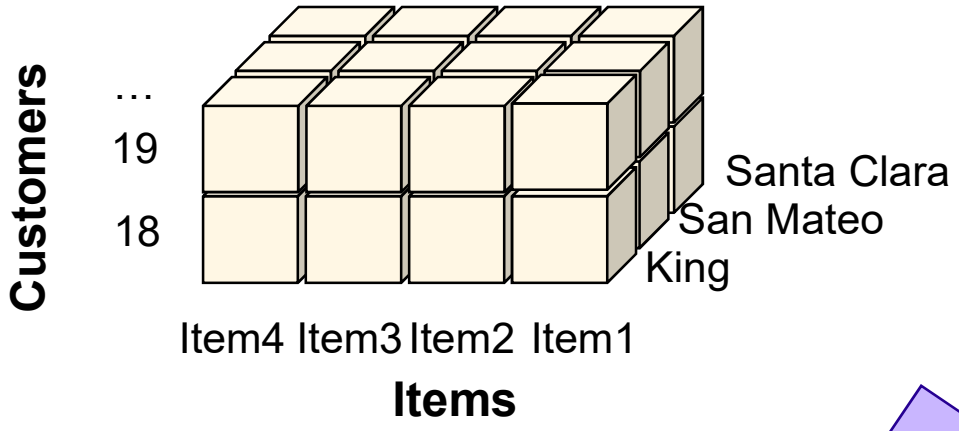

```
SELECT county, itemID, custID,
                SUM(sales)
FROM    AllSales F, Store S
WHERE F.storeID = S.storeID
GROUP BY county, itemID, custID;
```

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)

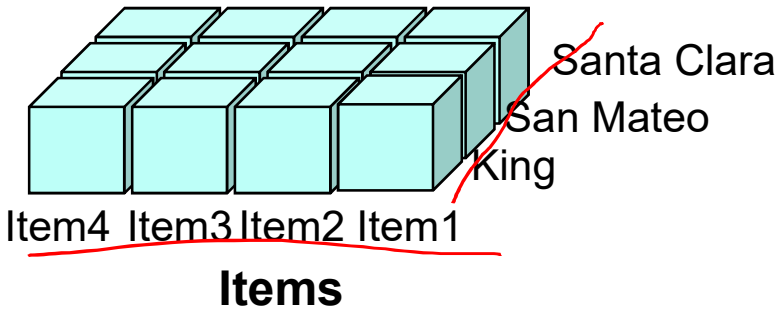
Use Roll-up on total sales by item, age and county to find total sales by item for each county.




```
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;
```

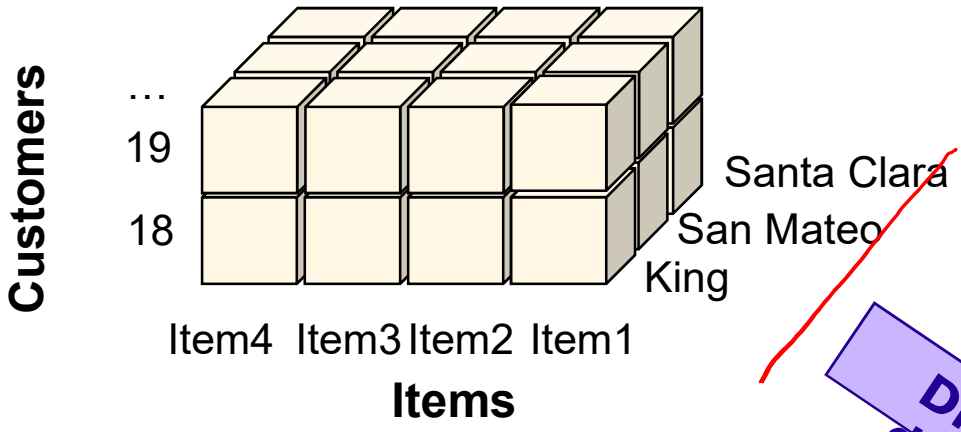


OLAP Queries – Drill-down

- 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)

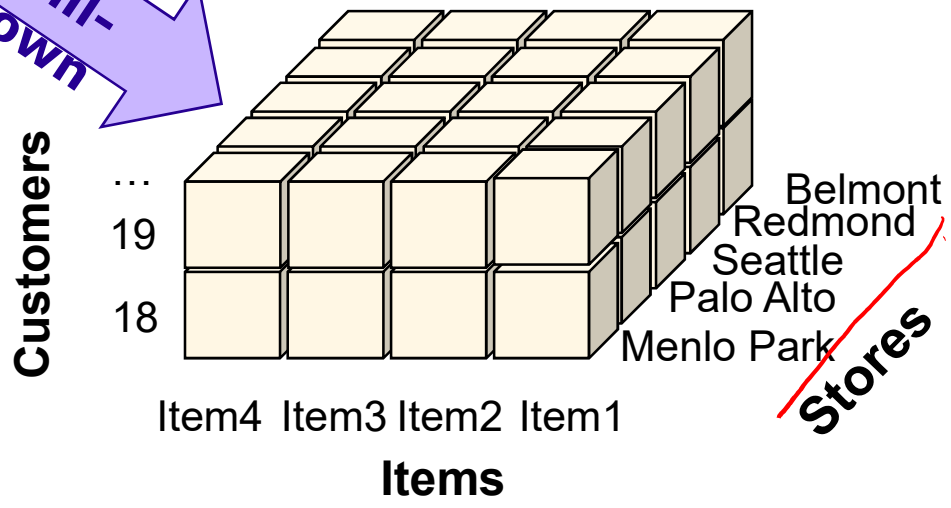
Use Drill-down on total sales by item and age for each county to find total sales by item and age for each city.




```
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;
```

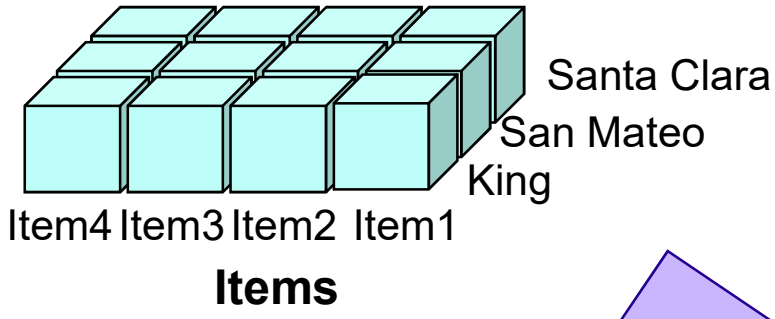
Drill-down

```
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)

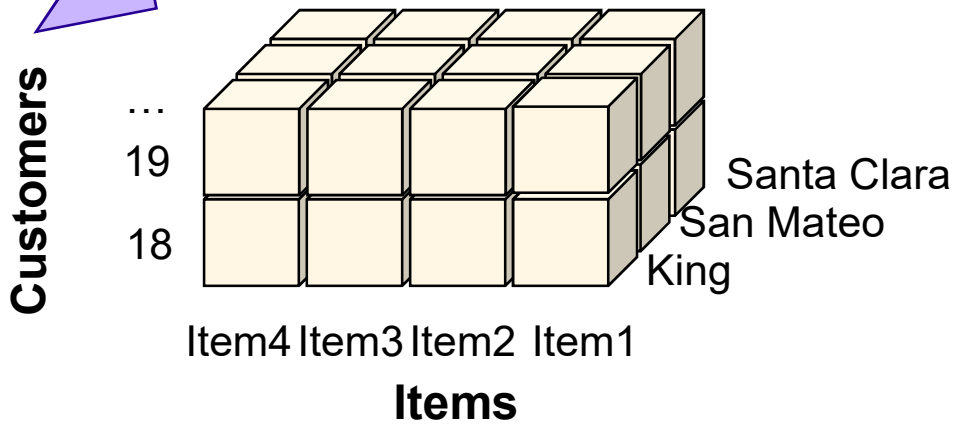
Use Drill-down on total sales by item and county to find total sales by item and age for each county.




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

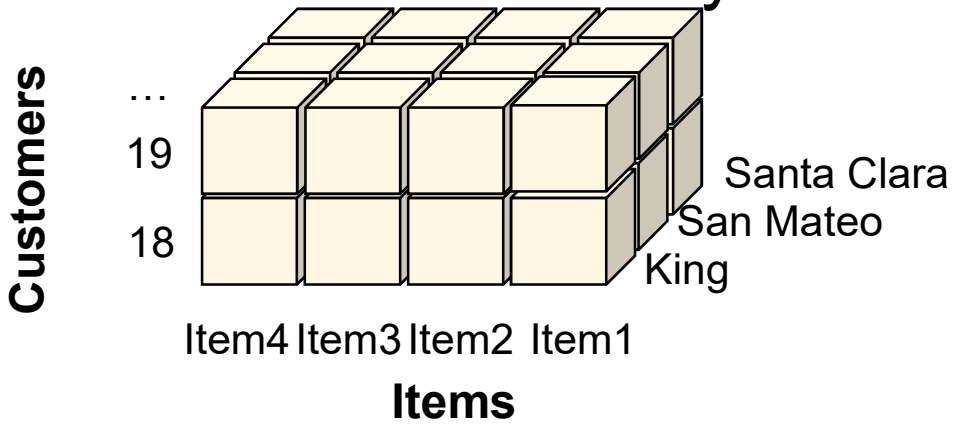
Drill-down

```
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

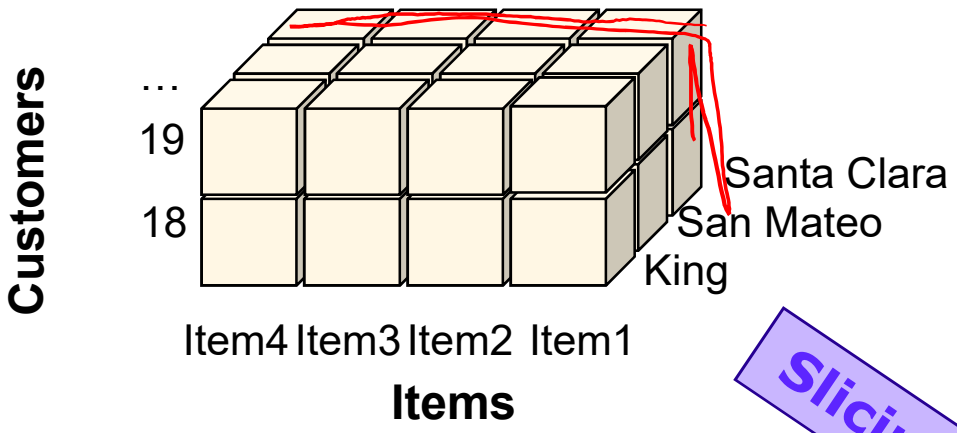
- The slice operation produces a slice of the cube by picking a specific value for one of the dimensions.
- To start our example, let's specify:
  - Total sales by item and age for each county




```
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

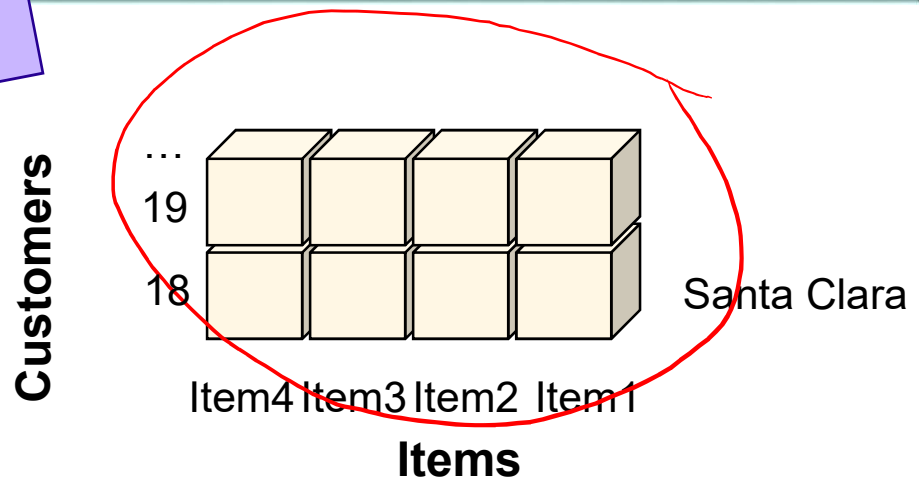
Use Slicing on total sales by item and age for each county to find total sales by item and age for Santa Clara.




```
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;
```

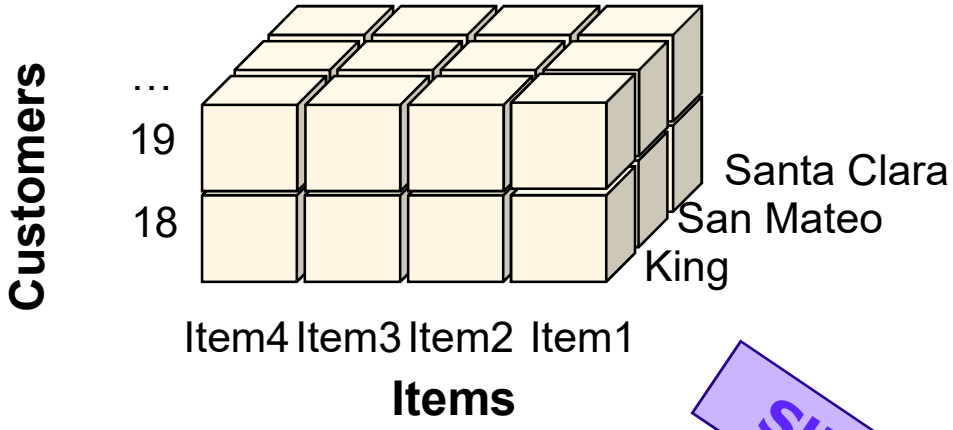
Slicing

```
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

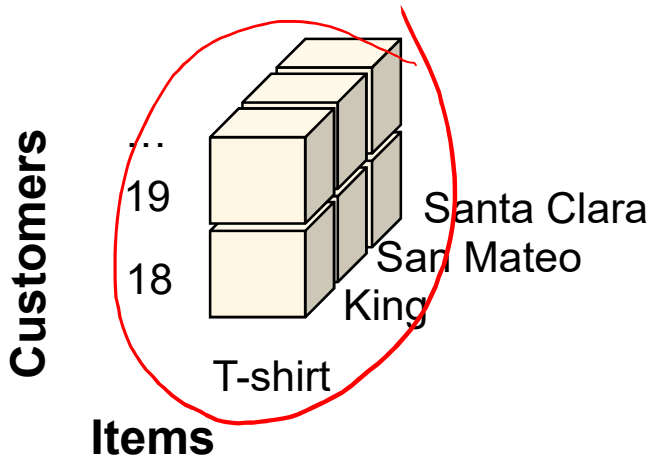
Use Slicing on total sales by item and age for each county to find total sales by age and county for T-shirts.




```
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;
```

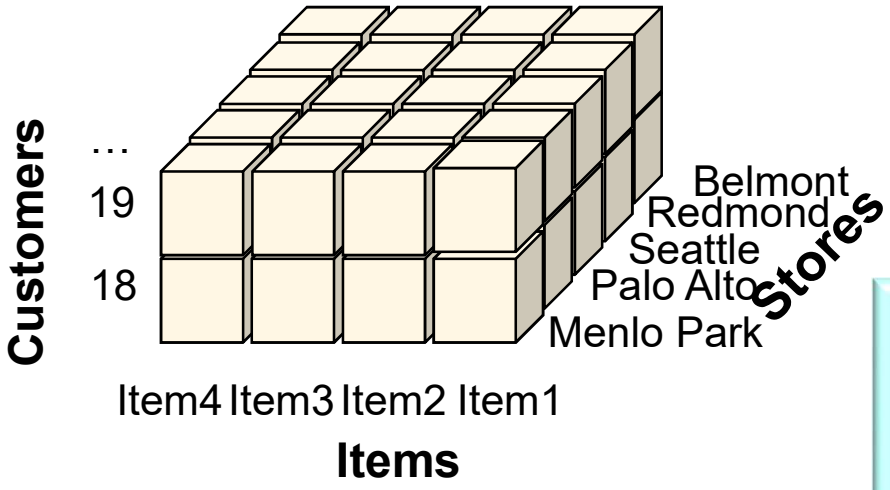
Slicing

```
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

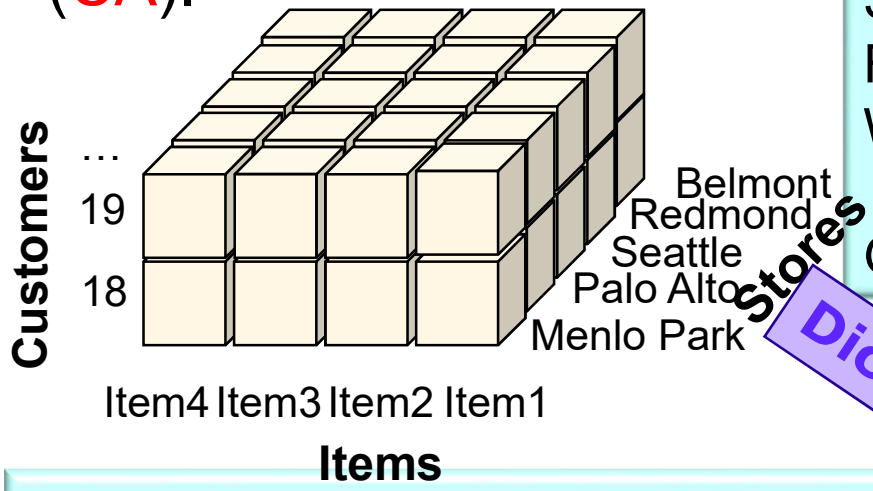
- The dice operation produces a sub-cube by picking specific values for multiple dimensions.
- To start our example, let's specify:
  - Total sales by age, item, and city




```
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

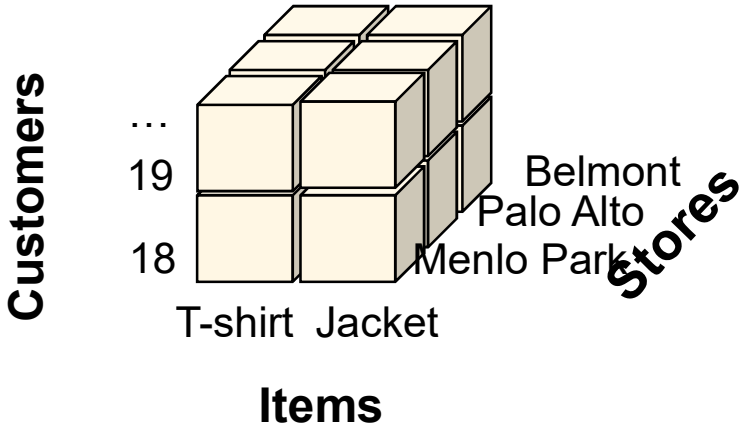
Use Dicing on total sales by age, item, and city to find total sales by age, category, and city for red items in the state of California (CA).




```
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