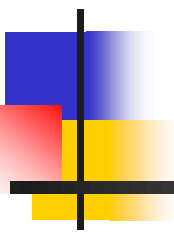


Chapter 10 Advanced topics in relational databases

- 
-
- Security and user authorization in SQL
 - Recursion in SQL
 - Object-relational model
 1. User-defined types in SQL
 2. Operations on object-relational data
 - Online analytic processing & data cubes



Overview

- Traditional database systems are tuned to many, small, simple queries.

Eg: Find the price of bud in Joe's bar?

- Some new applications use fewer, more time-consuming, analytic queries.

Eg: What are the average prices of each bar in the last 3 months? To see how the price varies by times.

- New architectures have been developed to handle analytic queries efficiently.



OLTP

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



OLAP

- On-Line Application Processing (OLAP, or “analytic”) queries are, typically:
- **Few, but complex queries** --- may run for hours.
- Queries do not depend on having an absolutely **up-to-date** database.



The Data Warehouse

- The most common form of data integration.
 1. Copy sources into a single DB (warehouse) and try to keep it up-to-date.
 2. Usual method: periodic reconstruction of the warehouse, perhaps overnight.
 3. Complex Queries (touch large portion of data for analytic queries.)
 4. Infrequent updates.



Common Architecture

- Databases at store branches handle **OLTP**.
- Local store databases copied to a central warehouse overnight.
- Analysts use the warehouse for **OLAP**.



Star Schemas

- A star schema is a common organization for data at a warehouse. It consists of:
 1. **Fact table** : a very large accumulation of facts such as sales.
Often “insert-only.”
 2. **Dimension tables** : smaller, generally static information about the entities involved in the facts.



Example: Star Schema

- Record in a warehouse **information about every beer sale**: the bar, the brand of beer, the drinker who bought the beer, the day, the time, and the price charged.
- The fact table is a relation:

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



Example -- Continued

- The dimension tables include information about the bar, beer, and drinker “dimensions”:

Bars(bar, addr, license)

Beers(beer, manf)

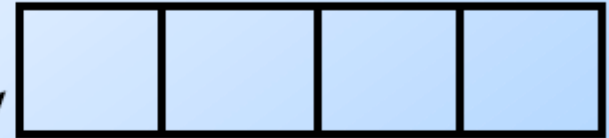
Drinkers(drinker, addr, phone)

Visualization – Star Schema

Dimension Table (**Bars**)



Dimension Table (**Drinkers**)



Dimension Attrs.

Dependent Attrs.



Fact Table - **Sales**



Dimension Table (**Beers**)



Dimension Table (etc.)

Time dimension is very special
Days(day,week,month,year)

Dimensions and Dependent Attributes

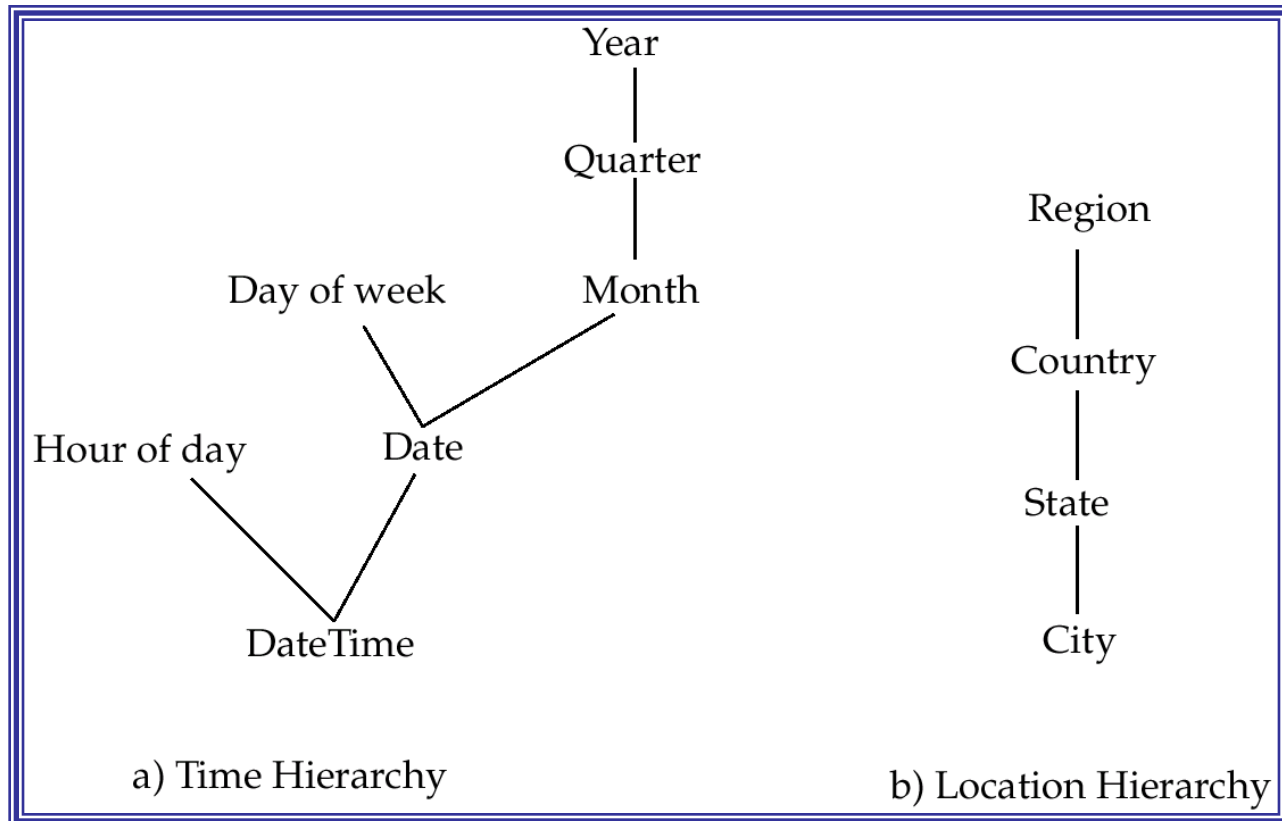


- Two classes of fact-table attributes:
 1. ***Dimension*** attributes : the key of a dimension table, such as **bar**.
 2. ***Dependent*** attributes : a value determined by the dimension attributes of the tuple, such as **price**.

Dimensions Attributes

- **Hierarchy** on dimension attributes: lets dimensions to be viewed at different levels of detail

👉 E.g. the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year





Example: Dependent Attribute

- **Price** is the dependent attribute.
- It is determined by the combination of dimension attributes: bar, beer, drinker, and the time (combination of day and time-of-day attributes).



Approaches to Building Warehouses

1. **ROLAP** = “relational OLAP”: Tune a relational DBMS to support star schemas.
2. **MOLAP** = “multidimensional OLAP”: Use a specialized DBMS with a model such as the “data cube.”



ROLAP Techniques

1. *Bitmap indexes* : For each key value of a dimension table (e.g., each beer for relation Beers) create a **bit-vector** telling which tuple of the fact table have that value.
2. *Materialized views* : Store the answers to several useful queries (views) in the warehouse itself.



Bitmap Index

- Sales(bar, beer, drinker, day, time, price)
- Assume there are 4 kinds of beer, the **bitmap** index on Beer is 4 columns.

Fact Table

Bud	Budlit	Coor	Qinda
1	0	0	0
0	1	0	0

bar	beer	drinker	day	time	price
	Bud				
	Budlit				
	...				



Typical OLAP Queries

- OLAP queries begin with a “star join”: the **natural join of the fact table with all or most of the dimension tables.**

- **Example:**

```
SELECT *
```

```
FROM Sales, Bars, Beers, Drinkers
```

```
WHERE Sales.bar = Bars.bar AND
```

```
    Sales.beer = Beers.beer AND
```

```
    Sales.drinker = Drinkers.drinker;
```



Typical OLAP Queries (cont.)

- The typical OLAP query will:
 1. Start with a star join.
 2. Select for interesting tuples, based on **dimension data**.
 3. Group by **one or more dimensions**.
 4. Aggregate **certain attributes** of the result.



Example: OLAP Query

For **each bar** in **Palo Alto**, find the **total sale** of **each beer** manufactured by **Anheuser-Busch**.

2. Filter: *addr* = "Palo Alto" and *manf* = "Anheuser-Busch".
3. Grouping: by bar and beer.
4. Aggregation: Sum of *price*.



Example: In SQL

```
SELECT bar, beer, SUM(price)
FROM Sales NATURAL JOIN Bars
      NATURAL JOIN Beers
WHERE addr = 'Palo Alto' AND
      manf = 'Anheuser-Busch'
GROUP BY bar, beer;
```



Using Materialized Views

- A direct execution of this query from Sales and the dimension tables could take **too long**.
- If we create **a materialized view** that contains enough information, we may be able to answer our query **much faster**.



Example: Materialized View

- Which views could help with our query?
- Key issues:
 1. It must join **Sales**, **Bars**, and **Beers**, at least.
 2. It must group by at least bar and beer.
 3. It must not select out **Palo-Alto** bars or **Anheuser-Busch** beers.
 4. It must not project out *addr* or *manf*.

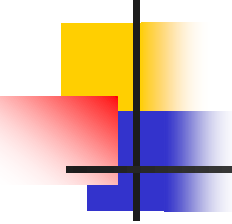


Example --- Continued

- Here is a materialized view that could help:

```
CREATE VIEW BABMS (bar, addr,  
    beer, manf, sales) AS  
(SELECT bar, addr, beer, manf,  
    SUM(price) as sales  
FROM Sales NATURAL JOIN Bars  
    NATURAL JOIN Beers  
GROUP BY bar, addr, beer, manf);
```

Since bar -> addr and beer -> manf, there is no real grouping. We need addr and manf in the SELECT.



Example --- Concluded

- using **the materialized view BABMS**:

```
SELECT bar, beer, sales
FROM BABMS
WHERE addr = 'Palo Alto' AND
       manf = 'Anheuser-Busch';
```



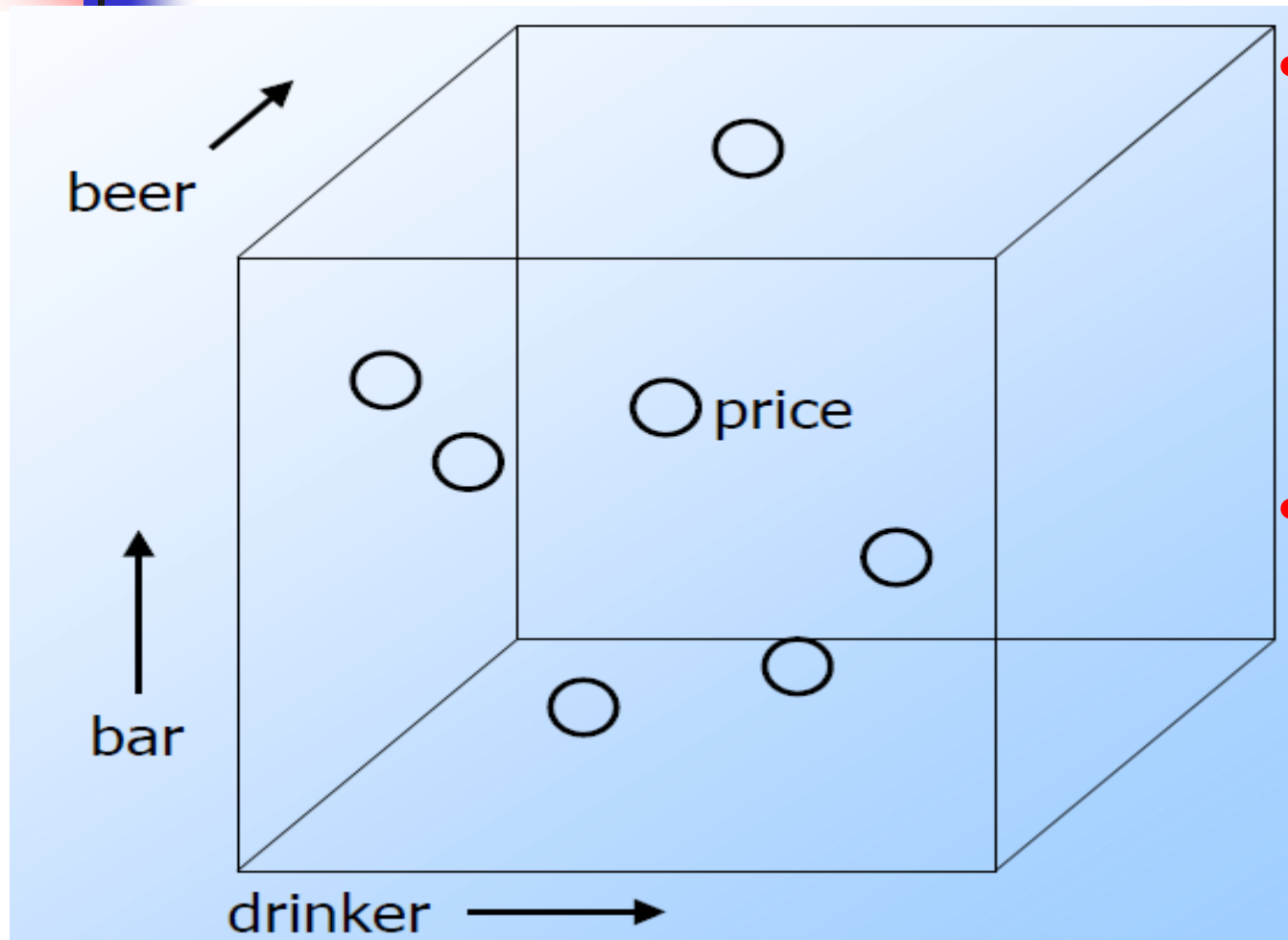

MOLAP and Data Cubes

- Keys of dimension tables are the **dimensions** of a **hypercube**.

Example: for the Sales data, the four dimensions are bar, beer, drinker, and time.

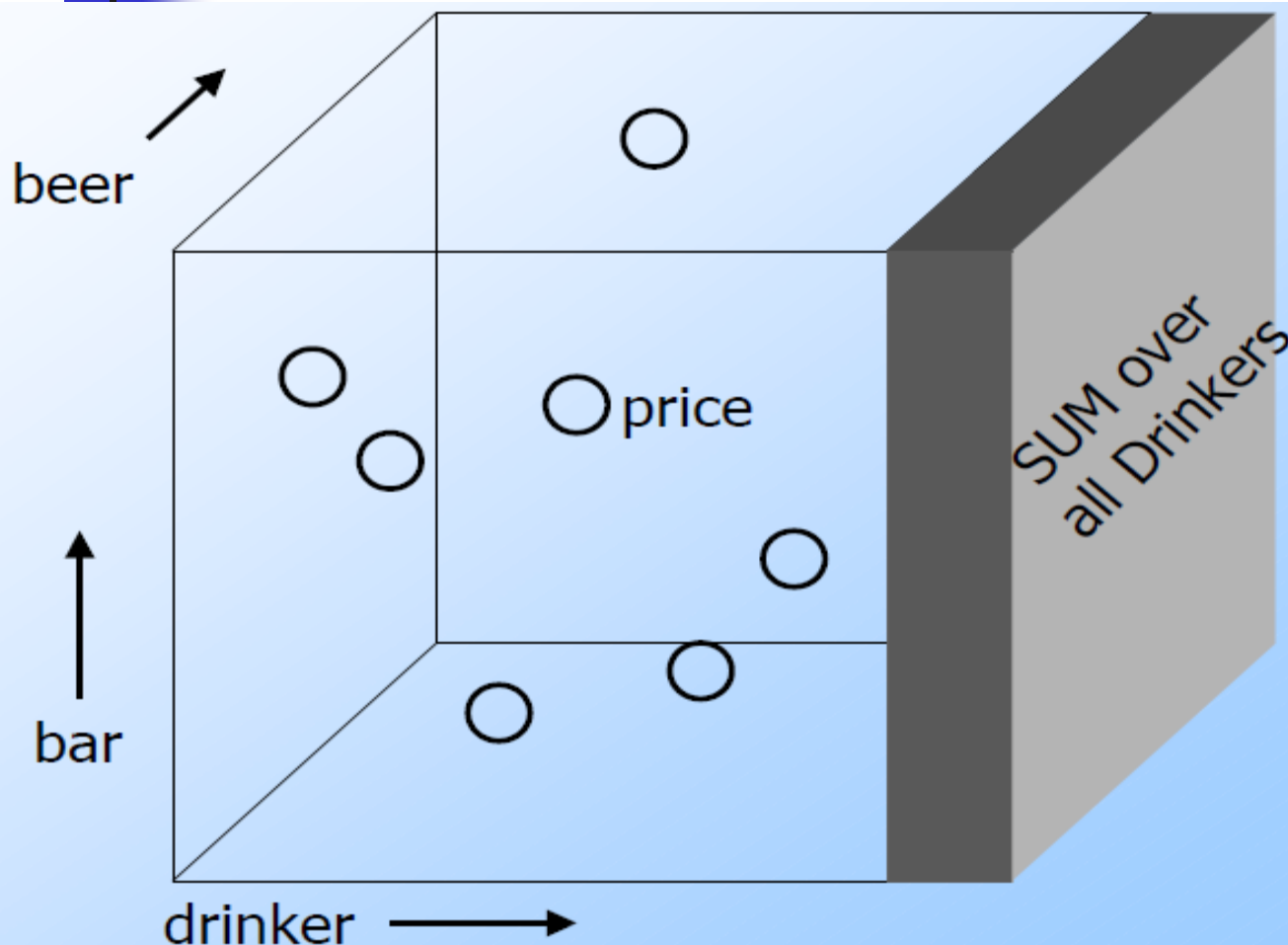
- Dependent attributes (e.g., price) appear at the **points** of the **cube**.

Visualization – Data Cubes



- Dimension data forms axes of "cube"
- Dependent data in cells.

Visualization --- Data Cube w/Aggregation



Aggregated
data on
sides,
edges,
cornor.



Example: Data Cube

Data cube with multiple dimensions, for example:

- 4-dimensional Sales cube includes the sum of price over each **bar**, each **beer**, each **drinker**, and each **time unit** (perhaps days).
- How to show it ?
- How to show those aggregated values?

Structure of the Cube



- Think of each dimension as a component of a tuple

Sales('joe'sbar', 'Bud', 'Mary', 2013-05-27)

- Think of each dimension as having an additional value *****. A point with one or more *****'s in its coordinates aggregates over the dimensions with the *****'s.

Sales("Joe's Bar", "Bud", *, *) holds the sum, over **all drinkers and all time, of the Bud consumed at Joe's.**



Drill-Down

- Drill-down = “de-aggregate” = break an aggregate into its constituents.
- Example: having determined that Joe’s Bar sells very few Anheuser-Busch beers, **break down his sales by particular A.-B. beer.**



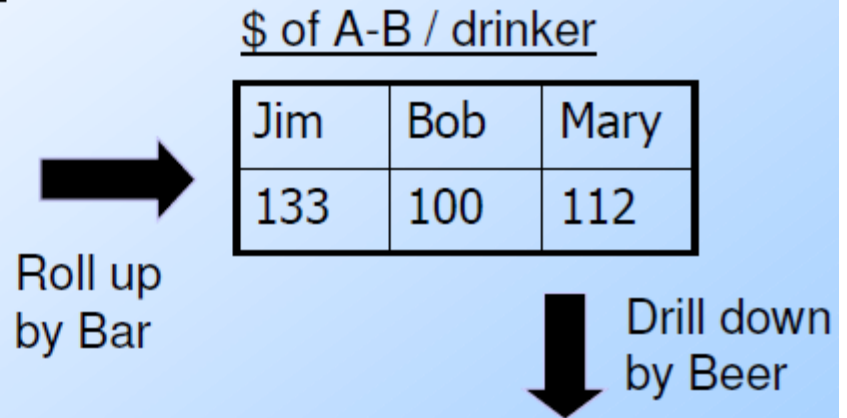
Roll-Up

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

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



\$ of A-B Beers / drinker

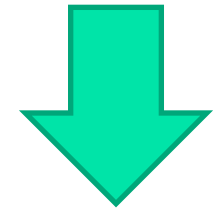
	Jim	Bob	Mary
Bud	40	29	40
M'lob	45	31	37
Bud Light	48	40	35

\$ means the price charged.

Examples: Roll Up



Select manf, bar, drinker, sum(price)
From sales natural join beers
Group by manf, **bar**, drinker;



Select manf, drinker, sum(price)
From sales natural join beers
Group by manf, drinker;

Example: Drill Down (cont.)

Select manf, drinker, sum(price)
From sales natural join beers
Group by manf, drinker;



Select manf, **beer**, drinker, sum(price)
From sales natural join beers
Group by manf, **beer**, drinker;



The general form of “slicing and dicing”

Slicing: focusing on particular one dimension with fixed value.

Dicing: focusing on particular partitions along one or more dimensions.

Select <*grouping attributes and aggregations*>

From <*fact table joined with some dimension tables*>

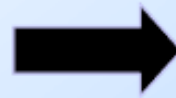
Where <*certain attributes are constant*>

Group by <*grouping attributes*>;

Same example

\$ 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

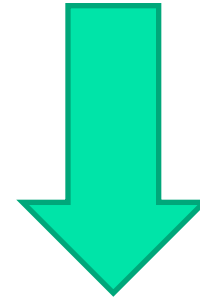
\$ of A-B Beers / drinker

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

Examples: Slicing



Select manf, bar, drinker, sum(price)
From sales natural join beers
Group by manf, bar, drinker;



Select bar, drinker, sum(price)
From sales natural join beers
Where manf='Anheuse-Busch'
Group by bar, drinker;

Example: Dicing

Find the sales of those bar located in Palo Alto and sold beers from manufacture of "A-B":

Select bar, drinker, sum(price)



From sales natural join beers natural join bars

Where manf='Anheuse-Busch' and addr='Palo Alto'

Group by bar, drinker;

The **cube operator** in SQL

--with cube

Select dimension-attributes

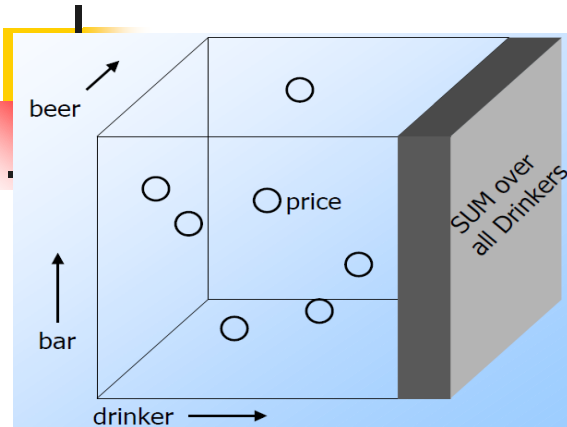
From tables

Where conditions

Group by dimension-attributes **with cube**

- Add to results: faces, edges and corner of cube using null value.

The cube operator in SQL (examples)



- Construct a materialized view that is data cube

Create **materialized view salesCube** as

Select drink,bar,beer, sum(price)

From sales

Group by drink,bar,beer **WITH CUBE;**

Null is used to indicate a rolled-up dimension, similar as *

(Jim,Joe'sbar,Bud,10)

(Jim, null, Bud, 20)

(Jim, Joe'sbar, null, 45)

(null, Joesbar, Bud,55)

(Jim, null,null,133)

(null,Joesbar,null,80)

(nill,null,Bud,80)

(null, null, null,345)

...

SQL: Cube Operator

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes

E.g. consider the query

```
select drinker, bar, beer, sum(price)
```

```
from sales
```

```
group by drinker, bar, beer with cube
```

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

{ (*drinker, bar, beer*), (*drinker, bar*), (*drinker, beer*),
(*bar, beer*), (*drinker*), (*bar*), (*beer*), () }

where () denotes an empty **group by** list.

- For each grouping, **the result contains the null value for attributes not present in the grouping.**

The cube operator in SQL

--with rollup

Select dimension-attributes

From tables

Where conditions

Group by dimension-attributes with rollup

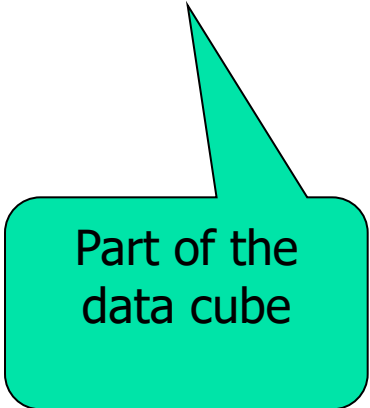
The cube operator in SQL (example)



```
Create materialized view salesRollup as  
Select drink,bar,beer, sum(price)  
From sales  
Group by drink,bar,beer WITH ROLLUP;
```

Will contain tuples:

```
(Jim, Joe'sbar, Bud, 20)  
(Jim, Joe'sbar, null, 45)  
(Jim, null,null,133)  
(null,null,null, 345)
```



Part of the
data cube

SQL : Cube operator with Rollup

- The **rollup** construct generates union on every prefix of specified list of attributes
- E.g.

```
select drinker, bar, beer, sum(price)  
from sales  
group by drinker, bar, beer with Rollup
```
- Generates **union of four groupings**:
{ (*drinker, bar, beer*), (*drinker, bar*), (*drinker*), () }
- Rollup can be used to generate aggregates at **multiple levels of a hierarchy**.

Classroom Exercises

--mydbdata (online)

- create table **students**(sid int primary key,name char[10],dept char[2],age int default 20);
- create table **courses** (cid int primary key, cname text, spring boolean, teacher char[10]);
- create table **teachers**(teacher char[10] unique,dept char[2]);
- create table **sc** (**sid** int, **cid** int, **teacher** char [10],grade int); --a fact table (three dimensions)



Classroom Exercises:

- **Find** students information who take a course in **autumn** and **the teacher** from `'cs'`
- **Find** the average grade for *each student's dept* and *each course*
- **Roll up** to find the average grade for *each course*
- **Drill down** to see *each students*
- **Slice** the average score for *C++*



Fact Table

Create view **fact** as

Select ***sid,cid,dept, grade***

from sc natural join teachers

Group by sid,cid,dept;

--here **dept** is from the teacher



Find the average grade for each dept according to their students grade.

Select students.dept,avg(grade)

From **fact**, students

Where fact.sid=students.sid

Group by students.dept;



Find the average grade for each dept **according to the courses.**

Select dept,avg(grade)

From **fact**

Group by dept;



With Cube

```
Select sid,cid,avg(grade)
from fact
group by sid,cid with cube
```

```
create view scube as
```

```
select sid,cid,avg(grade) from fact group by
sid,cid
```

```
union
```

```
select sid,null,avg(grade) from fact group by sid
```

```
union
```

```
select null,cid,avg(grade) from fact group by cid
```

```
union
```

```
select null,null,avg(grade) from fact;
```



Queries to scube

sid	cid	Avg(grade)

- ◆ Learn the average grade of each students.
- ◆ Learn the average grade of each courses.
- ◆ Learn the average grade of students of all the courses taken.



With RollUp

```
Select sid,cid,avg(grade)
from fact
group by sid,cid with RollUp
```

```
create view srollup as
```

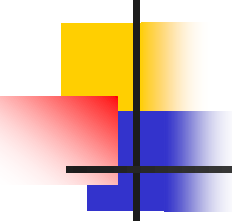
```
  select sid,cid,avg(grade) from fact group by
sid,cid
```

```
  union
```

```
  select sid,null,avg(grade) from fact group by sid
```

```
  union
```

```
  select null,null,avg(grade) from fact;
```



Number of tuples in scube and scrollup

- `Select count(*) from fact;`
- `Select count(distinct sid) from fact;`
- `Select count(distinct cid) from fact;`

- `Select count(*) from scube;`
- `Select count(*) from scrollup;`

Create three dimensional data cube



Select sid,cid,dept,avg(grade) from fact
group by sid,cid,dept with cube



Chapter Summary

- Privileges & Grant diagrams
- SQL Recursive Queries
- Object-relational model
- UDT
- OLAP:

star schemas, rollup, drill-down, slicing, dicing and cube operator