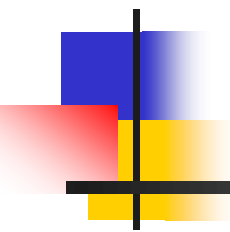


# Chapter 10 Advanced topics in relational databases

- 
- 
- 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. Frequently essential for analytic queries.



# 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

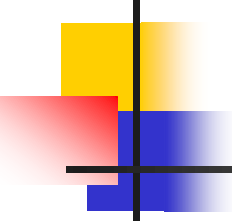
---

- Suppose we want to 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

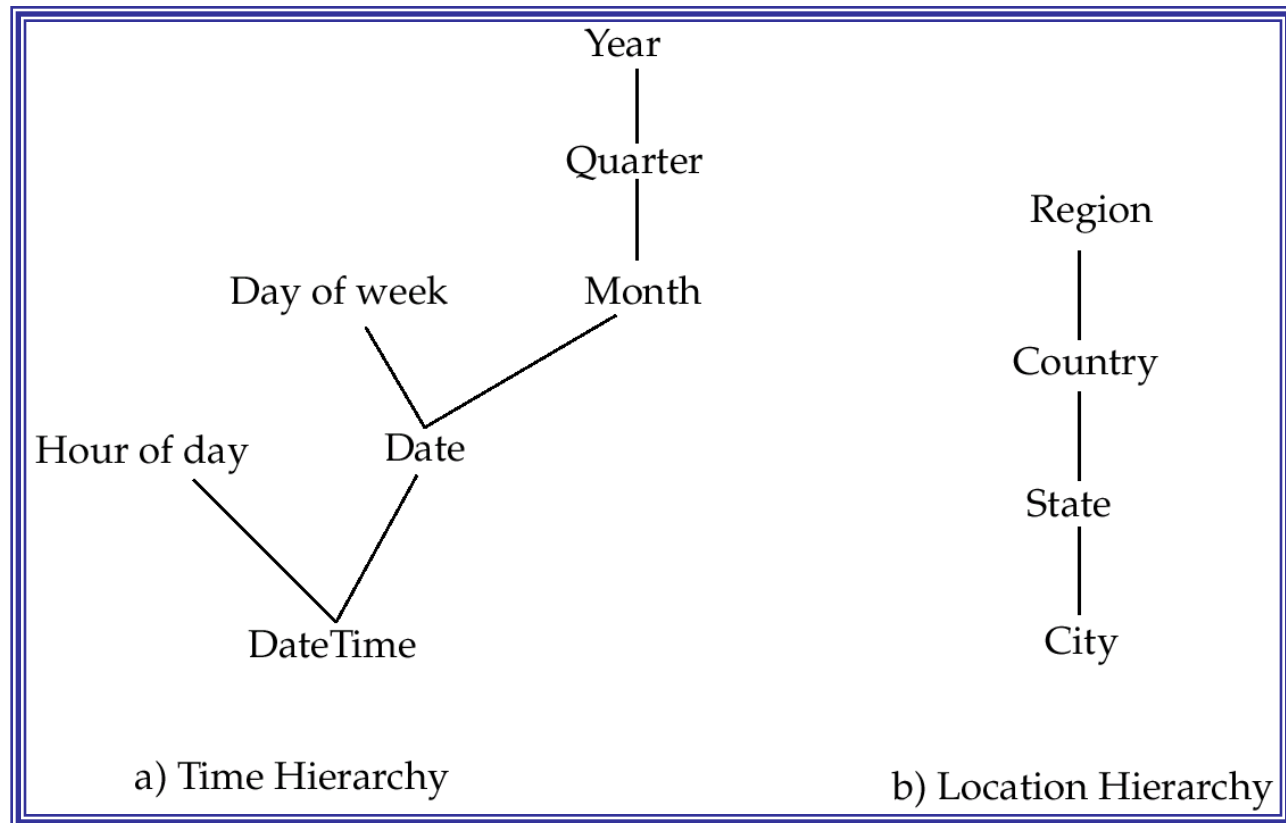
---

- 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 of our example Sales relation.
- 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 tuples of the fact table have that value.
2. *Materialized views* : Store the answers to several useful queries (views) in the warehouse itself.



# Typical OLAP Queries

---

- Often, 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*.

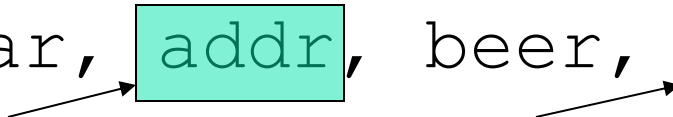


## Example --- Continued

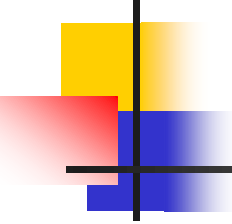
---

- 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

---

- Here's our query using the materialized view BABMS:

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



# MOLAP and Data Cubes

---

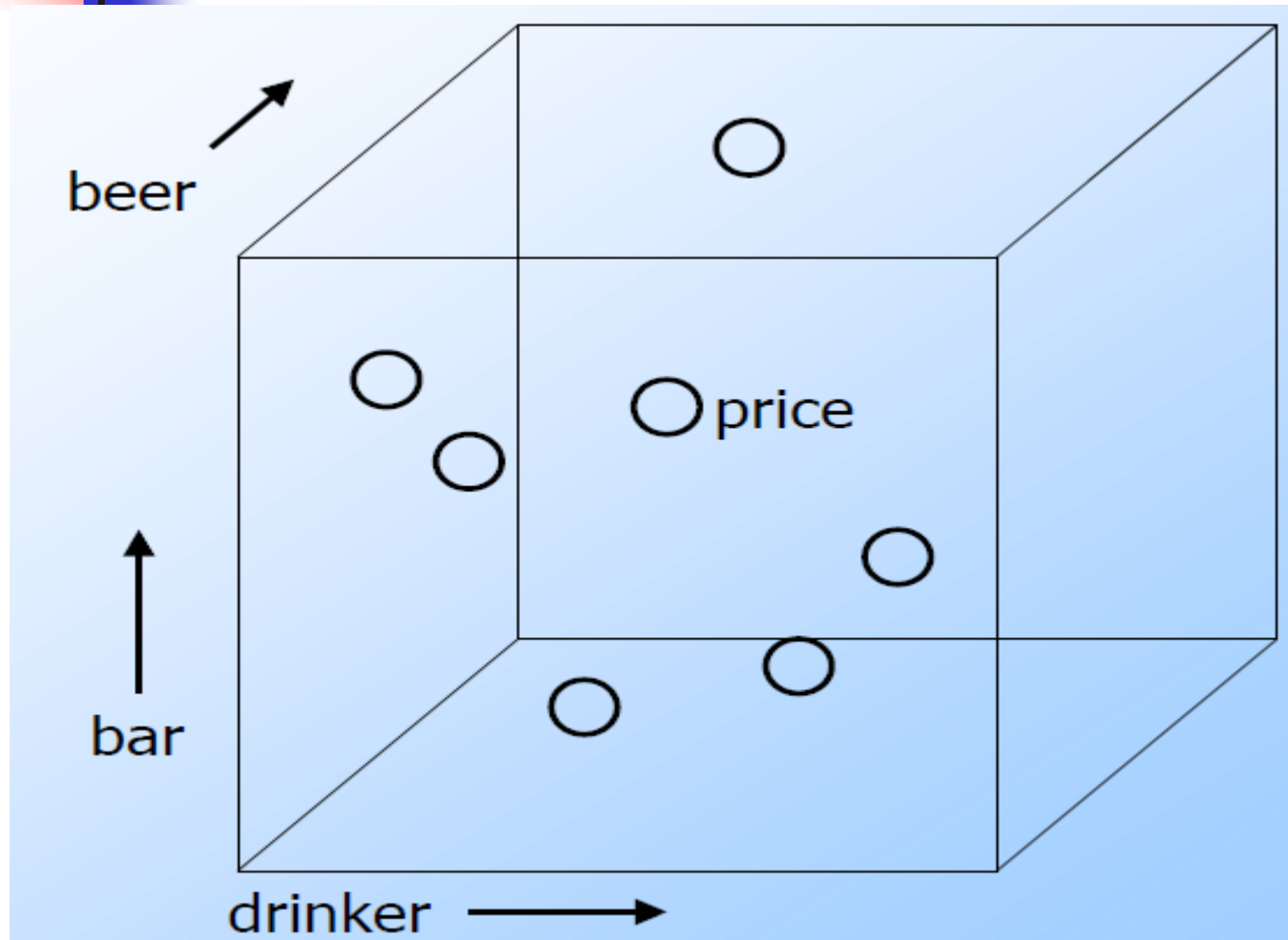
- Keys of dimension tables are the dimensions of a hypercube.

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

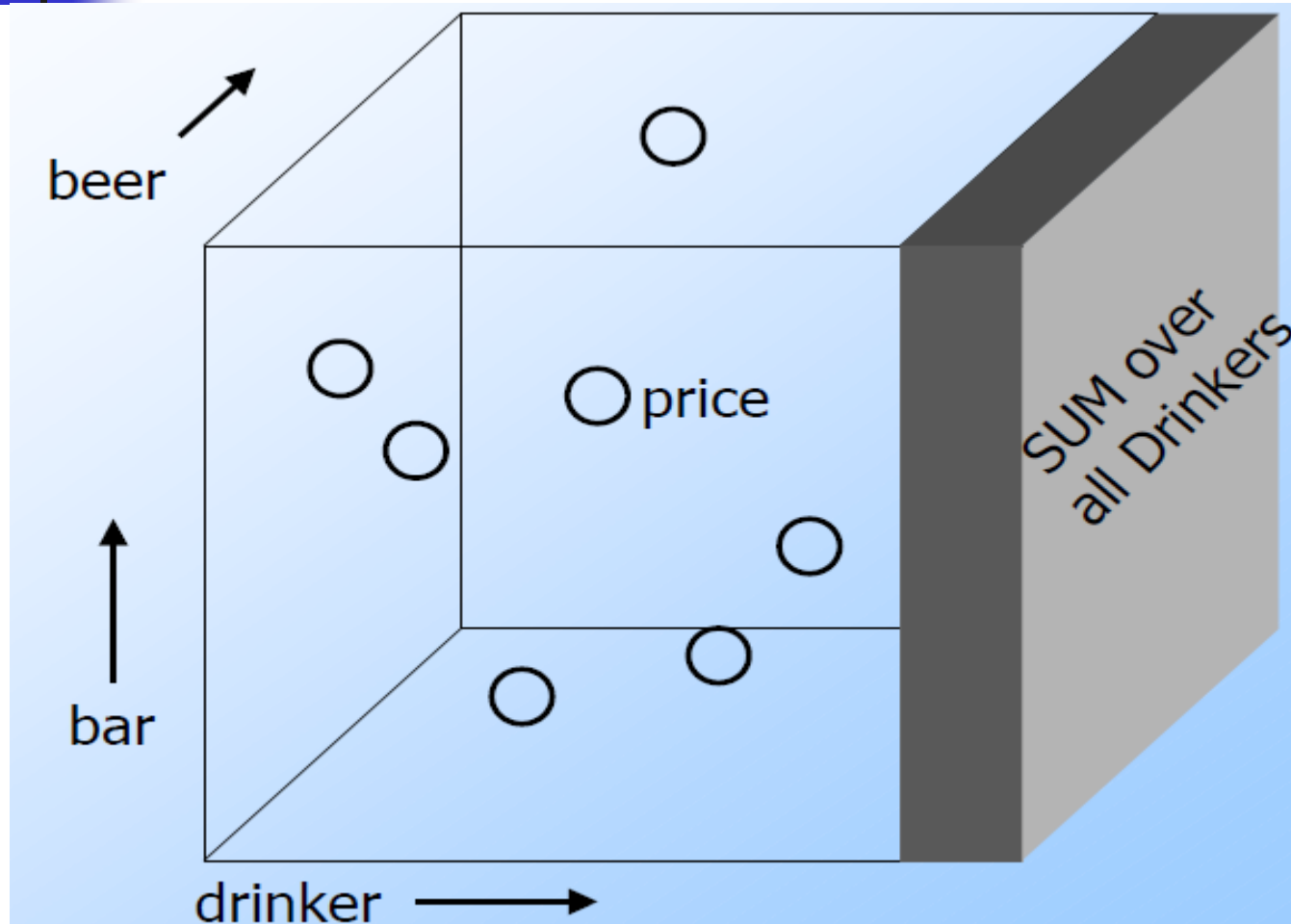


# Marginals

---

- The data cube also includes aggregation (typically SUM) along the margins of the cube.
- The marginals include aggregations over one dimension, two dimensions,...

# Visualization --- Data Cube w/Aggregation



Aggregated  
data on sides,  
edges, corners



# Example: Marginals

---

- Our 4-dimensional Sales cube includes the sum of price over each bar, each beer, each drinker, and each time unit (perhaps days).
- It would also have the sum of price over all bar-beer pairs, all bar-drinker day triples,...



# Structure of the Cube

---

- 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.
- Example: 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

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

\$ means the price charged.





# 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*>;

# examples

\$ 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

23

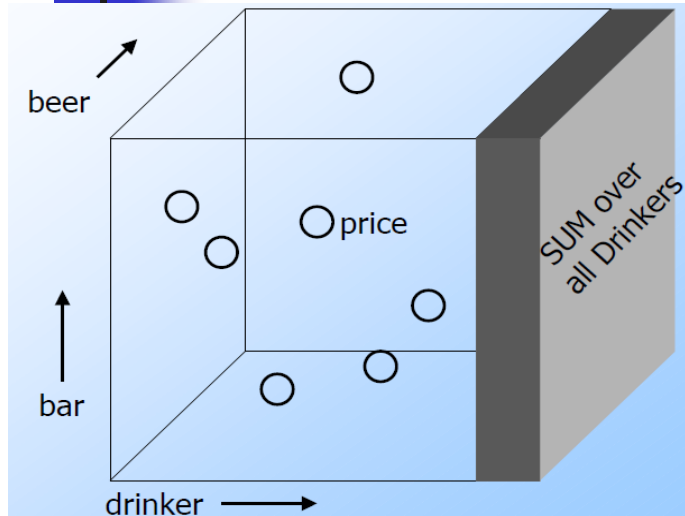
Select drinker, sum(price)  
From sales natural join beers  
Where manf='Anheuse-Busch'  
Group by drinker;



Dices by drinker, then slices by beer' manf.

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

# The cube operator in SQL



- We 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;**

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

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



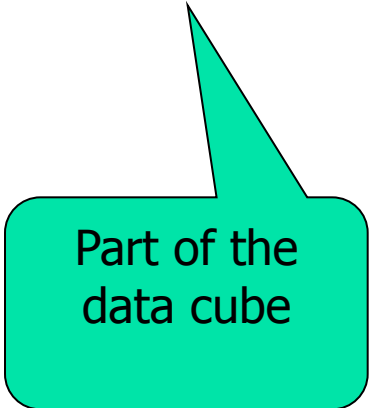
# The cube operator in SQL

---

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

# Extended Aggregation in SQL:1999

- 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 cube(drinker, bar, beer)
```

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.



## Extended Aggregation (Cont.)

---

- 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 rollup(drinker, bar, beer)
```

Generates union of four groupings:

$\{ (drinker, bar, beer), (drinker, bar), (drinker), ( ) \}$

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.



# Chapter Summary

---

- Privileges & Grant diagrams
- SQL Recursive Queries
- Object-relational model
- UDT: how to define UDT? What are the uses of UDT (as type of a relation, type of an attribute )? How to access components of a UDT?
- OLAP: star schemas, cube operator, rollup, drill-down