

# Data Warehousing and Business Intelligence

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo, Professor  
Department of Computer Science  
Purdue University Fort Wayne

# References

---

- W. Lemanhieu, et al., Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data, Ch 17, Ch18.1.2.1
- ❖ Jeffrey et al., Modern Database Management (Ch 9)

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ❑ Data Warehouse Modeling
- ❑ ETL Process
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

# Decision Making Levels

---

- ❑ **Operational decision making level** where
  - Day-to-day business decisions are made
  - Typically in real-time or a short time frame
  - On-Line Transaction Processing (OLTP) systems
- ❑ **Tactical decision making level** where
  - Decisions are made by middle management
  - With a medium-term focus (e.g., a month, a quarter, a year)
- ❑ **Strategic decision making level** where
  - Decision are made by senior management
  - With long-term implications (e.g., 1, 2, 5 years, or more)

# Operational System

---

- **Operational system** – a system that is used to run a business (in real time,) based on current data
  - Operational level
  - Focus on simple INSERT, UPDATE, DELETE and/or SELECT statements
  - Transaction throughput

## vs. Decision Support Systems

---

- **Informational system** (also referred as **Decision support system (DSS)**)
  - Needs at both the tactical and strategic level
  - Provides information to support decisions either the medium or long term
  - Focus on data retrieval by answering complex ad-hoc queries in a user-friendly way
    - Represent data in a multidimensional way
    - Support various levels of data aggregation or summarization
    - Provide interactive facilities for advanced data analysis

# Transforming Operational Data into Decision Support Data

Operational Data

	A	B	C	D	E
3	Year	Region	Agent	Product	Value
4	2004	East	Carlos	Erasers	50
5	2004	East	Tere	Erasers	12
6	2004	North	Carlos	Widgets	120
7	2004	North	Tere	Widgets	100
8	2004	North	Carlos	Widgets	30
9	2004	South	Victor	Balls	145
10	2004	South	Victor	Balls	34
11	2004	South	Victor	Balls	80
12	2004	West	Mary	Pencils	89
13	2004	West	Mary	Pencils	56
14	2005	East	Carlos	Pencils	45
15	2005	East	Victor	Balls	55
16	2005	North	Mary	Pencils	60
17	2005	North	Victor	Erasers	20
18	2005	South	Carlos	Widgets	30
19	2005	South	Mary	Widgets	75
20	2005	South	Mary	Widgets	50
21	2005	South	Tere	Balls	70
22	2005	South	Tere	Erasers	90
23	2005	West	Carlos	Widgets	25
24	2005	West	Tere	Balls	100

Operational data have a narrow timespan, low granularity, and single focus. Such data are usually presented in tabular format, in which each row represents a single transaction. This format often makes it difficult to derive useful information.

Decision Support Data

	A	B	C	D	E	F
1	Year	2005				
2						
3	Sum of Value	Region				
4	Product	East	North	South	West	Total
5	Balls	55		70	100	225
6	Erasers		20	9		110
7	Pencils	45	60			105
8	Widgets			15	25	180
9	Total	100	80	15	125	620
10						
11	Year	(All)				
12	Product	(All)				
13						
14	Sum of Value	Region				
15	Agent	East	North	South	West	Total
16	Carlos	95	150	30	25	300
17	Mary		60	25	145	330
18	Tere	12	100	160	100	372
19	Victor	55	20	259		334
20						
21	Total	162	330	574	270	1,336

Decision support system (DSS) data focus on a broader timespan, tend to have high levels of granularity, and can be examined in multiple dimensions. For example, note these possible aggregations:

- Sales by product, region, agent, etc.
- Sales for all years or only a few selected years.
- Sales for all products or only a few selected products.

# OLTP (Online Transaction Processing)

---

- ❑ Traditional applications for operational system is called **OLTP(Online Transaction Processing)** applications.
  - For maintaining data from organization's everyday operations, e.g., order entry and bank transactions.
  - Up-to-date data
  - Frequent updates (insert, delete) and simple predefined queries.
  - Applications for managing such operational data is called OLTP applications.
  - Well supported by traditional DMBSs.



# OLAP (Online Analytical Processing)

---

## ❑ **OLAP (Online Analytical Processing) :**

Advanced data analysis environment that supports decision making, business modeling and operations research

- For analyzing current and historical data, and identifying useful trends and creating summaries of data
  - ❑ More important than detailed, individual records.
- Hundreds of gigabytes to terabytes in data size
- Query intensive with mostly ad hoc, complex queries that can access millions of records
- Query throughput and response times are more important than transaction throughput
- Such applications are called OLAP applications

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ☞ **Data Warehouse**
- ❑ Data Warehouse Modeling
- ❑ ETL Process
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

# Data Warehouse Definition

---

- “A **data warehouse** is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management’s decision-making processes”  
(*Inmon & Hackathorn, 1994*)

# Properties of Data Warehouse

---

- *Subject-oriented* implies that
  - data are organized around subjects such as customers, products, sales, etc.
  - focusing on the subjects rather than on the applications or transactions,
  - The data warehouse is optimized to facilitate the analysis of the decision-makers by leaving out any data that are not relevant to the decision-making process
- *Integrated* implies that
  - The data warehouse integrates data from a variety of operational sources and a variety of formats to make one version of “the truth” – ensures that all data are named, transformed, and represented in a similar way

# Properties of Data Warehouse (cont.)

---

- ❑ ***Non-volatile (Non-updatable)*** implies that
  - Data are primarily read-only,
  - Data will not be frequently updated or deleted over time
  - Due to this property, data is often non-normalized /aggregated to speed up the analyses, and transaction management and concurrency control are less of a concern
- ❑ ***Time variant*** implies that
  - The data warehouse essentially stores a time series of periodic snapshots
  - Data are not updated or deleted as the business state changes, but new data are added, reflecting this new state
  - Also stores state information about the past, called historical data. Every piece of data stored in the data warehouse is accomplished by a time identifier.

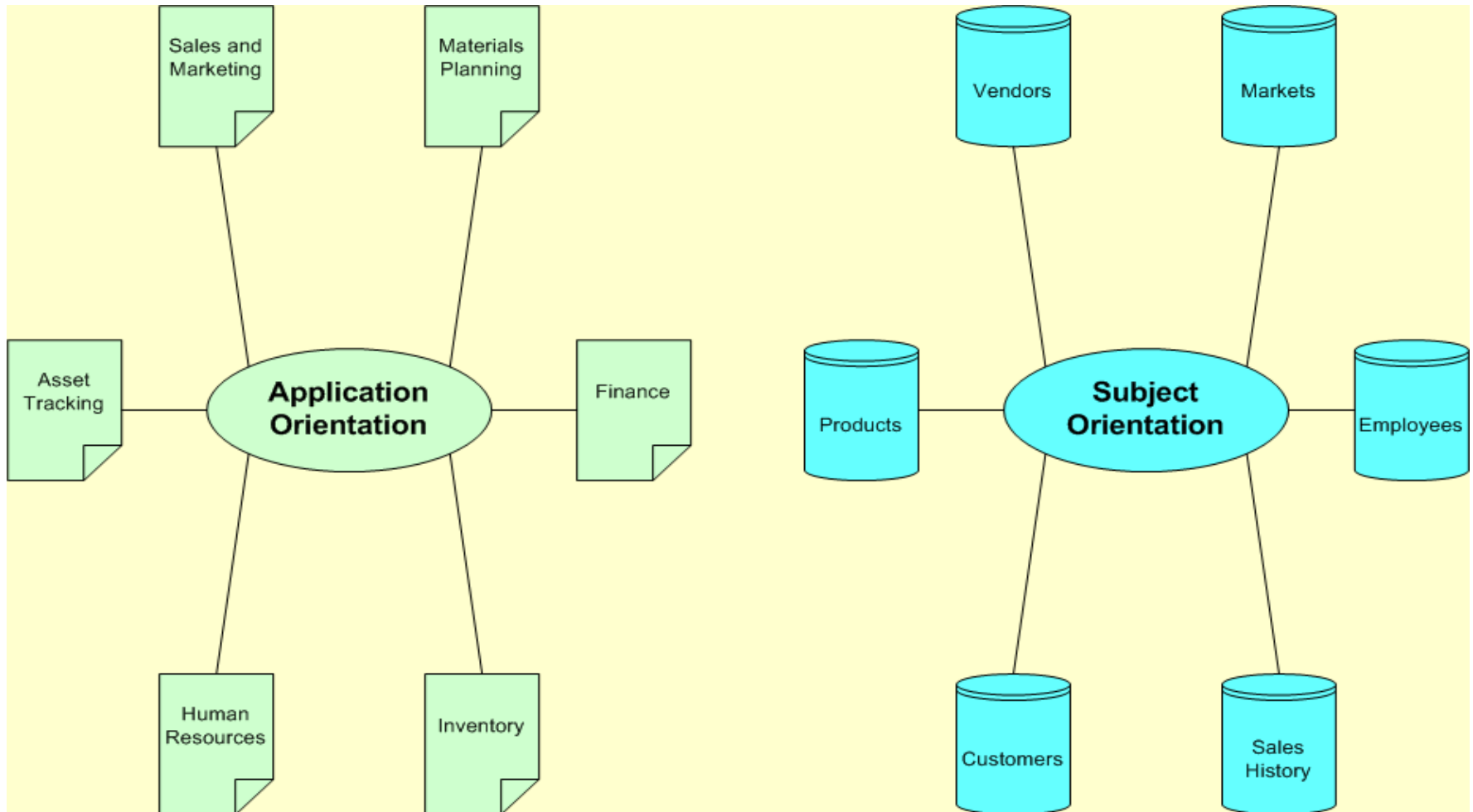
# Operational Database vs. Data Warehouse Characteristics

---

CHARACTERISTIC	OPERATIONAL DATABASE DATA	DATA WAREHOUSE DATA
Integrated	Similar data can have different representations or meanings. For example, Social Security numbers may be stored as ###-##-#### or as #####, and a given condition may be labeled as T/F or 0/1 or Y/N. A sales value may be shown in thousands or in millions.	Provide a unified view of all data elements with a common definition and representation for all business units.
Subject-oriented	Data are stored with a functional, or process, orientation. For example, data may be stored for invoices, payments, and credit amounts.	Data are stored with a subject orientation that facilitates multiple views of the data and facilitates decision making. For example, sales may be recorded by product, by division, by manager, or by region.
Time-variant	Data are recorded as current transactions. For example, the sales data may be the sale of a product on a given date, such as \$342.78 on 12-MAY-2004.	Data are recorded with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.
Nonvolatile	Data updates are frequent and common. For example, an inventory amount changes with each sale. Therefore, the data environment is fluid.	Data cannot be changed. Data are added only periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.

# Application Oriented Operational DB vs Subject Oriented DW

---



Reference: Marakas, Modern Data Warehousing, Mining, and Visualization: Core Concepts

# Decision Support System and Data Warehouse

---

- ❑ A new type of comprehensive data storage facility is needed to implement a DSS
- ❑ A **data warehouse** provides this centralized, consolidated data platform by integrating data from different sources and in different formats.
- ❑ The data warehouse provides a separate and dedicated environment **for both tactical and strategic decision-making.**
- ❑ The data warehouse focus on providing the master data for doing advanced analyses such as OLAP and analytics.



# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ☞ **Data Warehouse Modeling**
- ❑ ETL Process
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

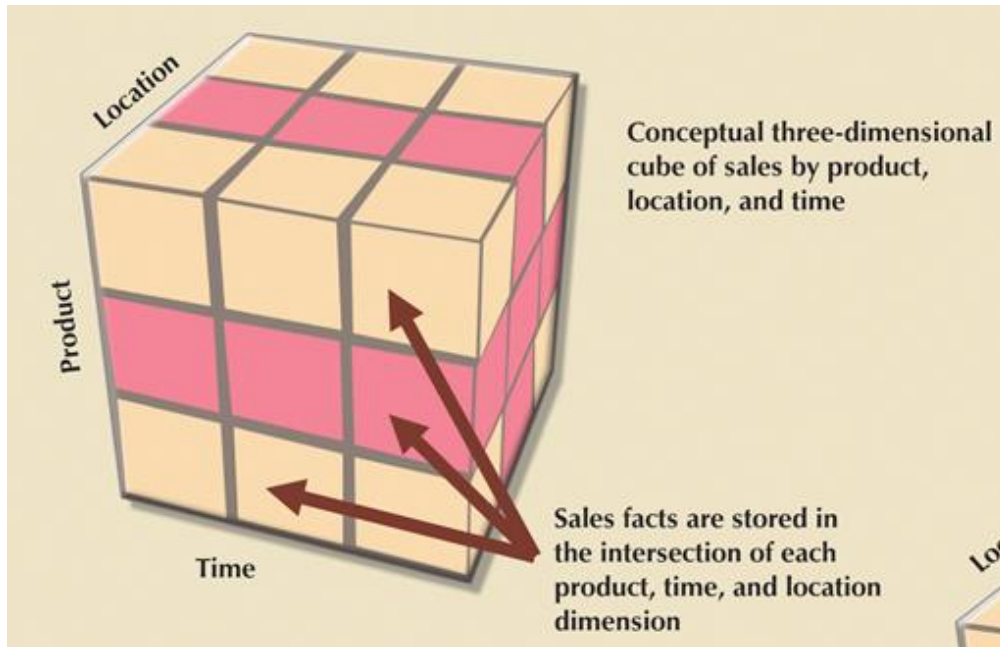
# Data Warehouse Modeling

---

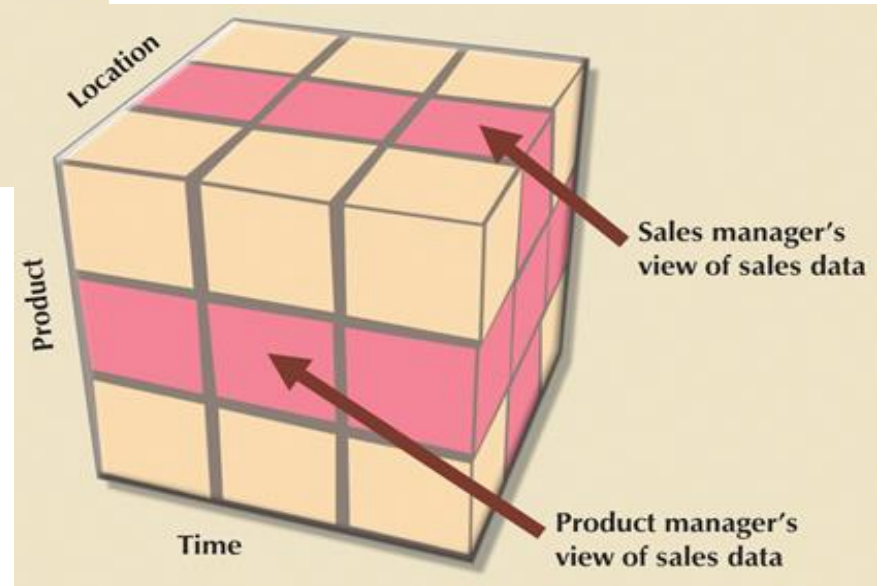
- ❑ Most data warehouses use a “**dimensional modeling**” approach [*Raph Kimball*].
- ❑ In **dimensional modeling**, data are processed and viewed as part of a *multidimensional structure* particularly attractive to business decision makers
- ❑ Augmented by following functions:
  - Advanced data presentation functions
  - Advanced data aggregation, consolidation and classification functions
  - Advanced computational functions
  - Advanced data modeling functions

# Example: Three Dimensional View of Sales

---



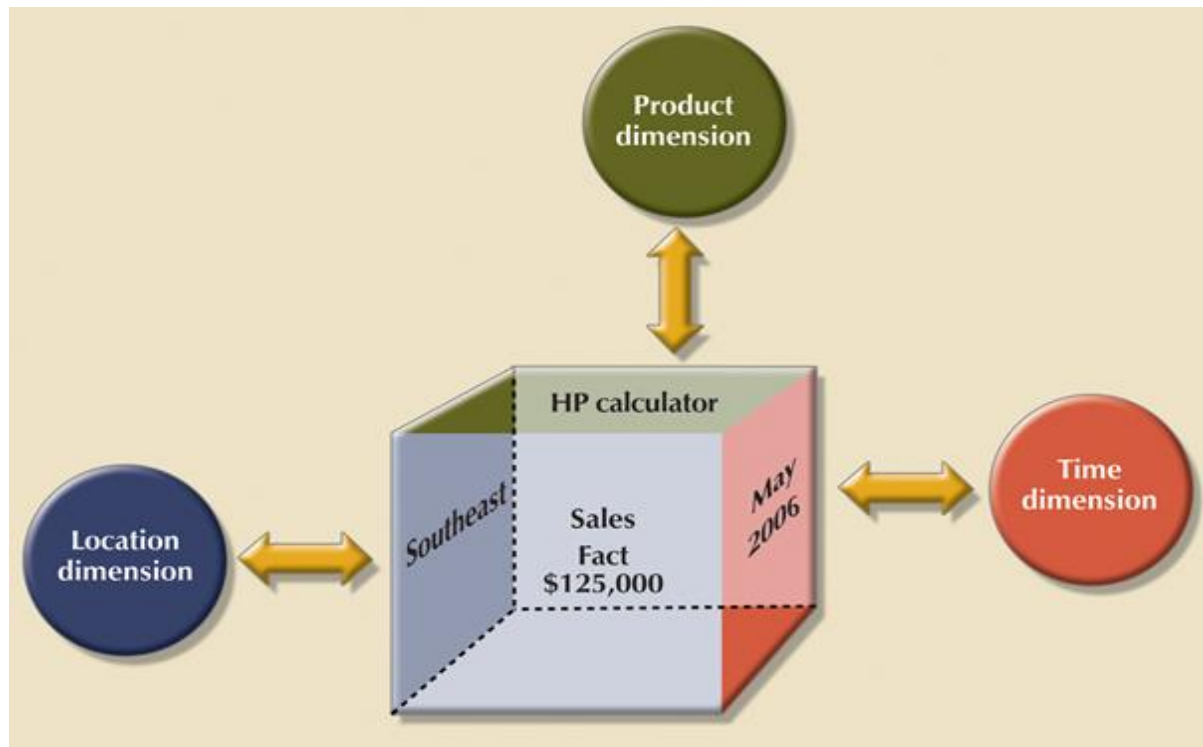
Reference: Coronel and Morris, Database Systems



# Multi-dimensional Schema

---

- ❑ **Multi-dimensional schema** divide data into **facts** and **dimensions**



# Facts

---

- ❑ **Facts** are typically *numeric measurements (values)* that represent specific business aspects or activity
- ❑ Facts normally stored in **fact tables**.
- ❑ One tuple per transaction or event (i.e., a fact) and also measurement or observation data (e.g., sales, budget, revenue, inventory )

# Dimensions

---

- ❑ **Dimensions** provide further information about each of the facts in the fact table (e.g., Time, Location, Customer, Product)
- ❑ Dimensions provide descriptive characteristics about the facts through their attributes
- ❑ Dimensions can be criteria for aggregating the measurement data (i.e., fact)
- ❑ A dimension can have a structure, often composed of one or more hierarchies, that categorizes data. However dimension tables are often denormalized

## Dimensions (cont.)

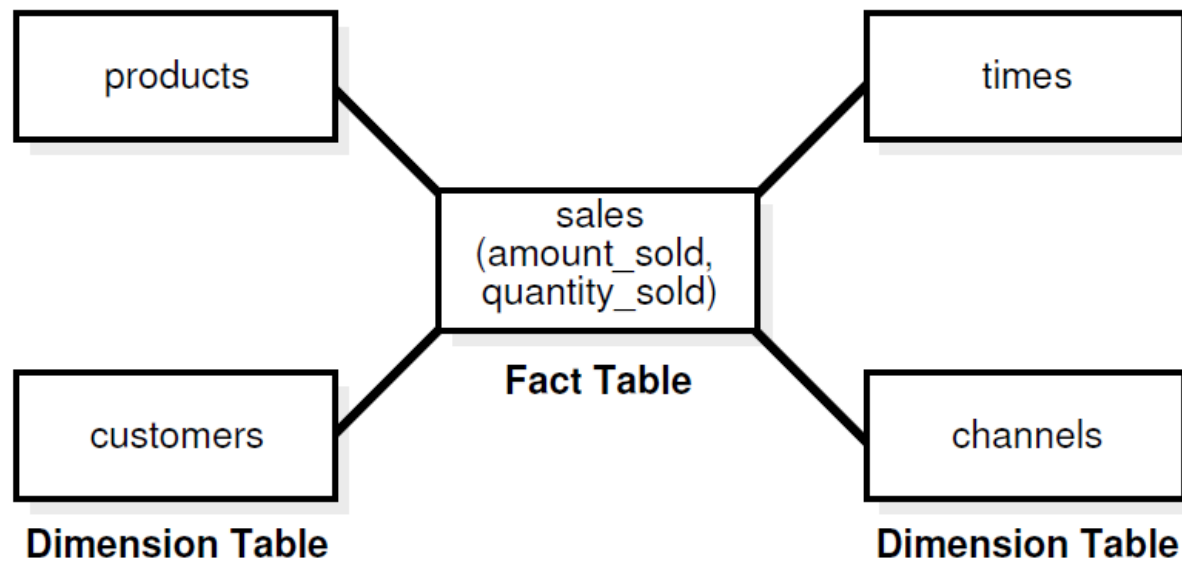
---

- ❑ Dimension data is stored in **dimension tables**.
- ❑ The dimension tables contain the criteria for aggregating the measurement data, and will thus be used as constraints to answer queries such as
  - What are the maximum sales during a particular quarter across all products, stores, and customers?
  - What are the average sales per customer across all time period, stores, and products?
  - What is the minimum number of units sold in store XYZ during Quarter 2 across all products?

# Fact Table

---

- A **fact table** contains facts that are linked through their **dimension tables**, which provide the context for the measure.



**Figure.** A sales fact table. The dimensions associated with a sale amount can be the city, product name, and date when the sale was made.



## Fact Table (cont.)

---

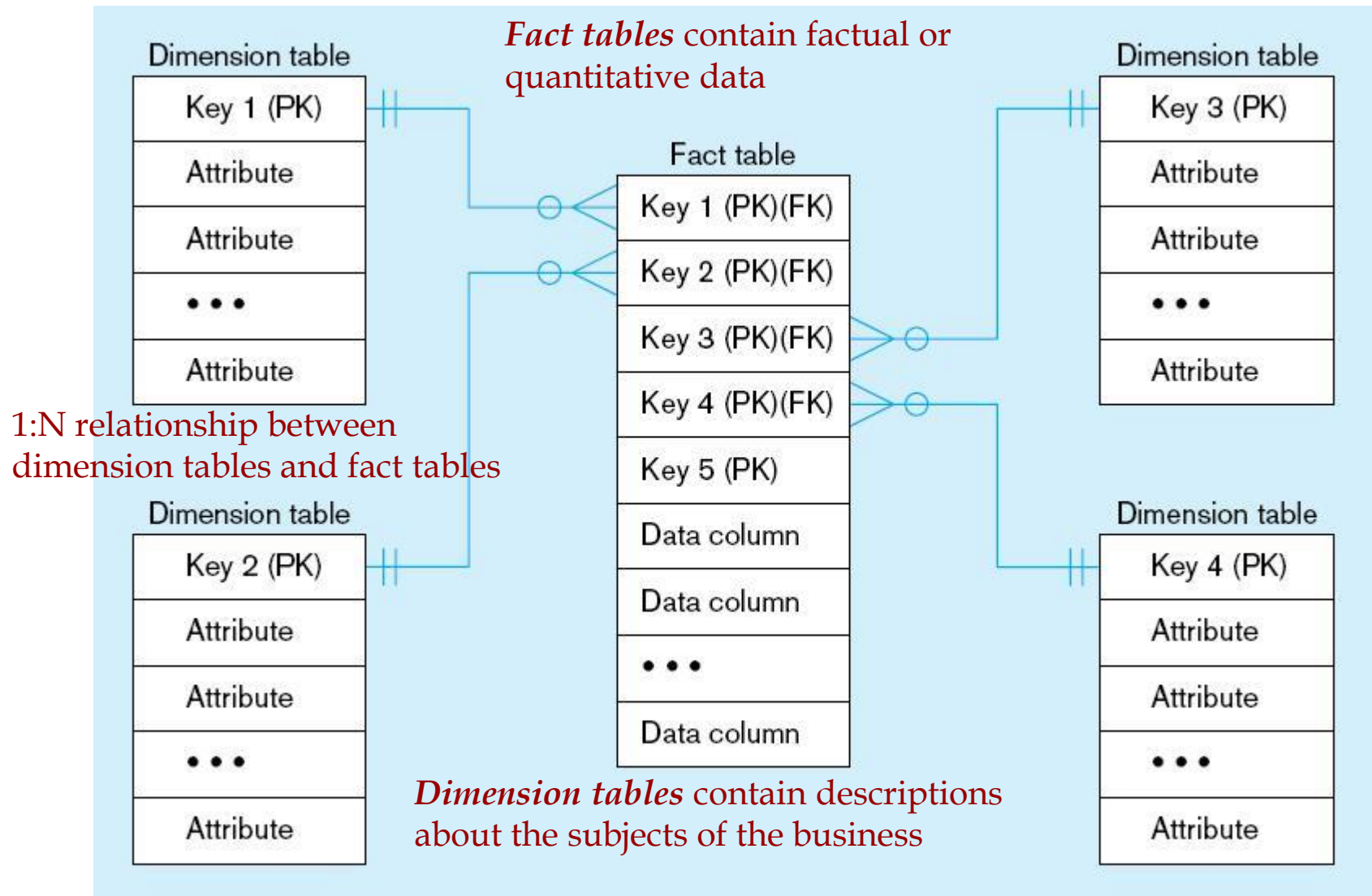
- ❑ A fact table contains either detail-level facts or facts that have been aggregated.
  - A fact table usually contain facts with the same level of aggregation
  - Fact tables that contain aggregated facts are often called **summary tables**.
- ❑ Fact tables have many rows but typically not many columns
- ❑ A fact table has a composite key made of primary keys of the dimension tables of the schema

# Dimension Table

---

- ❑ **Dimension tables** act as lookup or reference tables, and provide category data to give context to the fact data
- ❑ They are usually textual and descriptive.
  - You will use their values as the row headers, column headers and page headers of the reports generated by your queries.
- ❑ While dimension tables have far fewer rows than fact tables, they can be quite wide, with dozens of columns.
- ❑ The values in many dimension tables may change infrequently

# Key Constraints in Fact and Dimension Tables



# Attributes

---

- ❑ Each dimension is described by a set of **attributes of the dimension**.
- ❑ Examples:
  - Product dimension may consist of four attributes: the category, the industry of the product, year of its introduction, and average profit margin.
  - Location dimension may be anything provides a description of the location. Possible attributes for the location dimension is region, state, city, store and so on.

# Dimension Hierarchies

---

- A dimension table might have columns indicating every level of its rollup hierarchy, and may show multiple hierarchies reflected in the table.
- **Dimension hierarchies** provide the hierarchy information of a dimensional table.

## PRODUCT

category

pname

## TIME

year

quarter

week

month

date

## LOCATION

region

state

city

store

## Dimension Hierarchies (Cont.)

---

- ❑ Within a hierarchy, each level is logically connected to the levels above and below it.
- ❑ Data values at lower levels aggregate into the data values at higher levels.
- ❑ They provide capability to perform drill-down and roll-up searches in a data warehouse. Can be used to define data aggregation.
- ❑ A dimension can be composed of more than one hierarchy.
  - E.g., in the product dimension, there might be two hierarchies—one for product categories and one for product suppliers.

### LOCATION

region



state

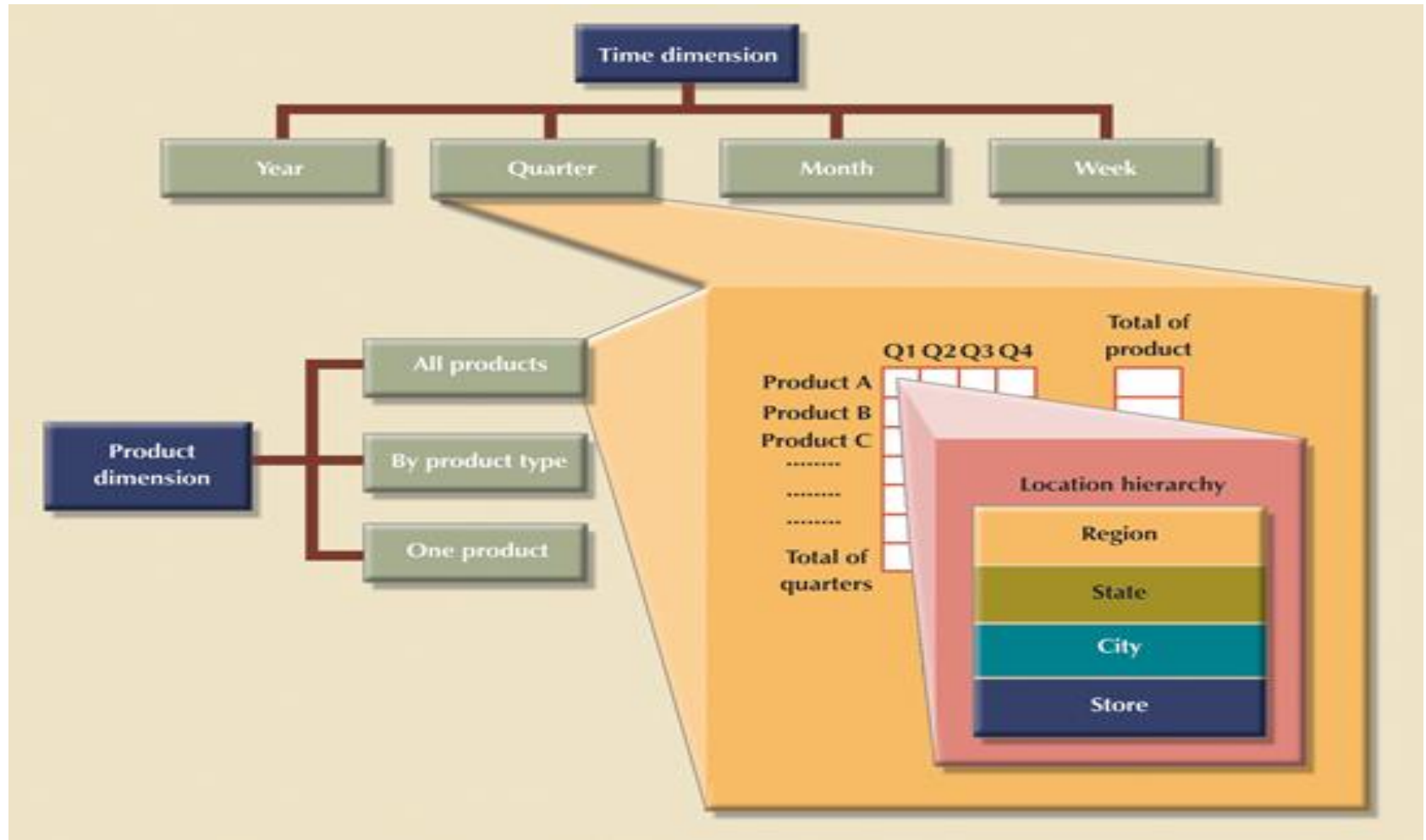


city



store

# Example of Dimension Hierarchies



# Data Warehouse Schema

---

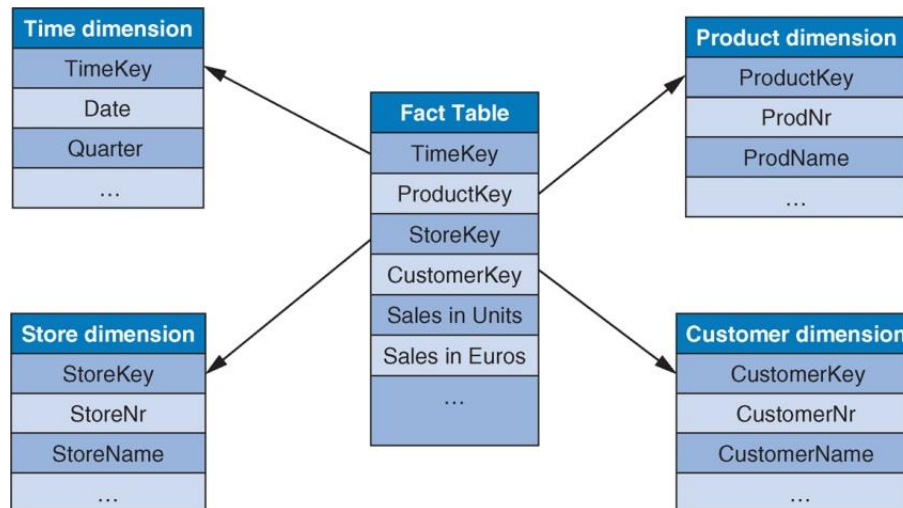
- ❑ Various conceptual data models or schema can be adopted to design a data warehouse, which all involve the modeling of facts, as well as dimensions with which to analyze these facts.
  - **Star schema**
  - **Snowflake schema**
  - **Galaxy schema** (a collection of star schema, also known **fact constellation schema**)
  - A mixture of these approaches (e.g., a schema combined with a denormalized star schema and a normalized snowflake schema)



# Start Schemas

---

- A **star schema** has one large central fact table that is connected to various smaller dimension tables.
- The fact table has multiple foreign keys referring to each of the dimension tables, implementing a 1:N relationship type.
- The primary key of the fact table consists of the composition of all these foreign keys



# Example Tuples of the Star Schema

---

**Fact table**

TimeKey	ProductKey	StoreKey	CustomerKey	Sales in Units	Sales in Euros
200	50	100	20006010	6	167.94
210	25	130	20006012	3	54
180	30	150	20006008	12	384
...					

**Dimension tables**

TimeKey	Date	Quarter	...
200	08/03/2017	1	...
210	09/11/2017	3	
180	27/02/2017	1	

CustomerKey	CustomerNr	CustomerName	...
20006008	20	Bart Baesens	...
20006010	10	Wilfried Lemahieu	
20006012	5	Seppe vanden Broucke	

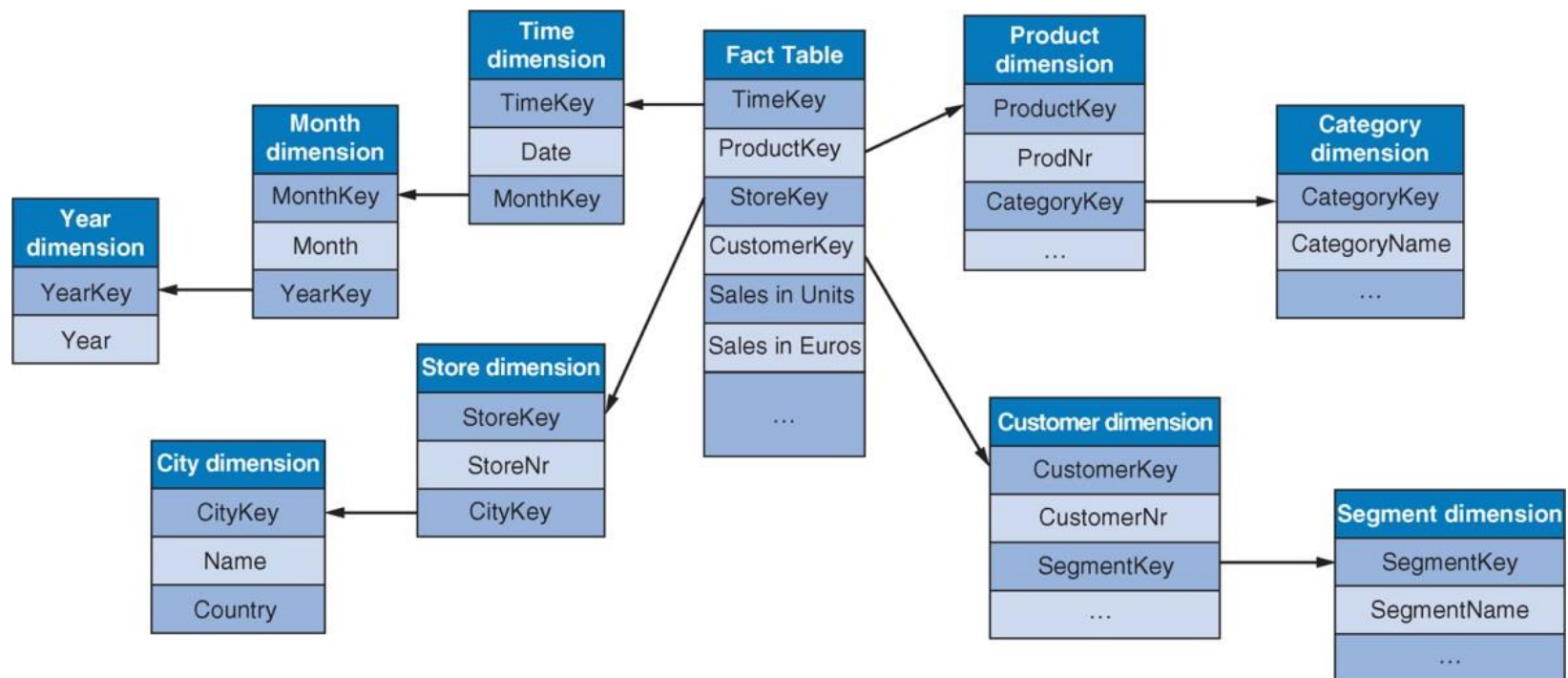
StoreKey	StoreNr	StoreName	...
100	68	The Wine Depot	...
130	94	The Wine Crate	
150	69	Vinos del Mundo	

ProductKey	ProdNr	ProdName	...
25	0178	Meerdael, Methode Traditionnelle Chardonnay, 2014	...
30	0199	Jacques Selosse, Brut Initial, 2012	
50	0212	Billecart-Salmon, Brut Réserve, 2014	

# Snowflake Schema

---

- ❑ A **snowflake schema** normalizes the dimension tables.
- ❑ It essentially decomposes the hierarchical structure of each dimension.



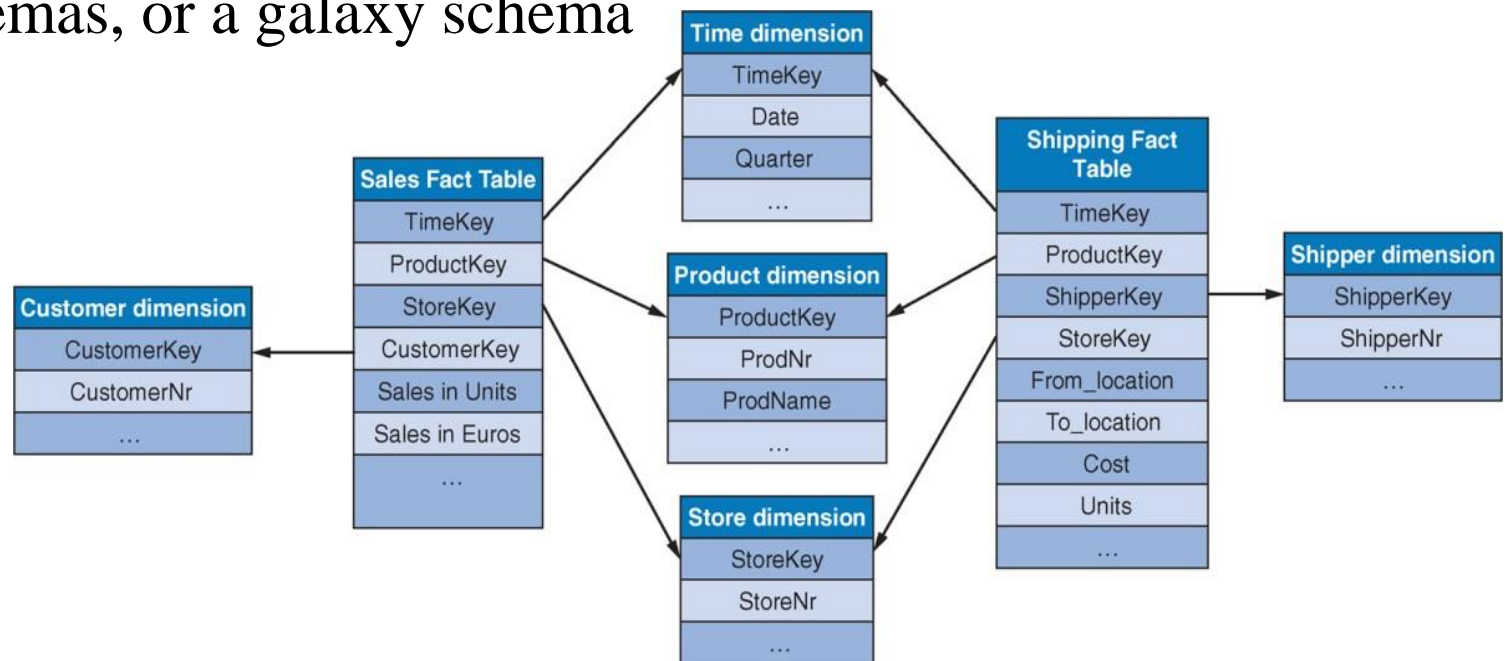
## Snowflake Schema (cont.)

---

- ❑ Snowflake schema requires more joins
- ❑ If the dimension tables grow too large and a more efficient usage of storage capacity is required, the snowflake approach is considered.
- ❑ It may also be beneficial if most queries don't make use of the outer-level dimension tables.

# Galaxy Schema (Fact Constellation Schema)

- ❑ A **fact constellation schema** (also known as **galaxy schema**) has more than one fact table.
- ❑ This schema is also referred to as a collection of star schemas, or a galaxy schema



- Sales fact Table and Shipping Fact table share the tables Time dimension, Product dimension and Store dimension

# DW Schema Issues

---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- ❑ Factless fact tables
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- ❑ Slowly changing dimensions
- ❑ Rapidly changing dimensions

# Business Value Columns as Keys

---

- ❑ Business value columns might not be good for keys to join the fact table with dimensional tables since
  - They usually have a business meaning in OLTP  
e.g., SSN number for Employee, VAT number for Company
  - They are tied to the business setting and requirements. If they are changed, all tables using these keys would need to be changed
  - They are usually bigger in size, which will result in big indexes and consequently slow down query execution time.
  - They are also often re-used over longer periods of time  
e.g., a product number “123abc” may be a different product now than five years ago.

# Surrogate keys

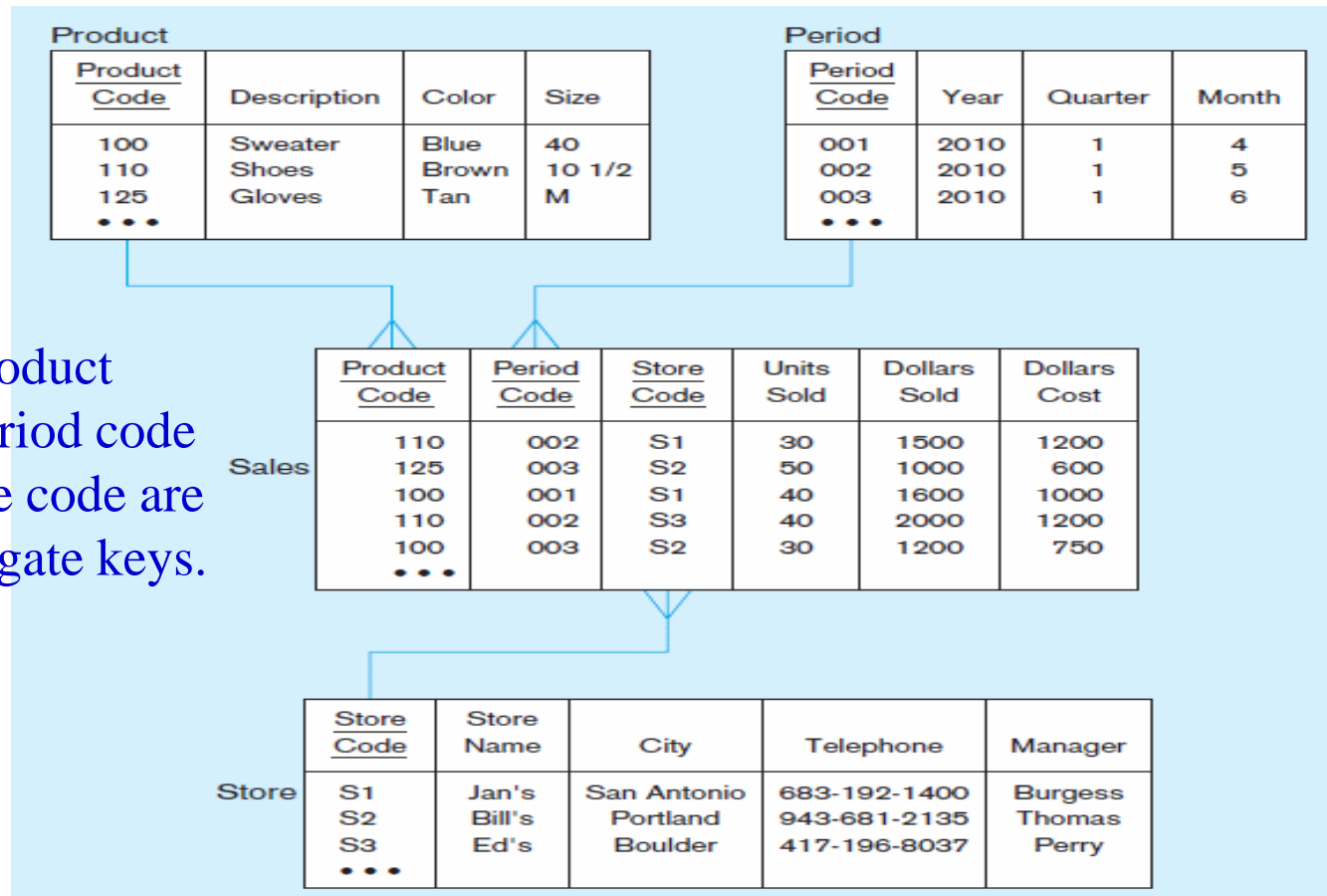
---

- ❑ **Surrogate or artificial keys**, are recommended to be used to join the fact table with dimension tables.
- ❑ Meaningless integers ( usually sequential integers) used to connect the fact to the dimension tables
  - StoreKey, ProductKey, ShipperKey, ...
- ❑ Surrogate keys can be of the same length and format for all keys, no matter what business dimensions are involved in the database, even dates.
- ❑ A surrogate key allows us to handle changing and unknown business keys with ease.



# Surrogate Key Example

Here, product code, period code and store code are all surrogate keys.



- Every key used to join is a simple **surrogate key**
- Using surrogate keys will save space (for fact table) and improve (query) performance.

# DW Schema Issues

---

- ❑ Surrogate keys
- **Granularity of the fact table**
- ❑ Factless fact tables
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- ❑ Slowly changing dimensions
- ❑ Rapidly changing dimensions

# Data Grain

---

- ❑ One of the most important tasks when designing your model is to consider the level of details a model will provide, referred to as **data grain**
- ❑ For example, consider a sale schema
  - Will the grain be very fine, storing every single item purchased by each customer? Or
  - Will it be a coarse grain, storing only the daily totals of sales for each product at each store?
- ❑ In modern data warehousing
  - there is a strong emphasis on providing the finest grain data possible, because this allows for maximum analytic power.

# Granularity of the fact table

---

- ❑ **Granularity** is a level of detail of one row of the fact table
- ❑ Higher (lower) granularity implies more (fewer) rows
- ❑ Trade-off between level of detailed analysis and storage requirements
- ❑ Examples:
  - One tuple of the fact table corresponds to one (item) line on a purchase order
  - One tuple of the fact table corresponds to one purchase order
  - One tuple of the fact table corresponds to all purchase orders made by a customer
- ❑ Recommend that each fact stores just one grain level.

# DW Schema Issues

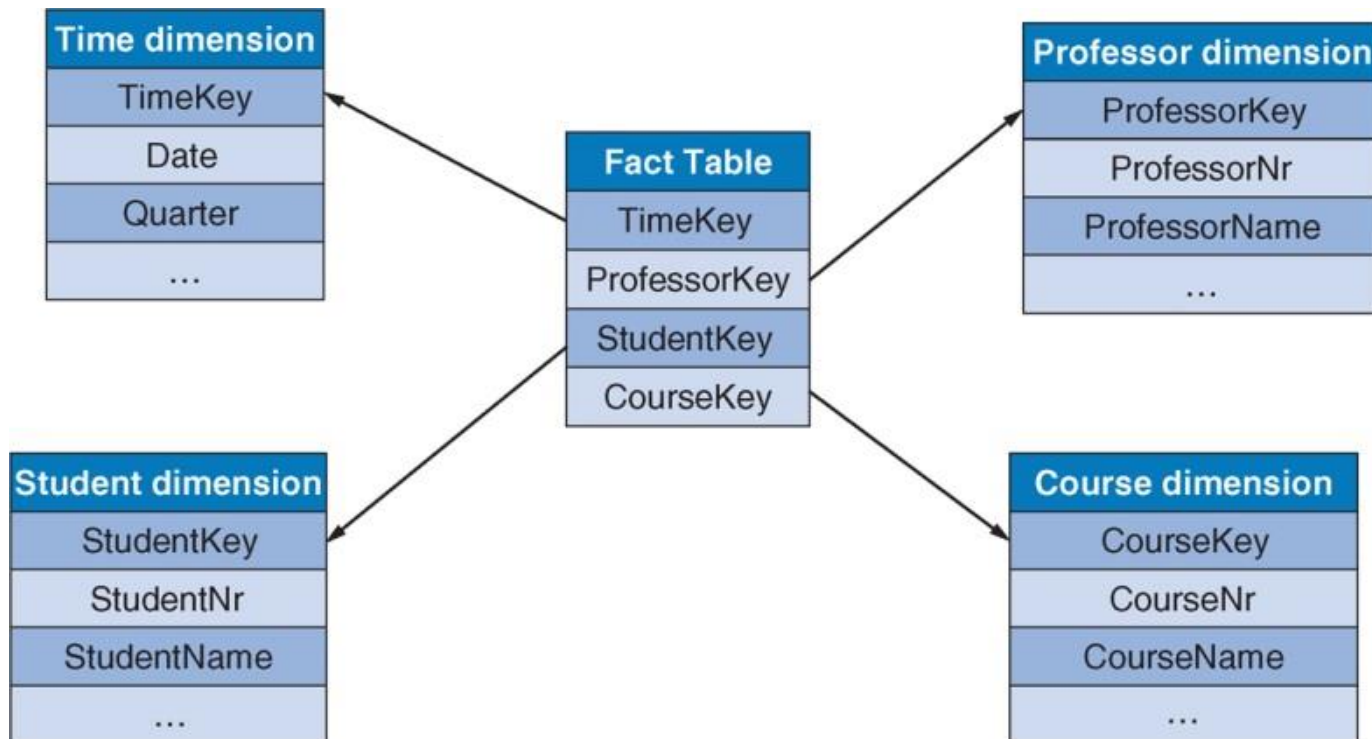
---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- **Factless fact tables**
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- ❑ Slowly changing dimensions
- ❑ Rapidly changing dimensions

# Factless Fact Tables

---

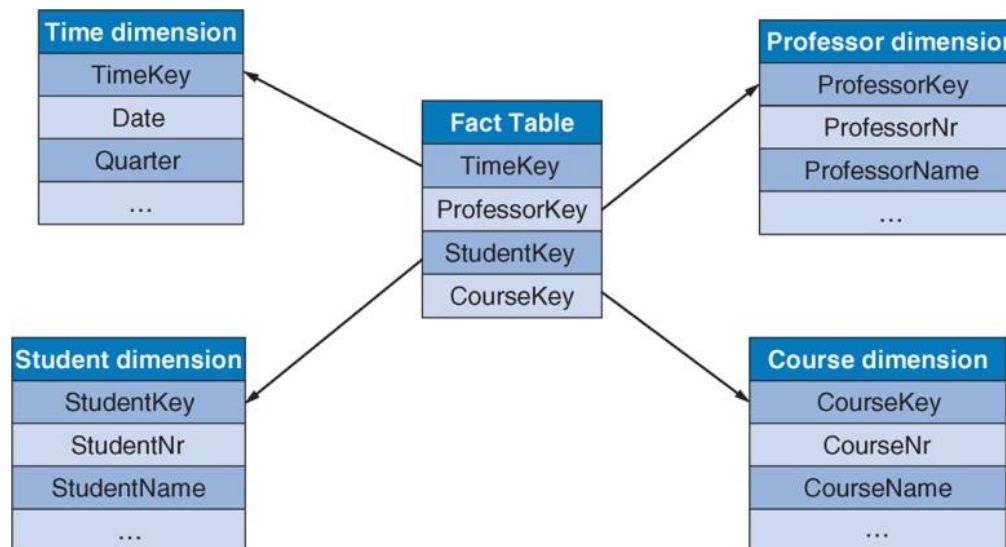
- A **factless fact table** is a fact table that contains only foreign keys and no measurement data.



# Usages of Factless Fact Tables

---

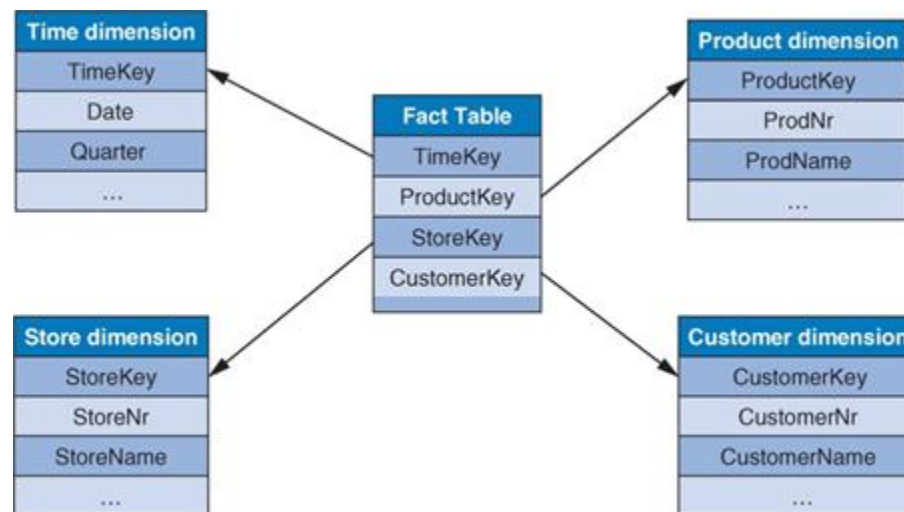
- ❑ Less common, but it can be used to track events and allows you to answer questions such as
  - Which professor teaches the highest number of courses?
  - What is the average number of students that attend a course?
  - Which course has the maximum number of students?



# Usages of Factless Fact Tables (cont.)

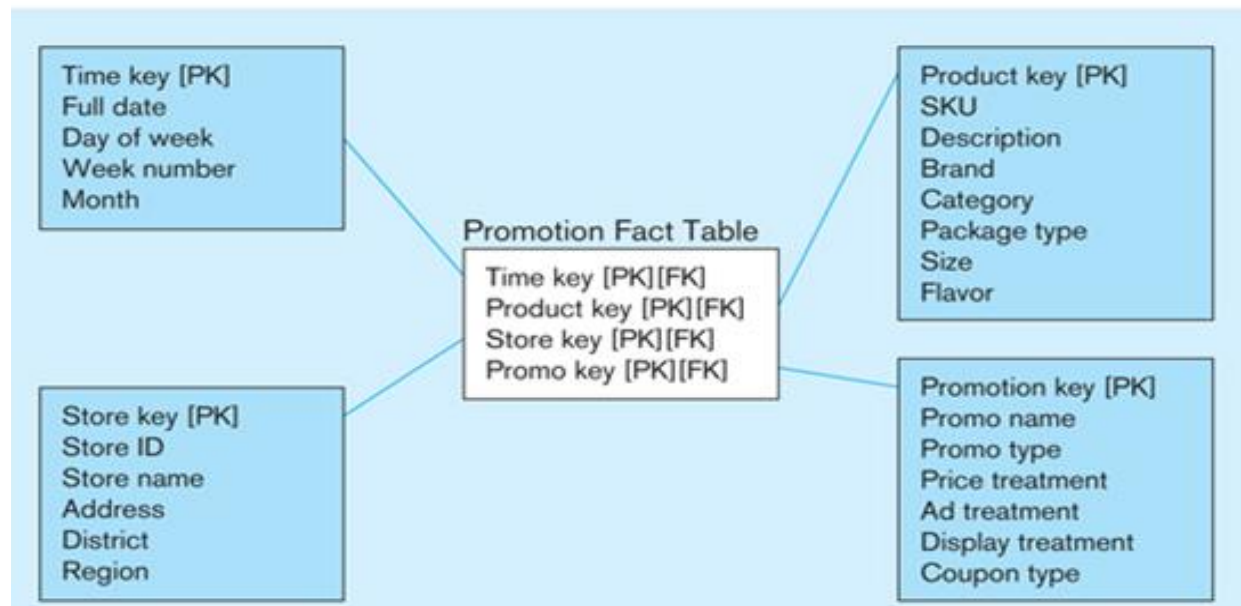
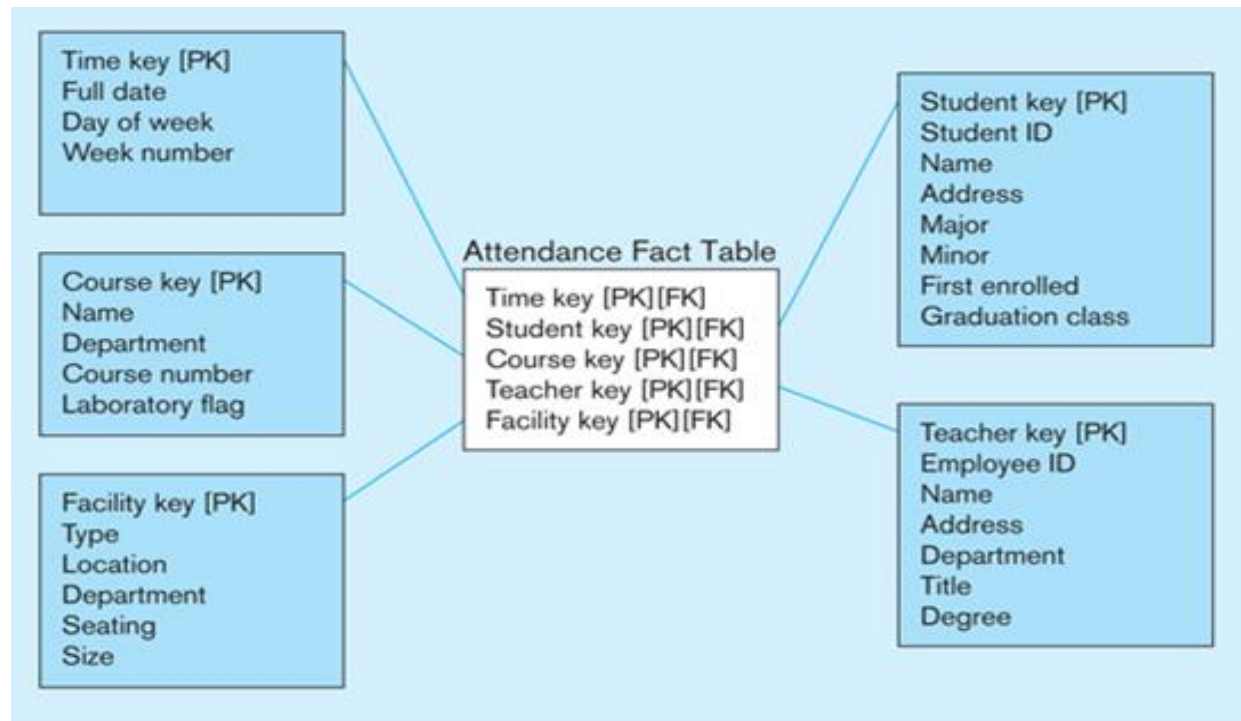
---

- A factless fact table is also used for analyzing coverage or negative reporting such as
  - What is the average number of products sold by a store?
  - Which customers did not purchase any products?
  - Which stores did not sell any products during a particular period?





# Examples of factless fact table



# DW Schema Issues

---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- ❑ Factless fact tables
- **Optimizing the dimension tables**
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- ❑ Slowly changing dimensions
- ❑ Rapidly changing dimensions

# Optimizing the Dimension Tables

---

- ❑ In a dimensional table, the number of columns can get quite large.
- ❑ e.g., Time dimension
  - TimeKey, Date, DayOfWeek, DayOfMonth, DayOfYear, Month, MonthName, Year, LastDayInWeekFlag, LastDayInMonthFlag, FiscalWeek, HolidayFlag, ...
- ❑ On average, the number of dimensional tables is between 5 and 10
- ❑ To improve query execution time, **the dimension tables should be heavily indexed**

Time dimension
TimeKey
Date
DayOfWeek
DayOfMonth
DayOfYear
Month
MonthName
Year
LastDayInWeekFlag
LastDayInMonthFlag
FiscalWeek
HolidayFlag
WeekendFlag
Quarter
Season
...

# Attributes with Low Cardinality

---

- How to deal with low cardinality attribute types such as flags or indicators
  - E.g., On-line Purchase (Y/N), Payment (cash or credit-card), Discount (Y/N)
- Method to implement attributes with low cardinality
  - Add these attributes directly to the fact table, or
  - Create a separate dimension per each one, or
  - Combine all the attributes into a **junk dimension**

# Junk Dimensions

---

- **Junk dimension** is a dimension that simply enumerates all feasible combinations of values of the low cardinality attribute types

JunkKey1	On-line purchase	Payment	Discount
1	Yes	Credit-card	Yes
2	Yes	Credit-card	No
3	No	Credit-card	Yes
4	No	Credit-card	No
5	Yes	Cash	Yes
6	Yes	Cash	No
7	No	Cash	Yes
8	No	Cash	No

$2^3 = 8$  combinations

- Greatly contribute to the maintainability and query performance of the data warehouse environment.

# DW Schema Issues

---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- ❑ Factless fact tables
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- **Defining outrigger tables**
- ❑ Slowly changing dimensions
- ❑ Rapidly changing dimensions

# Example

---

- Suppose a Customer dimension table that also includes demographic data obtained from an external data provider, such as average household size, unemployment rate, percentage female population, percentage population, percentage homeownership, etc.
- It may include many redundant data in different tuples

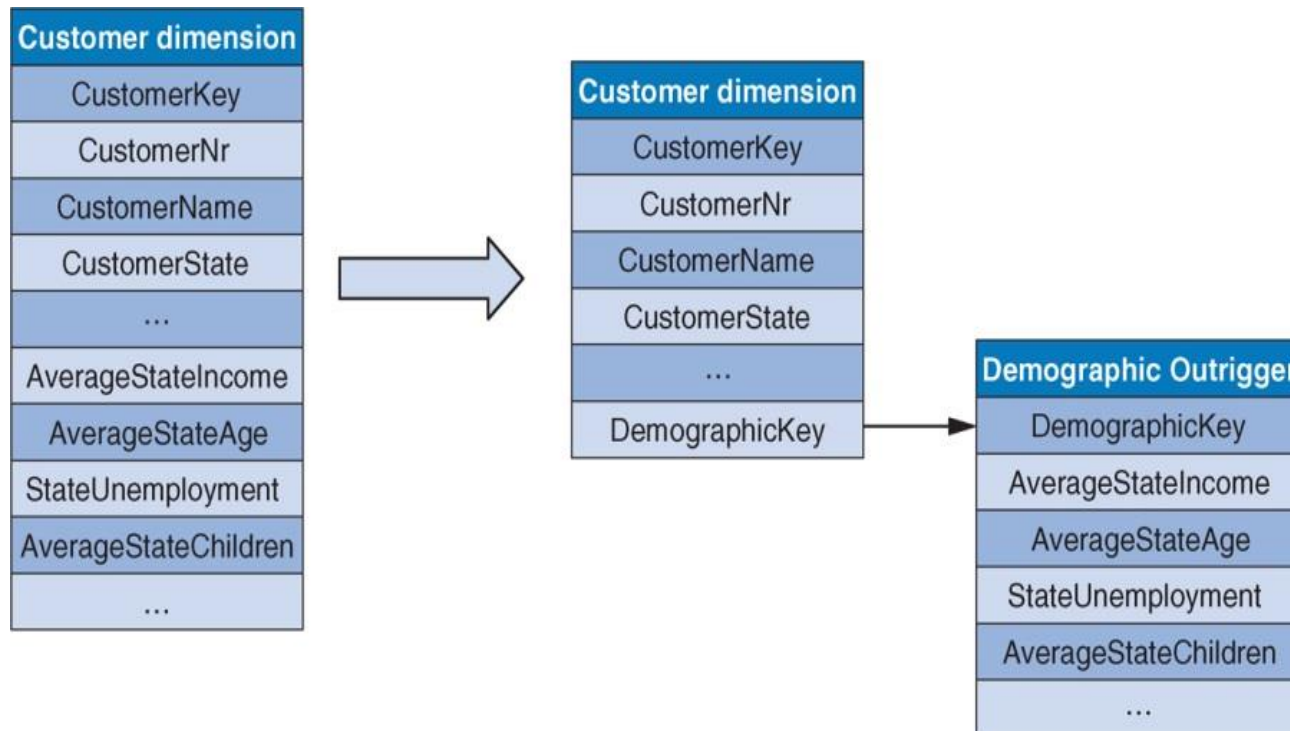
Customer dimension
CustomerKey
CustomerNr
CustomerName
CustomerState
...
AverageStateIncome
AverageStateAge
StateUnemployment
AverageStateChildren
...

} redundant  
data over  
tuples

# Outtrigger tables

---

- An **outrigger table** can be defined to store a set of attribute types of a dimension table which are highly correlated, low in cardinality and updated simultaneously





# DW Schema Issues

---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- ❑ Factless fact tables
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- **Slowly changing dimensions**
- ❑ Rapidly changing dimensions

# Slowly Changing Dimensions

---

- A **slowly changing dimension** is a dimensions that change slowly and irregularly over a period of time
- **Example:** customer segment ranging from AAA, AA, A, BBB, ... to C, determined on a yearly basis

# Approaches for Slowly Changing Dimensions

---

- ❑ Suppose to update a customer from AA to AAA
- ❑ Ways to accommodate a slow (e.g., yearly) change in a data warehouse environment
  - **Approach 1 : Simply overwrite**

## OLD STATUS

CustomerKey	CustomerNr	CustomerName	Segment
123456	ABC123	Bart Baesens	AA

## NEW STATUS

CustomerKey	CustomerNr	CustomerName	Segment
123456	ABC123	Bart Baesens	AAA

- 
- A loss of old data
  - Not good for historical recording

# Approaches for Slowly Changing Dimensions

## □ Approach 2: store all historical data values

### OLD STATUS

CustomerKey	CustomerNr	CustomerName	Segment	Start_Date	End_Date	Current_Flag
123456	ABC123	Bart Baesens	AA	27-02-2014	31-12-9999	Y

### NEW STATUS

CustomerKey	CustomerNr	CustomerName	Segment	Start_Date	End_Date	Current_Flag
123456	ABC123	Bart Baesens	AA	27-02-2014	27-02-2015	N
123457	ABC123	Bart Baesens	AAA	28-02-2015	31-12-9999	Y

↑  
Different  
surrogate  
key values

↑  
Same business  
number (customer  
number) key value

↑  
Valid state and end  
dates

↑  
Most recent  
tuple indication

# Approaches for Slowly Changing Dimensions

---

## □ Approach 3 : store partial historical data

### OLD STATUS

CustomerKey	CustomerNr	CustomerName	Segment
123456	ABC123	Bart Baesens	AA

### NEW STATUS

CustomerKey	CustomerNr	CustomerName	Old Segment	New Segment
123456	ABC123	Bart Baesens	AA	AAA

- Store only partial historical information with keeping the most previous value

# Approaches for Slowly Changing Dimensions

---

- **Approach 4** : Maintains two tables. One has the most recent information and the other has the full historical information. Both are linked to the fact table

## **CUSTOMER** (recent data)

CustomerKey	CustomerNr	CustomerName	Segment
123457	ABC123	Bart Baesens	AAA

## **CUSTOMER HISOTRY** (full historical data)

CustomerKey	CustomerNr	CustomerName	Segment	Start_Date	End_Date
123456	ABC123	Bart Baesens	AA	27-02-2014	27-02-2015
123457	ABC123	Bart Baesens	AAA	28-02-2015	31-12-9999

# DW Schema Issues

---

- ❑ Surrogate keys
- ❑ Granularity of the fact table
- ❑ Factless fact tables
- ❑ Optimizing the dimension tables
- ❑ Defining junk dimensions
- ❑ Defining outrigger tables
- ❑ Slowly changing dimensions
- **Rapidly changing dimensions**

# Rapidly Changing Dimensions

---

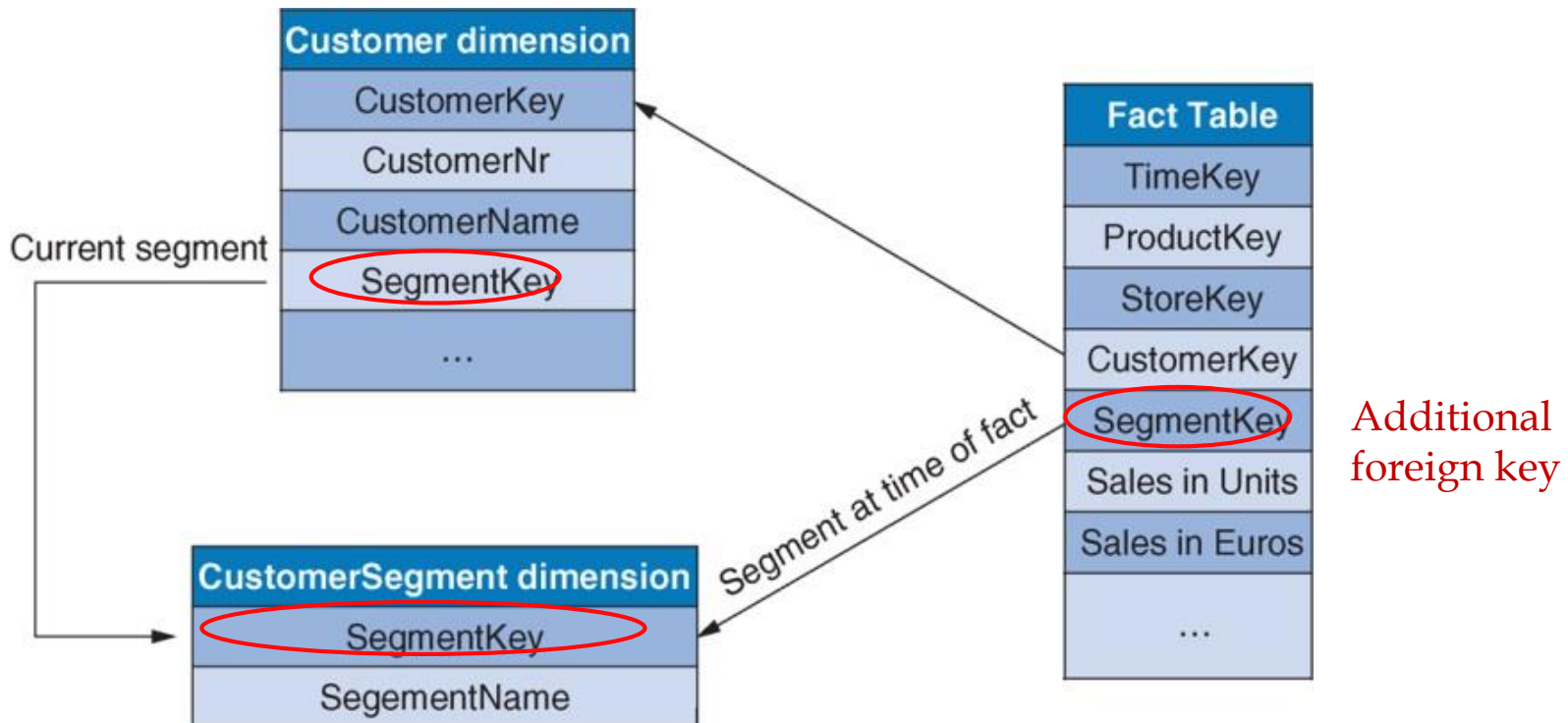
- ❑ **Dimensions that change rapidly and regularly** over a period of time
- ❑ Example: customer segment ranging from AAA, AA, A, BBB, ... to C, determined on a daily basis
- ❑ If approaches 2 and 4 for the slowly changing are applied, they will result into a lot of rows
- ❑ **Approach**
  - Split customer information into stable (e.g., gender, marital status, ...) and rapidly changing information (e.g., segment) which is put in a so-called **mini-dimension table** (e.g., CustomerSegment)



# Approaches for Rapidly Changing Dimensions

## □ Approach 1

The rapidly changing dimension was the "SegmentKey" in the original Customer Dimension



The Customer Segment Dimension is used to track changes in the customer segment over time.

# Approaches for Rapidly Changing Dimensions

## □ Example Tuples for Approach 1

Fact table

← The fact table store the historical information about the volatile customer data.

TimeKey	ProductKey	StoreKey	CustomerKey	SegmentKey	Sales in Units	Sales in Euros
200	50	100	1200	1	6	167.94
210	25	130	1400	4	3	54
180	30	150	1000	3	12	384
...						

One of customer segments of Bart

Customer Dimension

CustomerKey	CustomerNr	CustomerName	SegmentKey
1000	20	Bart Baesens	2
1200	10	Wilfried Lemahieu	1
1400	5	Seppe vanden Broucke	1

CustomerSegment Dimension

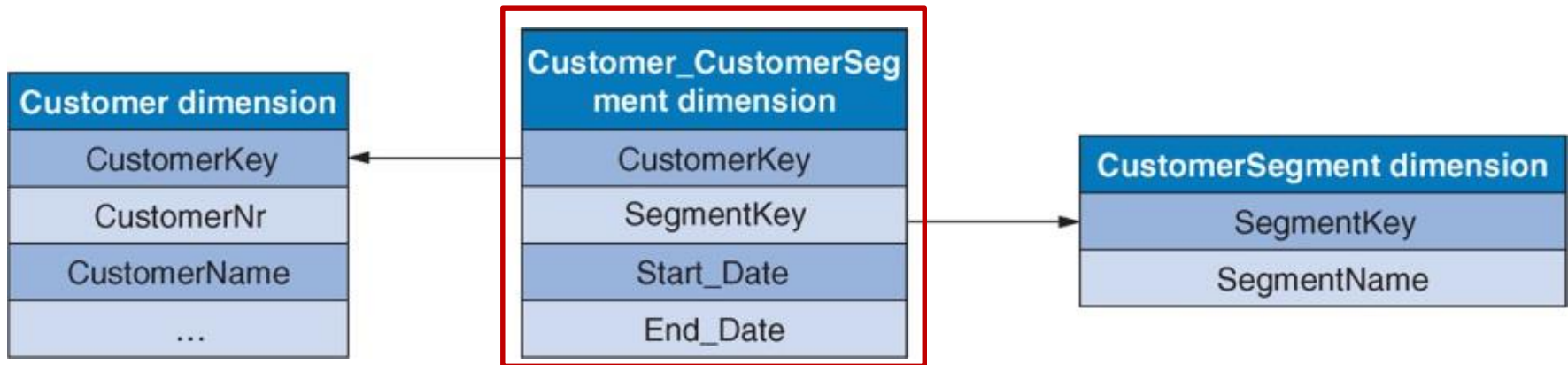
SegmentKey	SegmentName
1	A
2	B
3	C
4	D

The most current customer segment of Bart

# Approaches for Rapidly Changing Dimensions

---

## □ Approach 2



- **Additional table** which includes both surrogate keys in Customer and CustomerSegment tables (with many-to-many relationship)
- Keeps full track of the changing values while minimizing the storage requirements

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ❑ Data Warehouse Modeling
- ☞ **ETL Process**
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

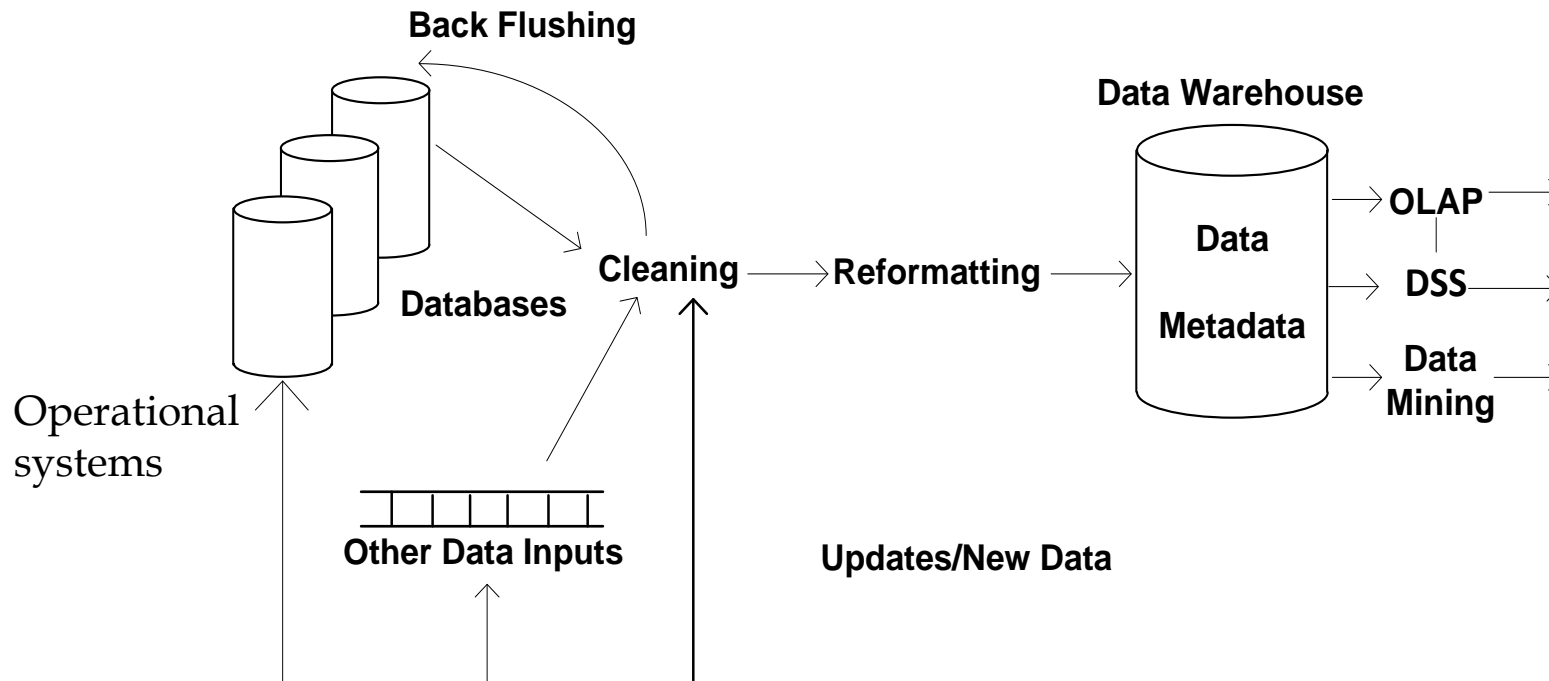
# Data Warehousing

---

- **Data Warehousing** integrates data
  - from many sources into one large repository,
  - spanning long time periods,
  - often augmented with summary information.
- A centralized repository of all the data is called a **data warehouse** (of several gigabytes to terabytes common).

# Conceptual Structure of Data Warehousing

---



- Data are extracted from the various data source systems.
- The data from the various source systems are transformed and integrated
- And then the integrated data is loaded into the data warehouse.
- Data warehouse data is used for OLAP, DSS, etc.
- The results may be fed back to operational system.

# Data Consolidation: Extract, Transform, Load (ETL)

---

- ❑ **Data consolidation** is to capture data from multiple, heterogeneous sources, and integrate into a single persistent store
- ❑ Data consolidation is typically accomplished by **ETL** activities
  - **Extract** data from heterogeneous data sources (including legacy and external sources)
  - **Transform** the data satisfy business needs
  - **Load** the transformed data into a target system (e.g., a data warehouse)

## Data Consolidation: Extract, Transform, Load (ETL)

---

- Once the DW schema has been designed, we can start populating it with data from the operational courses through ETS step (Extract, Transform, and Load)
- ETL step is an iterative process that should be executed at regular points in time (e.g., daily, weekly, monthly)



# Advantages of ETL Processes

---

- ❑ **Scalability:** ETL processes are designed to efficiently handle and transform vast volumes of data, making them indispensable for data warehousing.
- ❑ **Data Enhancement:** They enable complex transformations including data structuring, standardization, cleansing, and aggregation, enhancing the overall value of the data.
- ❑ **Quality Improvement:** ETL tools significantly improve data quality, ensuring data completeness, consistency, and interpretability.
- ❑ **Historical Context:** ETL processes populate data warehouses with both current and historical data, providing a comprehensive view for analytical and reporting purposes.

# Disadvantages of ETL Processes

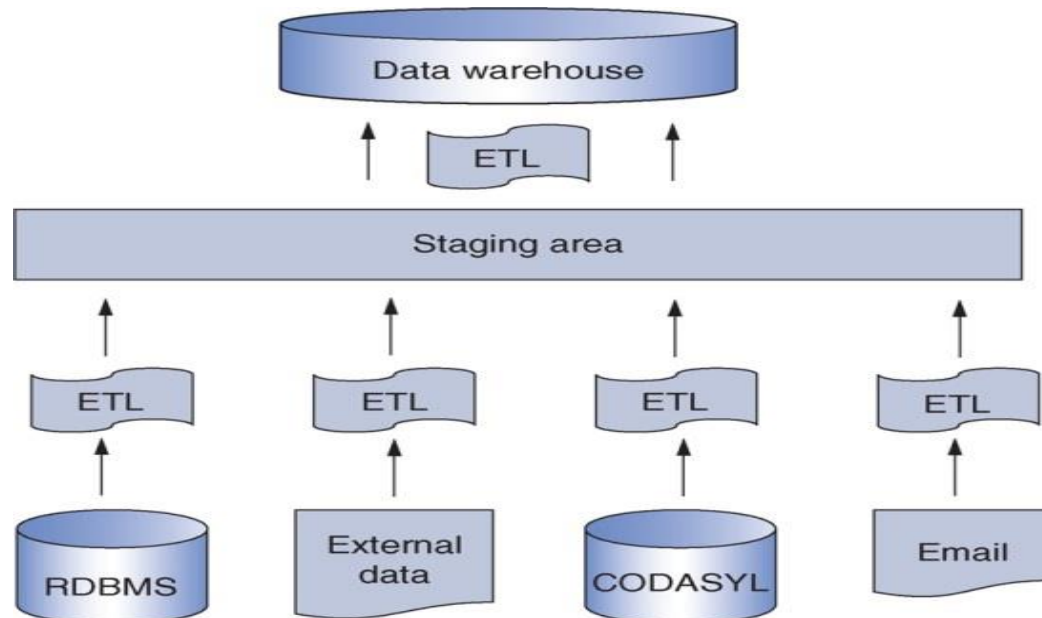
---

- ❑ **Latency:** ETL introduces a delay due to the time needed to extract, transform, and load data into a data warehouse, leading to data being slightly outdated.
- ❑ **Delay Metrics:** The latency period varies from mere seconds to hours or days, depending on the ETL setup and frequency of data refreshes.
- ❑ **Additional Storage Requirements:** ETL processes necessitate extra storage space to handle interim data states during the transformation and loading stages.

# Staging Area: The Backbone of ETL

---

- ❑ ETL step can take up to 80% of all efforts needed to set up a data warehouse
- ❑ Recommended to dump the data in a so-called **staging area** (intermediate storage environment for transformation) where all the ETL activities can be executed



# ETL Process

---

- ❑ Extraction
  - Full or incremental (Changed Data Capture)
- ❑ Transformation
  - formatting
  - cleansing
  - aggregation and merging
  - enrichment
- ❑ Loading
  - Fill the fact and dimension tables
- ❑ Documentation and metadata

# Extraction

---

- **Extraction strategy**
  - **Full**
  - **Incremental**: only the changes since the previous extraction are considered (“changed data capture”)
- It is important to properly accommodate the different types of data sources, operating systems, and hardware environment from where the data are sources

# Key Transformation Activities in ETL Process

---

- ❑ Formatting
- ❑ Cleansing
- ❑ Aggregation and merging
- ❑ Enrichment

# Why Transformation Hard

- Suppose we want to develop a profile for each student, combining all data into a single file format.
- Problems of developing a single corporate view
  - **Inconsistent key structures**
    - E.g., Social security number, student name
  - **Synonyms**
    - E.g., StudentNo, StudentID
  - **Free-form fields vs. structured fields**
    - E.g., StudentName vs (LastName, MI, FirstName)
  - **Inconsistent data values**
    - E.g., Smith's telephone number
  - **Missing data**
    - E.g., Smith's insurance

STUDENT DATA [from the class registration system](#)

<u>StudentNo</u>	LastName	MI	FirstName	Telephone	Status	...
123-45-6789	Enright	T	Mark	483-1967	Soph	
389-21-4062	Smith	R	Elaine	283-4195	Jr	

Located on a UNIX-based server running an Oracle DBMS

STUDENT EMPLOYEE [from the personnel system](#)

<u>StudentID</u>	Address	Dept	Hours	...
123-45-6789	1218 Elk Drive, Phoenix, AZ 91304	Soc	8	
389-21-4062	134 Mesa Road, Tempe, AZ 90142	Math	10	

STUDENT HEALTH [from the a health center system](#)

<u>StudentName</u>	Telephone	Insurance	ID	...
Mark T. Enright	483-1967	Blue Cross	123-45-6789	
Elaine R. Smith	555-7828	?	389-21-4062	

Located on an IBM mainframe running the DB2 DBMS

# Transformation

---

- ❑ **Formatting rules:** specify how data should be consistently and uniformly encoded in the data warehouse.
- ❑ All the different formats should be then mapped to a single one.
  - E.g., male/female, m/f, 0/1 ➔ male/female



# Transformation

---

## □ **Cleaning**

- Get rid of some of the inconsistencies or irregularities in the data, e.g., as birth date, 01/01/1000
- Report any irregularities to the business users to fully understand where they come from and how they should be properly treated.

# Transformation

---

## ❑ **Aggregation ( Deduplication / Merging)**

- Find multiple records referring to the same entity in the operational data sources
  - ❑ They might exist because of the usage of different attribute names (e.g., CustomerID, CustId, Client ID, ID) or data entry mistakes (E.g., Bart Baesens or Bart Baessens)
- Properly aggregate and merge before entering the data in DW

## ❑ **Enrichment**

- The data can be enriched by adding derived data elements or external data
  - ❑ e.g., customer age, based on the data of birth date
  - ❑ e.g., customer demographic information

# Loading

---

- The data warehouse is populated by filling the fact and dimensional tables and also generating the necessary surrogate keys

# Data Refresh

---

- ❑ Propagate updates on source data to correspondingly update the base data and derived data stored in the warehouse.
- ❑ Issues
  - When to refresh
  - How to refresh, e.g., periodically (daily or weekly), or propagate every update
  - Depends on user needs and traffic.
- ❑ Replication servers, e.g.,
  - Oracle Replication Server
  - Sybase Replication Server

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ❑ Data Warehouse Modeling
- ❑ ETL Process
- ☞ **Data Marts**
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

# Data Marts

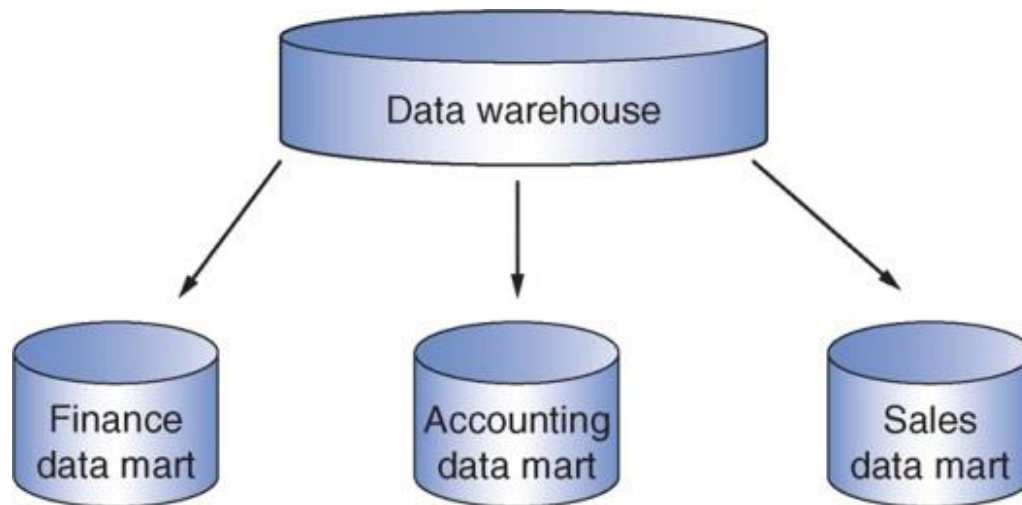
---

- ❑ A **data mart** is a scaled down version of a data warehouse aimed at meeting the information needs of a homogeneous small group of end-users such as a department or business unit (e.g., marketing, finance, logistics, HR, etc.)
- ❑ Reasons to make data marts
  - Provide focused content
    - ❑ such as finance, sales, or accounting information in a format tailored to the user group at hand.
  - Improve query performance by offloading complex queries and workload from other data source (e.g., a data warehouse)
  - Located near to end users, alleviating heavy network load

# Implementation of Data Marts

---

- **Dependent data marts** pull their data directly from a central data warehouse

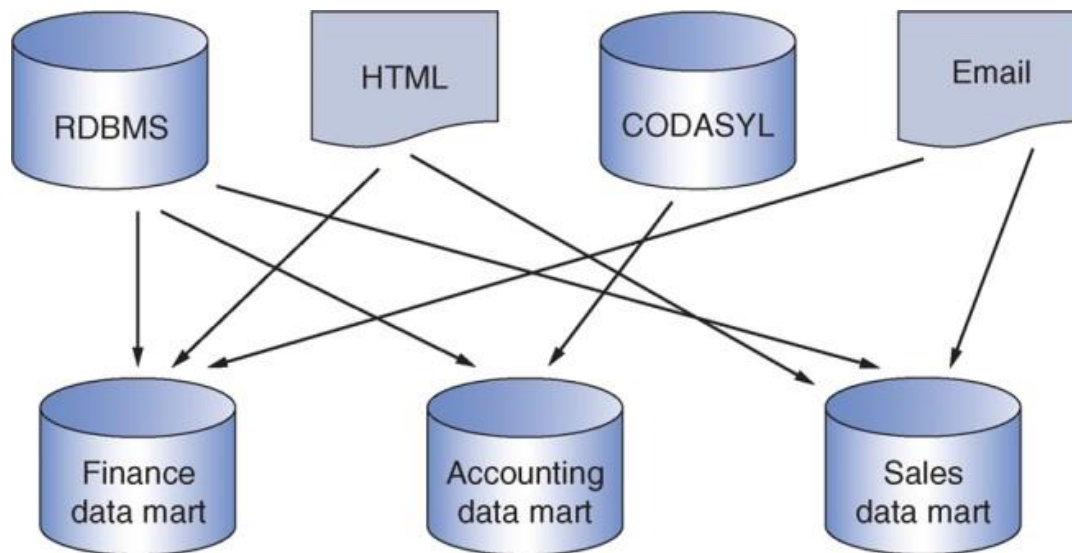


- The central data warehouse has more complex setup, but provide the advantage that they all draw their data form the same formatted, cleansed, etc.

# Implementation of Data Marts

---

- **Independent data marts** are standalone systems drawing data directly from the operational systems, external sources or a combination of both



- Independent data marts might be good for organizations who do not wish to make the substantial investment of a data warehouse.



# Data Warehouse vs Data Mart

---

## Data Warehouse

### Scope

- Application independent
- Centralized, possibly enterprise-wide
- Planned

### Data

- Historical, detailed, and summarized
- Lightly denormalized

### Subjects

- Multiple subjects

### Sources

- Many internal and external sources

### Other Characteristics

- Flexible
- Data oriented
- Long life
- Large
- Single complex structure

## Data Mart

### Scope

- Specific DSS application
- Decentralized by user area
- Organic, possibly not planned

### Data

- Some history, detailed, and summarized
- Highly denormalized

### Subjects

- One central subject of concern to users

### Sources

- Few internal and external sources

### Other Characteristics

- Restrictive
- Project oriented
- Short life
- Start small, becomes large
- Multi, semi-complex structures, together complex

# Outline

---

- ❑ Operational vs Tactical/Strategic Decision Making
- ❑ Data Warehouse Definition
- ❑ Data Warehouse Schemas
- ❑ ETL Process
- ❑ Data Marts
- ☞ **Virtual Data Warehouses and Virtual Data Marts**
- ❑ Data Warehouses vs Data Lakes
- ❑ Business Intelligence

# Virtual Data Warehouses and Virtual Data Marts

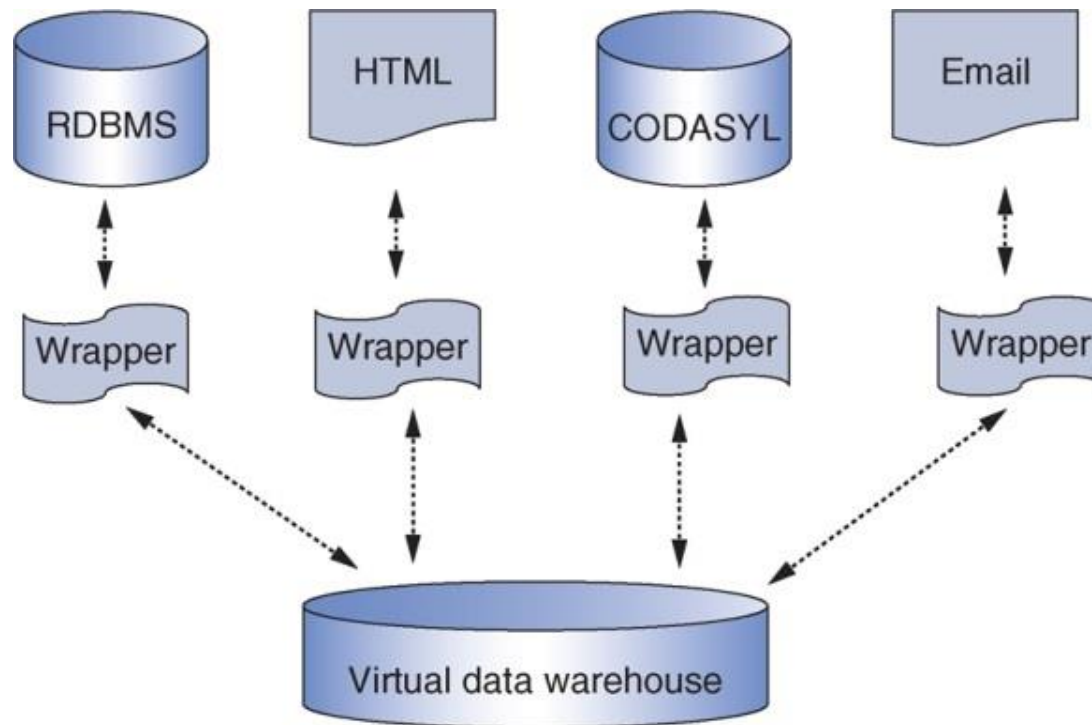
---

- ❑ Idea of virtualization is to use middleware to create a **logical** or **virtual data warehouse** or **virtual data mart** which has no physical data but provides a single point of access to a set of underlying physical data stores.
- ❑ Data is located in their original sources and the data is only accessed ('pulled') at query time.

# Virtual Data Warehouses

---

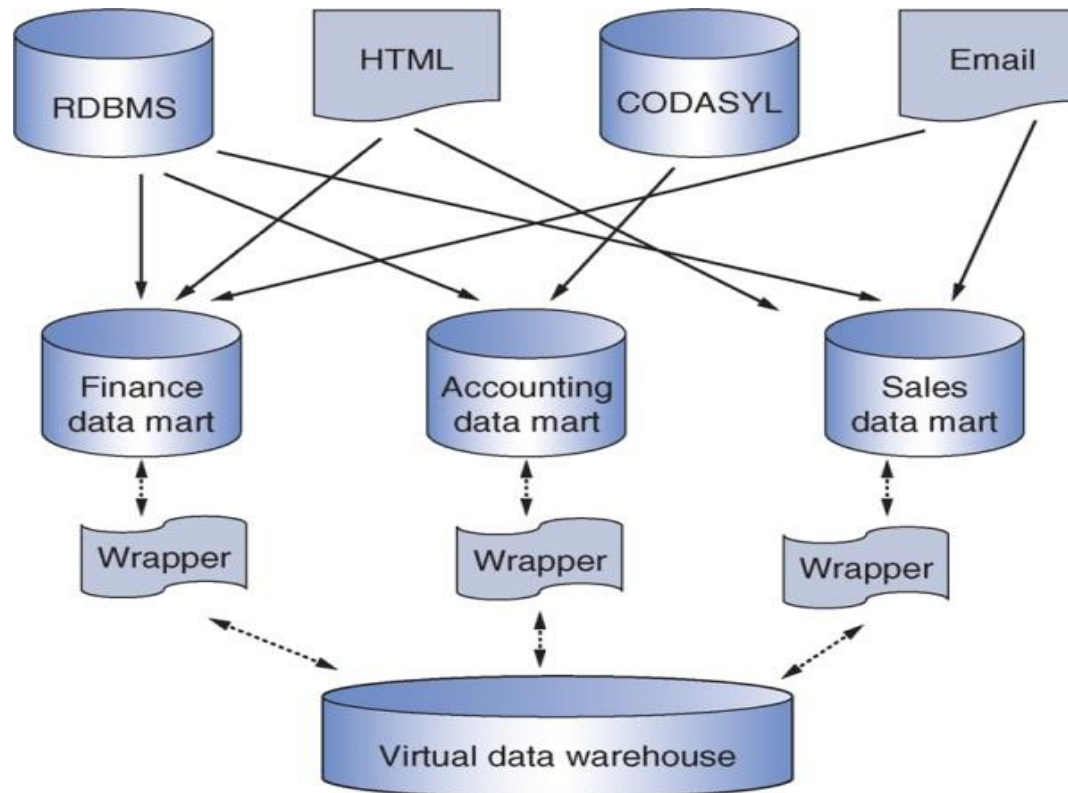
- ❑ **Virtual data warehouse** can be built as a set of SQL views directly on the underlying operational data sources



# Virtual Data Warehouses

---

- ❑ **Virtual data warehouse** can be built as an extra layer on top of a collection of physical independent data marts



# Virtual Data Warehouses

---

- ❑ **Virtual data warehouse** should provide a uniform and consistent metadata model which contains the schema mappings between the schemas of the underlying data stores and the schema of the virtual data warehouse
- ❑ Queries are reformulated and decomposed using these schema mappings on the fly.
- ❑ The wrappers are dedicated software components that receive queries from the upper level, execute them on the underlying data store, and convert the result to a format (e.g., relational tuples) that can be understood by the query processor
- ❑ The complexity of the wrapper (e.g., use API such as JDBC) depends on the data source

# Virtual Data Mart

---

- ❑ A **virtual data mart** is usually defined as a single SQL view
- ❑ The view can be directly defined on physical operational source data or a physical or virtual data warehouse
- ❑ View design is important
  - Virtual independent vs. Virtual dependent data mart
- ❑ Disadvantages:
  - Extra processing capacity from the underlying (operational) data sources
  - Not possible to keep track of historical data

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ❑ Data Warehouse Schemas
- ❑ ETL Process
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ☞ **Data Warehouses vs Data Lakes**
- ❑ Business Intelligence



# Data Lake

---

- ❑ Both data warehouses and data lakes are essentially data repositories
- ❑ A key distinguishing property of a **data lake** is that it stores raw data in its native format, which could be structured, unstructured, or semi-structured.
  - A data lake collects data emanating from operational sources “as is”, often without knowing upfront which analyses will be performed on it, or even whether the data will ever be involved in analysis at all.
- ❑ For these reason, either no or only very limited transformations of data before they enter the data lake
- ❑ No (less) schema structure. The data schema definitions are only determined when the data are read.

# Data Warehouses versus Data Lakes

---

	<b>Data Warehouse</b>	<b>Data lake</b>
<b>Data</b>	Structured	Often unstructured
<b>Processing</b>	Schema-on-write	Schema-on-read
<b>Storage</b>	Expensive	Low cost
<b>Transformation</b>	Before entering the DW	Before analysis
<b>Agility</b>	Low	High
<b>Security</b>	Mature	Maturing
<b>Users</b>	Decision makers	Data Scientists

# Outline

---

- ❑ Operational Systems (OLTP) vs Decision Support Systems (OLAP)
- ❑ Data Warehouse
- ❑ Data Warehouse Modeling
- ❑ ETL Process
- ❑ Data Marts
- ❑ Virtual Data Warehouses and Virtual Data Marts
- ❑ Data Warehouses vs Data Lakes
- ☞ **Business Intelligence**

# Business Intelligence

---

- ❑ **Business intelligence (BI)** is referred to as the set of activities, techniques and tools aimed at understanding patterns in past data and predicting the future.
- ❑ It is important that data is appropriately stored and managed and is of good quality
  - Otherwise, Garbage In, Garbage Out (GIGO) – bad data gives bad insights.
- ❑ **BI techniques**
  - Query and Reporting
  - Pivot tables
  - OLAP

# Query and Reporting

---

- ❑ **Query and reporting tools** are an essential component of a comprehensive BI solution
- ❑ They typically provide a user-friendly GUI.
- ❑ They should be by a business user , not an IT expert
  - Called to “Self Service BI”
  - “Query by Example” (QBE) - A query is composed in a user-friendly and visual way
- ❑ Report can be refreshed at any time
- ❑ Innovative visualization techniques

# Pivot Tables

---

- A **pivot-table** (or **cross-table**) is a popular data summarization tool.
- It essentially cross-tabulates a set of dimensions in such a way that multidimensional data can be represented in a two-dimensional tabular format.

Sales		Quarter				<u>Total</u>
		Q1	Q2	Q3	Q4	
Region	Europe	100	200	50	100	450
	Africa	50	100	200	50	400
	Asia	20	50	10	150	230
	America	50	10	100	100	260
	<u>Total</u>	220	360	360	400	1340

Aggregated  
data



# Example: MS Sever Analysis Services

Analysis Services Project2 - Microsoft Visual Studio

File Edit View Project Build Debug Database Cube Tools Window Help

Development

Finance data source view.cube [Design] Finance data source view.dsv [Design] Dim Organization.dim [Design]

Cube Stru... Dimension... Calculations KPIs Actions Partitions Aggregati... Perspecti... Translations Browser

Perspective: Finance data sour Language: Default

Measure Group: <All>

- Finance data source view
  - Measures
    - Fact Finance
      - Amount
      - Fact Finance Count
  - Dim Account
    - Account Description
    - Account Type
    - Parent Account Key
  - Dim Date
    - Calendar Quarter
    - Calendar Year
    - Date Key
    - Day Number Of Month
    - Day Number Of Week
    - Day Number Of Year
    - Fiscal Quarter
    - Fiscal Year
    - Month Number Of Year
    - Week Number Of Year
    - Calendar
      - Fiscal
  - Dim Department Group
  - Dim Organization
  - Dim Scenario
    - Scenario Key
    - Scenario Name

Dimension Hierarchy Operator Filter Expression

<Select dimension>

Scenario Name Budget

	Calendar Year 2005				Calendar Quarter 4		2006	Grand Total
	7	8	9	Total	Total			
Account Description	Amount	Amount	Amount	Amount	Amount	Amount	Amount	Amount
Amortization of Goodwill	1760	1760	1760	5280	5310	10590	10350	20940
Building Leasehold	9750	9750	9750	29250	30120	59370	59970	119340
Commissions	45000	45000	45000	135000	185400	320400	331200	651600
Conferences	1140	1140	1140	3420	3360	6780	7200	13980
Discounts		2300	600	2900	4500	7400	519500	526900
Employee Benefits	28110	28110	28110	84330	78300	162630	184380	347010
Entertainment	2360	2360	2360	7080	6510	13590	13800	27390
Equipment	660	660	660	1980	1950	3930	3990	7920
Furniture and Fixtures	2190	2190	2190	6570	7230	13800	13440	27240
Intercompany Sales	20700	116400	54200	191300	349700	541000	490500	1031500
Marketing Collateral	2540	2540	2540	7620	10980	18600	19710	38310
Meals	2930	2930	2930	8790	8280	17070	17010	34080
Office Supplies	4620	4620	4620	13860	12870	26730	27180	53910
Other Assets	1950	1950	1950	5850	5850	11700	11580	23280
Other Expenses	2720	2720	2720	8160	8010	16170	16170	32340
Payroll Taxes	35380	35380	35380	106140	116250	222390	248340	470730
Professional Services	2980	2980	2980	8940	8100	17040	17220	34260
Rent	8490	8490	8490	25470	26220	51690	51660	103350
Returns and Adjustments	10750	52400	59200	122350	168900	291250	375000	666250
Salaries	380900	380900	380900	1142700	1207200	2349900	2643300	4993200
Standard Cost of Sales	244300	630500	547700	1422500	2170100	3592600	4222400	7815000
Telephone	13650	13650	13650	40950	53370	94320	94740	189060
Trade Sales	744100	2241200	1612500	4597800	7214000	11811800	12511500	24323300
Travel Lodging	5780	5780	5780	17340	15510	32850	34470	67320
Travel Transportation	6270	6270	6270	18810	16770	35580	37260	72840
Utilities	8460	8460	8460	25380	25470	50850	52230	103170

Solution Explorer

- AdventureWorksCube1
  - Data Sources
    - Adventure Works D
  - Data Source Views
    - Finance data source
  - Cubes
    - Finance data source
  - Dimensions
    - Dim Department Gr
    - Dim Account.dim
    - Dim Scenario.dim
    - Dim Date.dim
    - Dim Organization.d
  - Mining Structures
  - Roles
  - Assemblies
  - Miscellaneous

Deployment Progress...

Server: localhost  
Database:

- Command
- Command
  - Processing Data
  - Start time: 4/
  - Processing D

Status:

Deployment Completed

# On-Line Analytical Processing (OLAP)

---

- ❑ **On-line analytical processing (OLAP)** allows you interactively analyze the data, summarize it and visualize it in various ways
- ❑ Provide the business-user with a powerful tool for ad-hoc querying
- ❑ Types of OLAP system
  - MOLAP (Multidimensional OLAP)
  - ROLAP (Relational OLAP)
  - HOLAP (Hybrid OLAP)



# Multidimensional OLAP (MOLAP)

---

- ❑ **Multidimensional OLAP (MOLAP)** stores the multidimensional data using a Multidimensional DBMS (MDBMS) whereby
  - the data is stored in a multi-dimensional array-based data structure optimized for efficient storage and quick access.
  - The dimensions represent the index keys of the array

<u>Array (key, value)</u>	Q1	Q2	Q3	Q4	<u>Total</u>
Product A	(1,1) 10	(1,2) 20	(1,3) 40	(1,4) 10	(1,5) 80
Product B	(2,1) 20	(2,2) 40	(2,3) 10	(2,4) 30	(2,5) 100
Product C	(3,1) 50	(3,2) 20	(3,3) 40	(3,4) 30	(3,5) 140
Product D	(4,1) 10	(4,2) 30	(4,3) 20	(4,4) 20	(4,5) 80
<u>Total</u>	(5,1) 90	(5,2) 110	(5,3) 110	(5,4) 90	(5,5) 400

\* NOTE: the totals are precomputed and stored in the array.

# MOLAP

---

- ❑ Fast in terms of data retrieval
- ❑ More storage space needed
- ❑ The array may become sparse with many zeros
- ❑ Scales poorly when the number of dimensions increases
- ❑ No universal SQL-like standard is provided
- ❑ Not optimized for transaction processing – updating, inserting, or deleting data is usually quite inefficient
- ❑ Typically not very portable because of their tight integration with particular BI tools

# ROLAP

---

- ❑ **Relational OLAP (ROLAP)** stores the data in a relational data warehouse, which can be implemented using a star, snowflake or fact constellation schema
- ❑ ROLAP scales better to more dimensions than MOLAP
- ❑ ROLAP query performance may however be inferior to MOLAP

# HOLAP

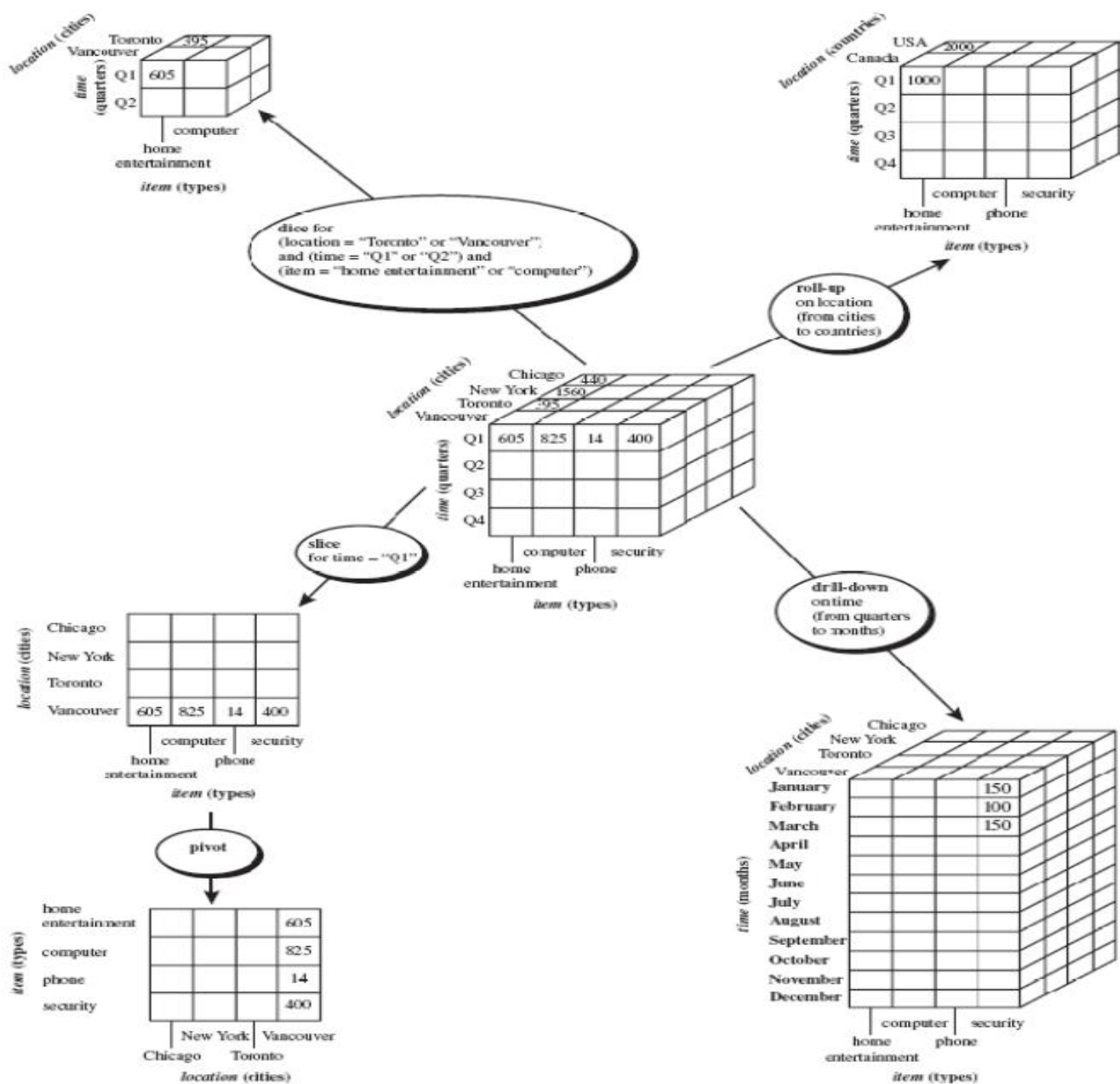
---

- ❑ **Hybrid OLAP (HOLAP)** tries to combine the best of both MOLAP and ROLAP
- ❑ RDBMS used to store the detailed data in a relational data warehouse whereas the pre-computed aggregated data is kept as a multidimensional array managed by an MDBMS
- ❑ OLAP analysis first starts from the multidimensional database
- ❑ Combine the performance of MOLAP with the scalability of ROLAP

# Typical OLAP Operations

---

- Functionality that can be expected:
  - **Slice and Dice:** Performing projection operations on the dimensions.
  - **Pivot:** Cross tabulation is performed
  - **Roll-up:** Data is summarized with increasing generalization
  - **Roll-down:** Increasing levels of detail are revealed
  - **Drill-across:** Information from two or more connected fact tables is accessed
  - **Sorting:** Data is sorted by ordinal value.
  - **Selection:** Data is available by value or range.
  - **Derived attributes:** Attributes are computed by operations on stored derived values

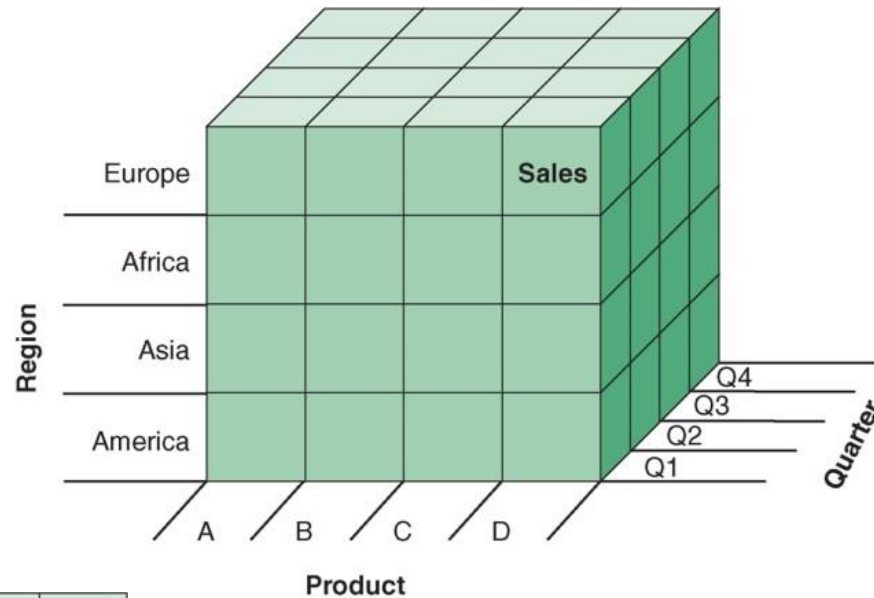


# Roll-up

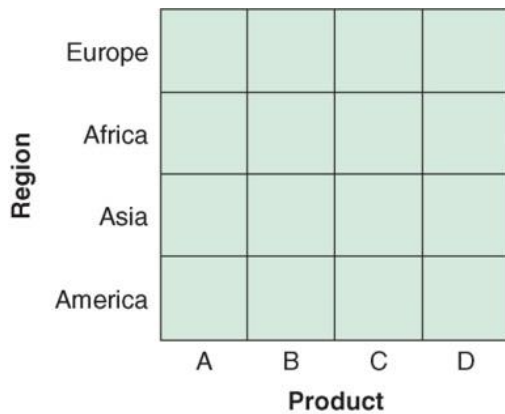
---

- **Roll-up** (or **drill-up**) refers to aggregating the current set of fact values within or across one or more dimensions
- Types:
  - A **hierarchical roll-up** aggregates within a particular dimension by climbing up the attribute hierarchy (e.g., from day to week to month to quarter to year)
  - A **dimensional roll-up** aggregates across an entire dimension and then drops it.

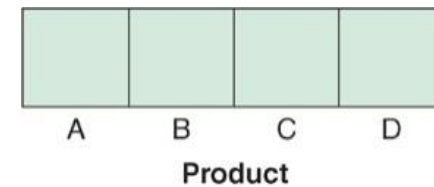
# Example



**Figure.** OLAP Cube



**Figure.** Rolling up the time dimension



**Figure.** Rolling up the time and region dimension



# Roll-down

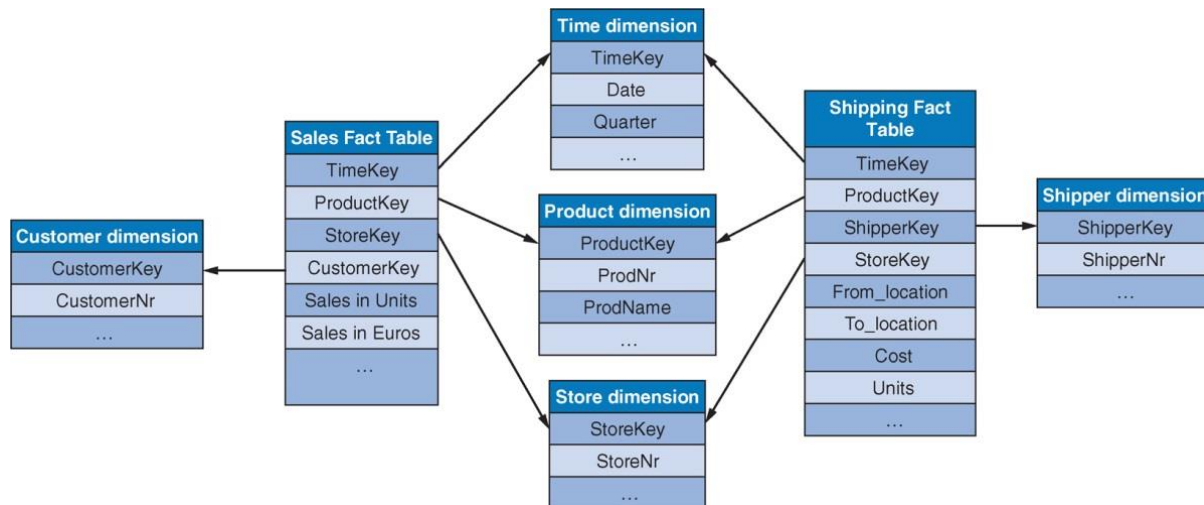
---

- ❑ **Roll-down** (or **drill-down**) is the reverse process of roll-up.
- ❑ Roll-down de-aggregates by navigating from a lower level of detail to a higher level of detail.
- ❑ Types:
  - **Hierarchical roll-down** (e.g., from year to quarter to month to week to day)
  - **Dimensional roll-down** in which a new dimension is added to the analysis

# Drill-across

---

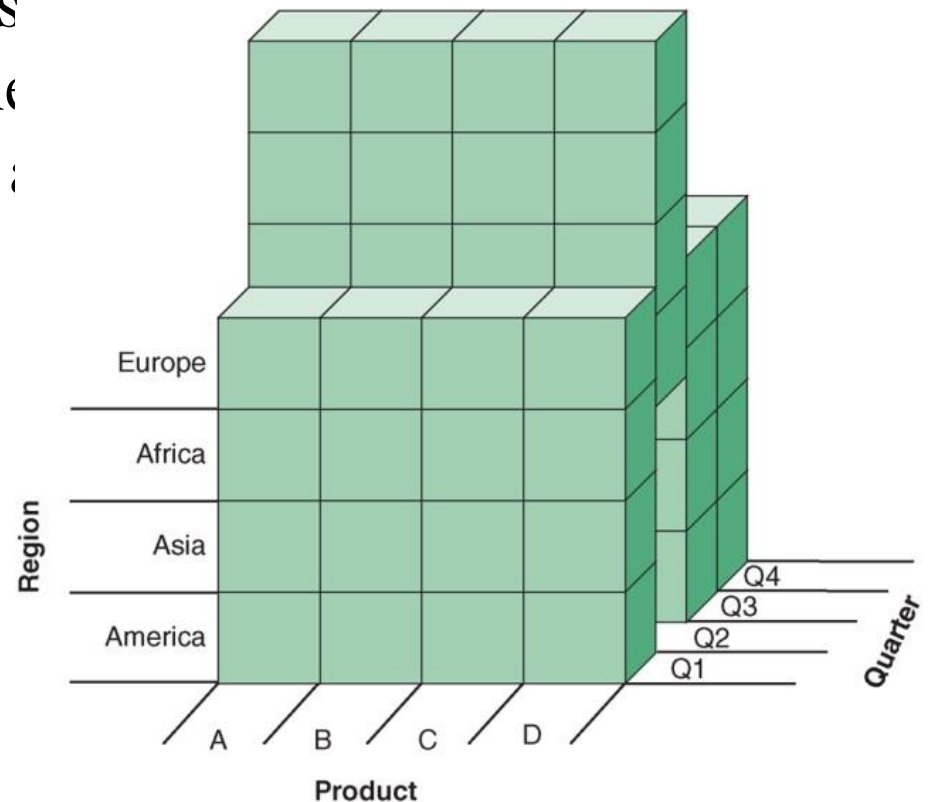
- ❑ **Drill-across** is another OLAP operation whereby information from two or more connected fact tables is accessed
- ❑ Example: Adding shipping fact data to an analysis on sales fact data is an example of a drill-across operation



# Slicing Operation

---

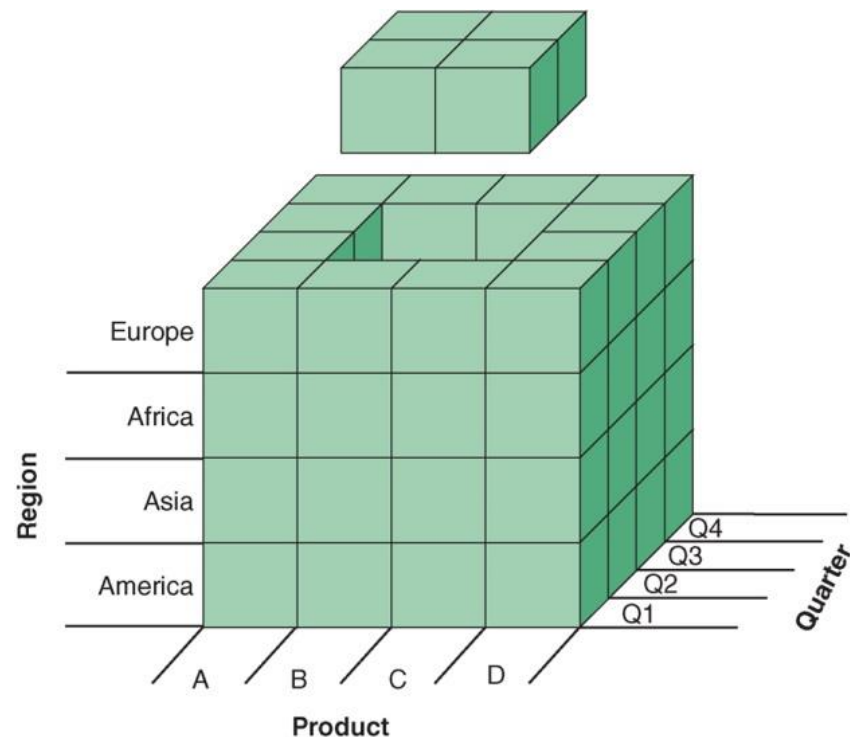
- ❑ The **slicing** operation performs a selection on one dimension of the given cube, resulting a sub-cube.
- ❑ The operation performs a selection whereby one of the dimension is set a particular value.



# Dicing Operation

---

- The **dicing** operation defines a sub-cube by performing a selection on two or more dimensions



# OLAP Queries in SQL

---

- ❑ CUBE operator
- ❑ ROLLUP operator
- ❑ GROUPING SETS operator
- ❑ RANK operator

# CUBE Operator

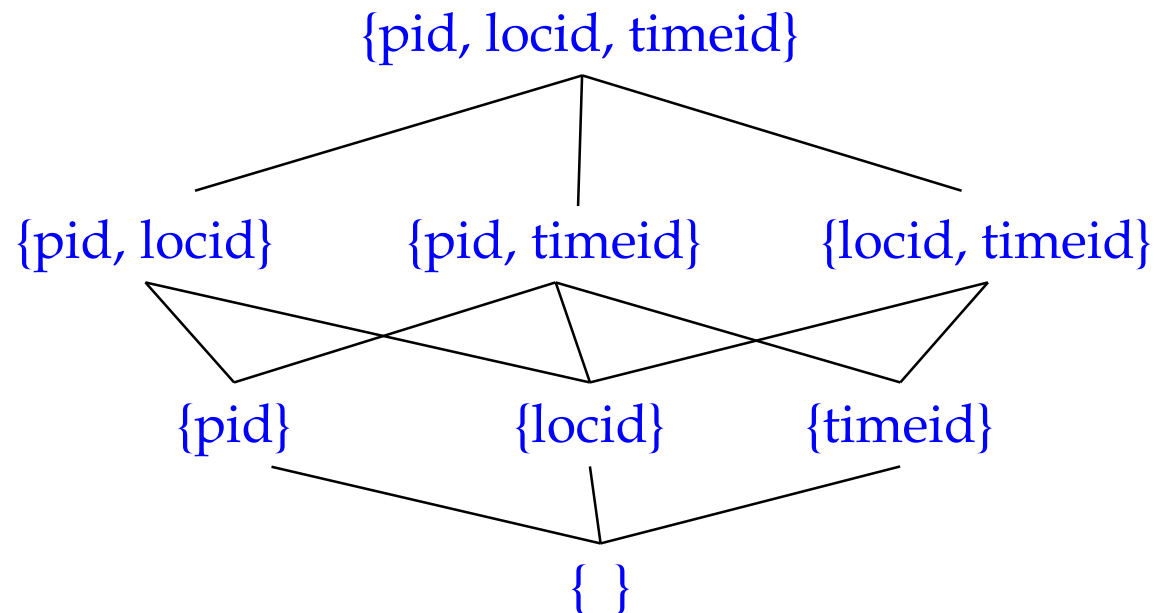
---

- ❑ The **CUBE** operator computes a union of GROUP Bys on every subset of the specified attribute types.
- ❑ Its result set represents a multidimensional cube based upon the source table.
- ❑ It is useful generating both subtotals and totals

# CUBE Operator

---

- The CUBE operator computes a union of GROUP BY's on every subset of the specified attributes
  - Generalizing the previous example, if there are  $k$  dimensions, we have  $2^k$  possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.



# Example: CUBE Operator

SELECT quarter, region, SUM(sales)  
FROM SALESTABLE  
GROUP BY **CUBE**(quarter, region)

- $2^2 = 4$  groupings of SALESTABLE:  
{(quarter, region), (quarter), (region), ()}
- Result:  $4 \times 2 + 4 \times 1 + 1 \times 2 + 1 = 15$   
tuples

QUARTER	REGION	SALES
Q1	Europe	50
Q1	America	80
Q2	Europe	40
Q2	America	60
Q3	Europe	NULL
Q3	America	40
Q4	Europe	20
Q4	America	80
Q1	NULL	130
Q2	NULL	100
Q3	NULL	40
Q4	NULL	90
NULL	Europe	110
NULL	America	250
NULL	NULL	360

PRODUCT	QUARTER	REGION	SALES
A	Q1	Europe	10
A	Q1	America	20
A	Q2	Europe	20
A	Q2	America	50
A	Q3	America	20
A	Q4	Europe	10
A	Q4	America	30
B	Q1	Europe	40
B	Q1	America	60
B	Q2	Europe	20
B	Q2	America	10
B	Q3	America	20
B	Q4	Europe	10
B	Q4	America	40

SALESTABLE



# ROLLUP Operator

---

- ❑ The **ROLLUP** operator computes the union on every prefix of the list of specified attribute types, from the most detailed up to the grand total.
- ❑ The order in which the attribute types are mentioned is important for ROLLUP
- ❑ The key difference
  - ROLLUP generates a result set showing the aggregates for a hierarchy of values of the specified attribute types
  - CUBE generates a result set showing the aggregates for all combinations of values of the selected attribute types

# Example: ROLLUP Operator

SELECT quarter, region, SUM(sales)  
FROM SALESTABLE  
GROUP BY **ROLLUP**(quarter, region)

- $2^2 = 4$  groupings of SALESTABLE:  
{(quarter, region), (quarter), ()}
- Result:  $4 \times 2 + 4 + 1 = 13$  tuples

QUARTER	REGION	SALES
Q1	Europe	50
Q1	America	80
Q2	Europe	40
Q2	America	60
Q3	Europe	NULL
Q3	America	40
Q4	Europe	20
Q4	America	80
Q1	NULL	130
Q2	NULL	100
Q3	NULL	40
Q4	NULL	90
NULL	NULL	360

PRODUCT	QUARTER	REGION	SALES
A	Q1	Europe	10
A	Q1	America	20
A	Q2	Europe	20
A	Q2	America	50
A	Q3	America	20
A	Q4	Europe	10
A	Q4	America	30
B	Q1	Europe	40
B	Q1	America	60
B	Q2	Europe	20
B	Q2	America	10
B	Q3	America	20
B	Q4	Europe	10
B	Q4	America	40

SALESTABLE

# GROUPG SETS Operator

---

- The **GROUPING SETS** operator generates a result set equivalent to that generated by a UNION ALL of multiple simple GROUP BY clauses

```
SELECT quarter, region, SUM(sales)
FROM SALESTABLE
GROUP BY GROUPING SETS ((quarter), (region))
```

```
SELECT quarter, NULL, SUM(sales)
FROM SALESTABLE
GROUP BY quarter
UNION ALL
SELECT NULL, region, SUM(sales)
FROM SALESTABLE
GROUP BY region
```

# CUBE vs GROUPING SETS

---

```
SELECT quarter, region, SUM(sales)
FROM SALESTABLE
GROUP BY CUBE(quarter, region)
```

is equivalent to

```
SELECT quarter, region, SUM(sales)
FROM SALESTABLE
GROUP BY GROUPING SETS ((quarter, region),
(quarter), (region), ())
```

# ROLLUP vs GROUPING SETS

---

```
SELECT quarter, region, SUM(sales)
FROM SALESTABLE
GROUP BY ROLLUP(quarter, region)
```

is equivalent to

```
SELECT quarter, region, SUM(sales)
FROM SALESTABLE
GROUP BY GROUPING SETS ((quarter, region),
(quarter), ())
```

# RANK Operator

---

- ❑ SQL 2003 introduces additional analytical support for two types of frequently encounter OLAP activates:
- ❑ Ranking should always be done in combination with an SQL ORDER BY clause.
- ❑ Rank functions
  - **RANK()** assigns a rank based upon the ordered value, whereby similar values assigned the same rank.
  - **DENSE\_RANK()** calculates the percentage of values less than the current value, excluding the highest values,  $(\text{RANK}() - 1) / (\text{NumberofRows} - 1)$
  - **CUM\_DIST()** calculates the cumulative distribution or the percentage of values less than or equal to the current value.

# Example: Ranking with ORDER BY

---

```
SELECT product, sales,  
RANK() OVER (ORDER BY sales ASC) AS rank_sales,  
DENSE_RANK() OVER (ORDER BY sales ASC) AS dense_rank_sales,  
PERCENT_RANK() OVER (ORDER BY sales ASC) AS  
percent_rank_sales,  
CUM_DIS() OVER (ORDER BY sales ASC) AS cum_dist_sales,  
FROM SALESTABLE  
ORDER BY rank_sales ASC
```

Product	Sales	RANK_SALES	DENSE_RANK_SALES	PERC_RANK_SALES	CUM_DIST_SALES
C	10	1	1	0	0.1
I	15	2	2	1/9=0.11	0.2
B	20	3	3	2/9=0.22	0.4
H	20	3	3	2/9=0.22	0.4
J	25	5	4	4/9=0.44	0.5
F	30	6	5	5/9=0.55	0.6
E	40	7	6	6/9=0.66	0.7
D	45	8	7	7/9=0.77	0.8
A	50	9	8	8/9=0.88	0.9
G	60	10	9	9/9=1	1

PRODUCT	SALES
A	50
B	20
C	10
D	45
E	40
F	30
G	60
H	20
I	15
J	25

An example table

The query result

# Example: Ranking with PARTITION BY

---

```
SELECT region, product, sales,  
RANK() OVER (PARTITION BY region ORDER BY sales ASC)  
AS rank_sales,  
PERCENT_RANK() OVER (PARTITION BY region ORDER BY  
sales ASC) AS percent_rank_sales  
FROM SALESTABLE  
ORDER BY rank_sales ASC
```

PRODUCT	QUARTER	REGION	SALES
A	Q1	Europe	10
A	Q1	America	20
A	Q2	Europe	20
A	Q2	America	50
A	Q3	America	20
A	Q4	Europe	10
A	Q4	America	30
B	Q1	Europe	40
B	Q1	America	60
B	Q2	Europe	20
B	Q2	America	10
B	Q3	America	20
B	Q4	Europe	10
B	Q4	America	40

An example table



# Windowing Query

---

- ❑ **Windowing** allows calculating cumulative totals or running averages based on a specified window of values
- ❑ Windows allows getting access to more than one row of a table without requiring a self-join

```
SELECT quarter, region, sales,  
AVG(sales) OVER (PARTITION BY region ORDER BY quarter  
ROWS BETWEEN 1 and PRECEDING AND 1 FOLLOWING)  
AS sales_avg  
FROM SALESTABLE  
ORDER BY region, quarter, sales_avg
```

QUARTER	REGION	SALES	SALES_AVG
1	America	10	15
2	America	20	13.33
3	America	10	20
4	America	30	20
1	Europe	10	15
2	Europe	20	13.33
3	Europe	10	16.67
4	Europe	20	15

The query result

QUARTER	REGION	SALES
1	America	10
2	America	20
3	America	10
4	America	30
1	Europe	10
2	Europe	20
3	Europe	10
4	Europe	20

An example table

# **APPENDIX:** Analytical Queries in SQL

# Example Tables

---

<i>Locid</i>	<i>City</i>	<i>State</i>	<i>country</i>
1	Madison	WI	USA
2	Fresno	CA	USA
3	Chennai	TN	India

Locations

<i>Timeid</i>	<i>T_Date</i>	<i>T_Week</i>	<i>T_Month</i>	<i>T_Quarter</i>	<i>T_Year</i>	<i>T_Holiday</i>
1	01/01/08	1	Jan	1	2008	Y
2	02/01/07	1	Feb	1	2007	N
3	02/01/06	1	Feb	1	2006	N
...						

Times

<i>Pid</i>	<i>Pname</i>	<i>Category</i>	<i>Price</i>
11	Lee Jeans	Apparel	25
12	Zord	Toys	18
13	Biro Pen	Stationary	2
14	Bang Bang	Apparel	10

Products

<i>pid</i>	<i>timeid</i>	<i>locid</i>	<i>sales</i>
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	3	20
13	1	2	20
13	2	2	40
13	3	3	5

Slaes\_F

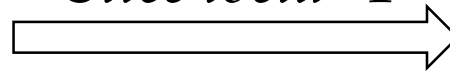
# Slicing in SQL

---

- In Relational OLAP, the slicing operation can be conducted with equality selections on one dimension

pid	11	12	13
	8	10	10
	30	20	50
	25	8	15
	1	2	3
	timeid		
	locid		

*Slice locid=1*

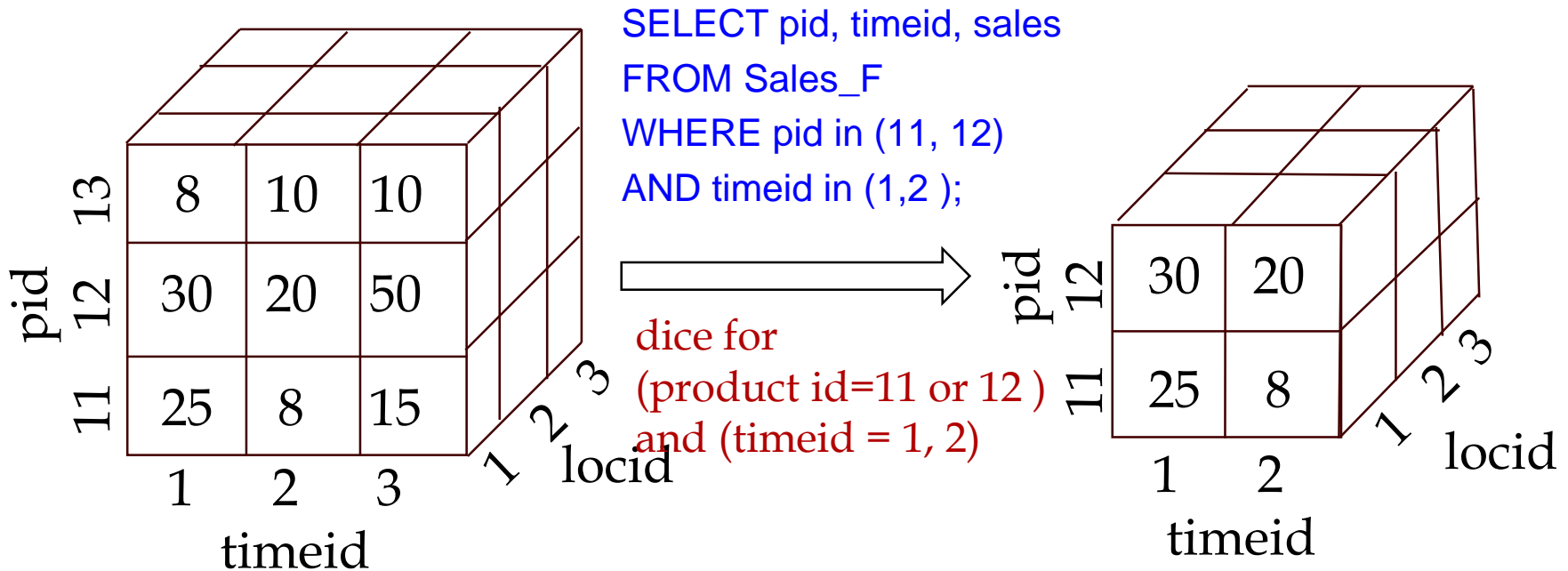


```
SELECT pid, timeid, sales  
FROM Sales_F  
WHERE locid=1;
```

pid	11	12	13
	8	10	10
	30	20	50
	25	8	15
	1	2	3
	timeid		

# Dicing in SQL

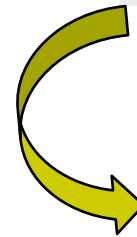
- The dicing operation can be implemented with range selections on one or more dimensions



# PIVOT() in SQL3

```
SELECT *
FROM (
  SELECT
    t.t_year AS " ",
    l.state AS lstate,
    sum(f.sales) sales_sum
  FROM sales_f f, locations l, times t
  WHERE f.locid=l.locid AND f.timeid=t.timeid
  GROUP BY t.t_year, l.state
) t
PIVOT (sum(sales_sum) for lstate in ('CA','WI','TN'));
```

	'CA'	'WI'	'TN'
2007	107	38	(null)
2006	10	75	25
2008	81	63	(null)





```
SELECT *
FROM (
  SELECT
    t.t_year AS tyear,
    l.state AS " ",
    sum(f.sales) sales_sum
  FROM sales_f f, locations l, times t
  WHERE f.locid=l.locid AND f.timeid=t.timeid
  GROUP BY t.t_year, l.state
) t
PIVOT (sum(sales_sum) for tyear in (2008, 2007, 2006));
```

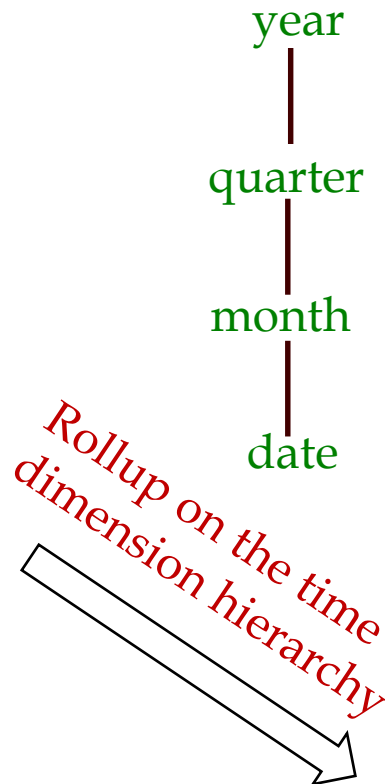
	2008	2007	2006
TN	(null)	(null)	25
CA	81	107	10
WI	63	38	75

# Roll-up in SQL



---

```
SELECT t.t_date, SUM(sales)
FROM sales_F s, times t
WHERE s.timeid=t.timeid
GROUP BY t.t_date;
```

 T_DATE	 SUM(SALES)
01-JAN-08	144
01-FEB-06	110
01-FEB-07	145



```
SELECT t.t_month, SUM(sales)
FROM sales_F s, times t
WHERE s.timeid=t.timeid
GROUP BY t.t_month;
```

 T_MONTH	 SUM(SALES)
Feb	255
Jan	144

# Drill-down in SQL


---

```
SELECT l.state, SUM(sales)
FROM sales_F s, locations l
WHERE s.locid=l.locid
GROUP BY l.state;
```

STATE	SUM(SALES)
CA	198
WI	176
TN	25

region  
|  
state  
|  
city  
|  
store

Roll down on the location  
dimension hierarchy



```
SELECT l.city, SUM(sales)
FROM sales_F s, locations l
WHERE s.locid=l.locid
GROUP BY l.city;
```

CITY	SUM(SALES)
Fresno	198
Madison	176
Chennai	25



# Cross-Tabulation in SQL

---

- The cross-tabulation obtained by pivoting can also be computed using a collection of SQL queries:

```
SELECT T.t_year, L.state, SUM(S.sales)
FROM Sales_F S, Times T, Locations L
WHERE S.timeid=T.timeid
AND S.locid=L.locid
GROUP BY T.t_year, L.state
```

```
SELECT T.t_year, SUM(S.sales)
FROM Sales_F S, Times T
WHERE S.timeid=T.timeid
GROUP BY T.t_year
```

```
SELECT SUM(S.sales)
FROM Sales_F S
```

```
SELECT L.state, SUM(S.sales)
FROM Sales_F S, Location L
WHERE S.locid=L.locid
GROUP BY L.state
```

	WI	CA	TN	Total
2008	63	81	0	144
2007	38	107	0	145
2006	75	10	25	110
Total	176	198	25	399

# CUBE() in SQL3

- The cross-tabulation obtained by pivoting

```
SELECT timeid, locid, SUM(sales)
FROM Sales_F
GROUP BY CUBE(timeid, locid);
```

	1	2	3	Total
1	75	10	25	110
2	38	107	0	145
3	63	81	0	144
Total	176	198	25	399

```
SELECT T.t_year, L.state, SUM(S.sales)
FROM Sales_F S, Times T, Locations L
WHERE S.timeid=T.timeid
AND S.locid=L.locid
GROUP BY CUBE(T.t_year, L.state);
```

	WI	CA	TN	Total
2008	63	81	0	144
2007	38	107	0	145
2006	75	10	25	110
Total	176	198	25	399

timeid	locid	SUM (sales)
1	2	10
1	3	25
1	1	75
1	null	110
2	2	107
2	1	38
2	null	145
3	2	81
3	1	63
3	null	144
null	2	198
null	3	25
null	1	176
null	null	399

T.year	L.state	SUM (S.sales)
2006	CA	10
2006	TN	25
2006	WI	75
2006	null	110
2007	CA	107
2007	WI	38
2007	null	145
2008	CA	81
2008	WI	63
2008	null	144
null	CA	198
null	TN	25
null	WI	176
null	null	399

# ROLLUP() in SQL3

---

```
SELECT T.t_year, L.state, SUM(S.sales)
FROM   Sales_F S, Times T, Locations L
WHERE  S.timeid=T.timeid
AND S.locid=L.locid
GROUP BY CUBE(T.t_year, L.state)
```

```
SELECT T.t_year, L.state, SUM(S.sales)
FROM   Sales_F S, Times T, Locations L
WHERE  S.timeid=T.timeid
AND S.locid=L.locid
GROUP BY ROLLUP(T.t_year, L.state)
```

	WI	CA	TN	Total
2008	63	81	0	144
2007	38	107	0	145
2006	75	10	25	110
				399

T.year	L.state	SUM (S.sales)
2006	CA	10
2006	TN	25
2006	WI	75
2006	null	110
2007	CA	107
2007	WI	38
2007	null	145
2008	CA	81
2008	WI	63
2008	null	144
<i>null</i>	null	399

# GROUPG SETS operation in SQL 3

---

- **GROUPING SETS** operator generates a result set equivalent to that generated by a UNION ALL of multiple simple GROUP BY clauses

```
SELECT QUARTER, REGION, SUM(SALES)
FROM SALESTABLE
GROUP BY GROUPING SETS ((QUARTER), (REGION))
```

```
SELECT QUARTER, NULL, SUM(SALES)
FROM SALESTABLE
GROUP BY QUARTER
UNION ALL
SELECT NULL, REGION, SUM(SALES)
FROM SALESTABLE
GROUP BY REGION
```

# Complex Analytical Queries

---

- Trend analysis is difficult to do in SQL-92:
  - Find the % change in monthly sales
  - Find the top 5 product by total sales
  - Find the trailing  $n$ -day moving average of sales
- The first two queries can be expressed with difficulty, but the third cannot even be expressed in SQL-92 if  $n$  is a parameter of the query.

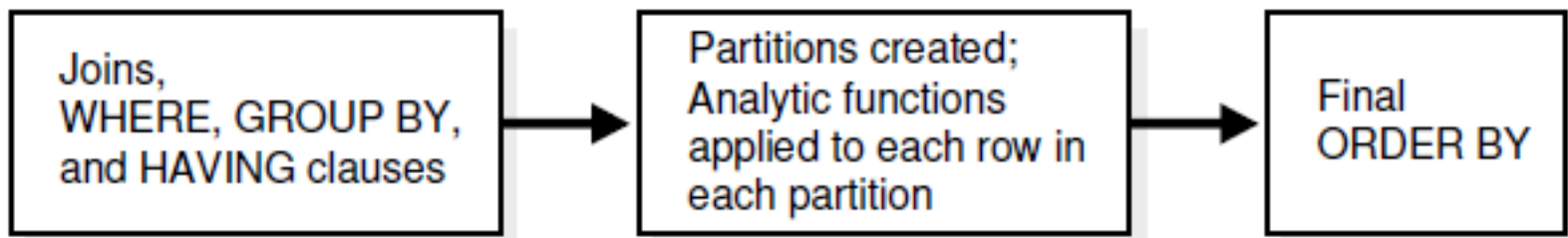
## *Analytic Functions and Their Uses*

Type	Used For
Ranking	Calculating ranks, percentiles, and n-tiles of the values in a result set.
Windowing	Calculating cumulative and moving aggregates. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE, and new statistical functions. Note that the DISTINCT keyword is not supported in windowing functions except for MAX and MIN.
Reporting	Calculating shares, for example, market share. Works with these functions: SUM, AVG, MIN, MAX, COUNT (with/without DISTINCT), VARIANCE, STDDEV, RATIO_TO_REPORT, and new statistical functions. Note that the DISTINCT keyword may be used in those reporting functions that support DISTINCT in aggregate mode.
LAG/LEAD	Finding a value in a row a specified number of rows from a current row.
FIRST/LAST	First or last value in an ordered group.
Linear Regression	Calculating linear regression and other statistics (slope, intercept, and so on).
Inverse Percentile	The value in a data set that corresponds to a specified percentile.
Hypothetical Rank and Distribution	The rank or percentile that a row would have if inserted into a specified data set.

# Processor Order

---

- ❑ Query processing using analytic functions takes place in three stages.
  - First, all joins, WHERE, GROUP BY and HAVING clauses are performed.
  - Second, the result set is made available to the analytic functions, and all their calculations take place.
  - Third, if the query has an ORDER BY clause at its end, the ORDER BY is processed to allow for precise output ordering.



# General Syntax of Analytic Function

---

```
function (arg1, ..., argn)  
OVER  
([PARTITION BY <...>][ORDER BY <...>] [<window_clause>])
```

Example, COUNT(\*) OVER (PARTITION BY deptno)

- ❑ SUM, COUNT, AVG, MIN, MAX are the common analytic functions the result of which does not depend on the order of the records
- ❑ Functions like LEAD, LAG, RANK, DENSE\_RANK, ROW\_NUMBER, FIRST, FIRST VALUE, LAST, LAST VALUE depends on order of records.



# Group by

```
SELECT locid, COUNT(*) SALE_COUNT  
FROM sales_f  
GROUP BY locid;
```

LOCID	SALE_COUNT
1	9
2	7
3	2

It groups the records in accordance with the GROUP BY clause.

## Group by clause

- Create partition of rows
- Apply an group (aggregate) function to a partion
- Single output row per partition
- Any non-"group by" column is not allowed in the select clause.

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	3	20
13	1	2	20
13	2	2	40
13	3	3	5

Slaes\_F

# How analytic function query is different from group functions

```
SELECT pid, locid,
COUNT(*) OVER (PARTITION BY locid) SALE_COUNT
FROM sales_f;
```

Note the repeating values of SALE\_COUNT

PID	LOCID	SALE_COUNT
11	1	9
12	1	9
13	1	9
13	1	9
13	1	9
12	1	9
12	1	9
11	1	9
11	1	9
12	2	7
11	2	7
13	2	7
12	2	7
13	2	7
11	2	7
11	2	7
12	3	2
13	3	2

## Partition by clause

- Create partitions of rows
- Apply an analytical function to a partition
- One output row per row.  
Return the group value multiple times with each row.
- Any other non “group-by” column or expression can be present in the select clause, e.g., pid in this example

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	3	20
13	1	2	20
13	2	2	40
13	3	3	5

Slaes\_F

# Analytic Function Query

---

- ❑ PARTITION BY is used to break the result set into groups.
- ❑ PARTITION BY can take any non-analytic SQL expression
- ❑ Analytic functions can only appear in the SELECT list and in the main ORDER BY clause of the query.
- ❑ Analytic functions are computed after all joins, WHERE clause, GROUP BY and HAVING are computed on the query.
- ❑ The main ORDER BY clause of the query operates after the analytic functions.

# Without PARTITION BY

- In absence of any PARTITION or <window\_clause> inside the OVER() portion, the aggregate function acts on entire record set returned by the WHERE clause

```
SELECT pid, locid,  
       COUNT(*) OVER () SALE_COUNT  
FROM sales_f  
WHERE locid in (1, 3);
```

Compare with

```
SELECT COUNT(*)  
FROM sales_f  
WHERE locid in (1,3);
```

```
COUNT(*)  
-----  
11
```

```
SELECT pid, locid,  
       COUNT(*) OVER (PARTITION BY locid) SALE_COUNT  
FROM sales_f  
WHERE locid in (1, 3);
```

PID	LOCID	SALE_COUNT
11	1	11
12	1	11
13	1	11
13	1	11
13	1	11
12	1	11
12	1	11
11	1	11
11	1	11
12	3	11
13	3	11

# ORDER BY in OVER() clause

---

- ❑ To specify the order of the records in the partition, use “ORDER BY” clause inside the OVER() clause.
  - **Different from the ORDER BY clause of the main query which comes after WHERE**

## ❑ Example

```
select dno, lname, ssno,  
row_number() over (partition by dno order by ssno ) as emp_id  
from employee;
```

```
select lname, r  
from ( select lname ,  
            row_number() over (order by lname) R from employee)  
where r between 10 and 12
```

# Ranking Functions

---

- ❑ RANK and DENSE\_RANK Functions
- ❑ Bottom (/Top)  $N$  Ranking Functions
- ❑ CUME\_DIST Function
  - Computes the cumulative distribution of a value in a group of values
- ❑ PERCENT\_RANK Function
  - Similar to CUME\_DIST but uses rank values rather than row counts
- ❑ NTILE Function
  - Allows easy calculation of quartiles and other common summary statistics
- ❑ ROW\_NUMBER Function
  - Assign a unique number

# RANK and DENSE\_RANK functions

- **RANK()** and **DENSE\_RANK()** both provide rank to the records based on some column value or expression

In case of a tie of 2 records at position N,

- RANK declares 2 positions N and skips position N+1 and gives position N+2 to the next records,
- DENSE\_RANK declares 2 positions N but does not skip position N+1.

PID	LOCID	SALES	SAL_RANK	SAL_DENSE_RANK
12	1	50	1	1
12	1	30	2	2
11	1	25	3	3
12	1	20	4	4
11	1	15	5	5
13	1	10	6	6
13	1	10	6	6
11	1	8	8	7
13	1	8	8	7
12	3	20	1	1
13	3	5	2	2

```
SELECT pid, locid, sales,  
       RANK() OVER (PARTITION BY locid ORDER BY sales DESC) SAL_RANK,  
       DENSE_RANK() OVER (PARTITION BY locid ORDER BY sales DESC ) SAL_DENSE_RANK  
FROM sales_f  
WHERE locid in (1, 3)  
ORDER BY locid, SAL_RANK;
```

# ROW\_NUMBER function

---

- ❑ **ROW\_NUMBER()** gives a running serial number to a partition of records.
  - Assign integer values to the rows depending on their order.
- ❑ Very useful in reporting, especially in places where different partitions have their own serial number.

```
SELECT pid, locid,  
       ROW_NUMBER() OVER (PARTITION BY locid ORDER BY pid) SRLNO  
FROM sales_f  
WHERE locid in (1, 3);
```

PID	LOCID	SRLNO
11	1	1
11	1	2
11	1	3
12	1	4
12	1	5
12	1	6
13	1	7
13	1	8
13	1	9
12	3	1
13	3	2



# ROW\_NUMBER function

---

## □ Examples

```
SELECT dno, lname, ssn,  
       ROW_NUMBER() OVER (PARTITION BY dno ORDER BY ssn )  
       AS emp_id  
FROM employee;
```

```
SELECT lname, r  
FROM (SELECT lname ,  
       ROW_NUMBER() over (ORDER BY lname) AS r  
       FROM employee)  
WHERE r BETWEEN 10 AND 12;
```

# Top N Queries in Oracle

---

```
SELECT * FROM  
(SELECT L.state, P.pname, S.sales,  
    RANK() OVER (PARTITION BY L.state  
                ORDER BY S.sales desc ) Top3  
FROM Sales_F S, Products P, Locations L  
WHERE S.pid=P.pid AND S.locid=L.locid  
)  
WHERE Top3 <=3
```

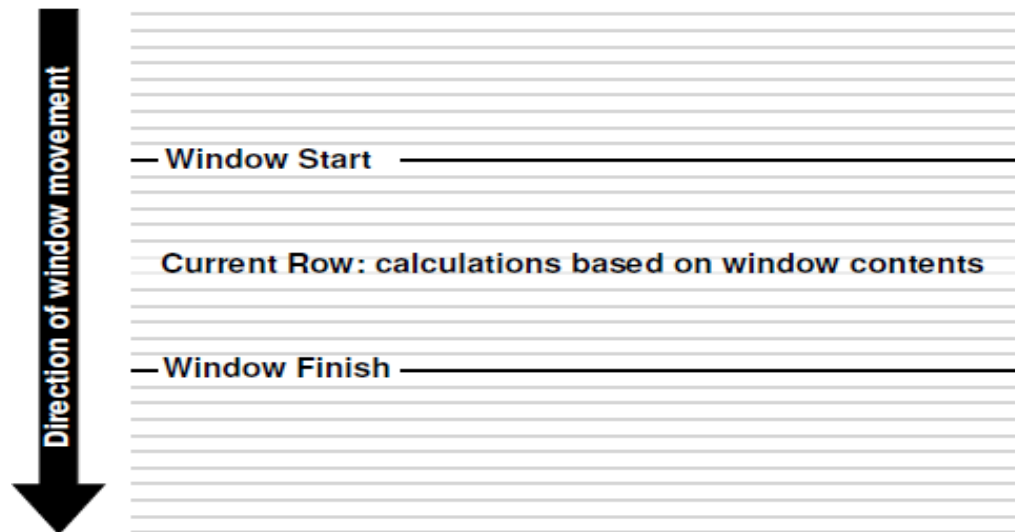
RANK calculates the rank of a value in a group of values. The return type is NUMBER.

STATE	PNAME	SALES	TOP3
CA	Zord	45	1
CA	Biro Pen	40	2
CA	Lee Jeans	35	3
TN	Zord	20	1
TN	Biro Pen	5	2
WI	Zord	50	1
WI	Zord	30	2
WI	Lee Jeans	25	3

# Window for Aggregation

---

- ❑ Each calculation performed with an analytic function is based on a current row within a partition.
- ❑ For each row in a partition, you can define a **sliding window** of data.
- ❑ This window determines the range of rows used to perform the calculations for the current row



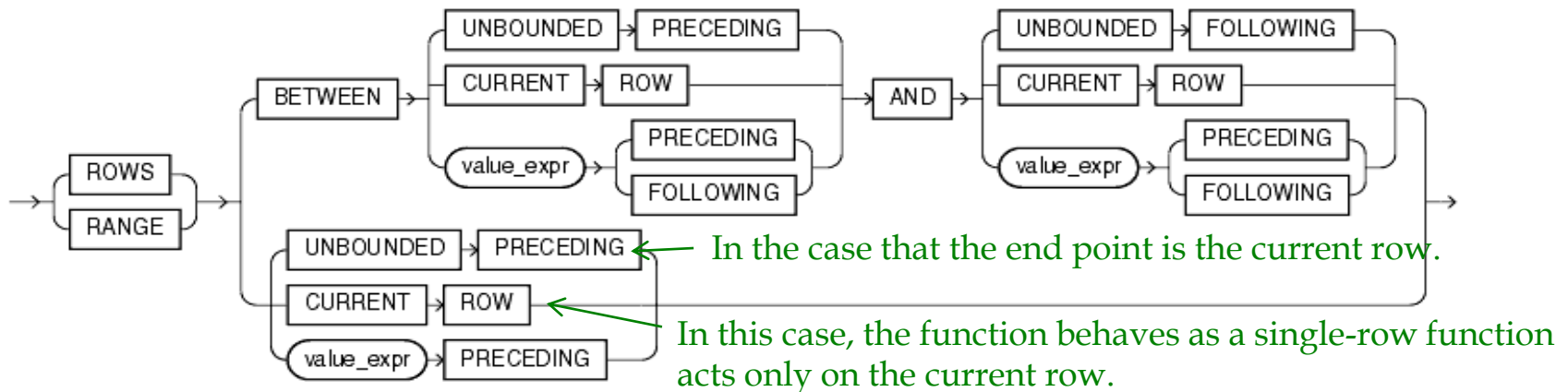
# Windowing Functions

---

- ❑ Some analytic functions (AVG, COUNT, FIRST\_VALUE, LAST\_VALUE, MAX, MIN and SUM) can take a window clause to further sub-partition the result, and compute cumulative, moving and centered aggregates.
- ❑ They return a value for each row in the table, which depends on other rows in the corresponding window.

# Window clause in Oracle

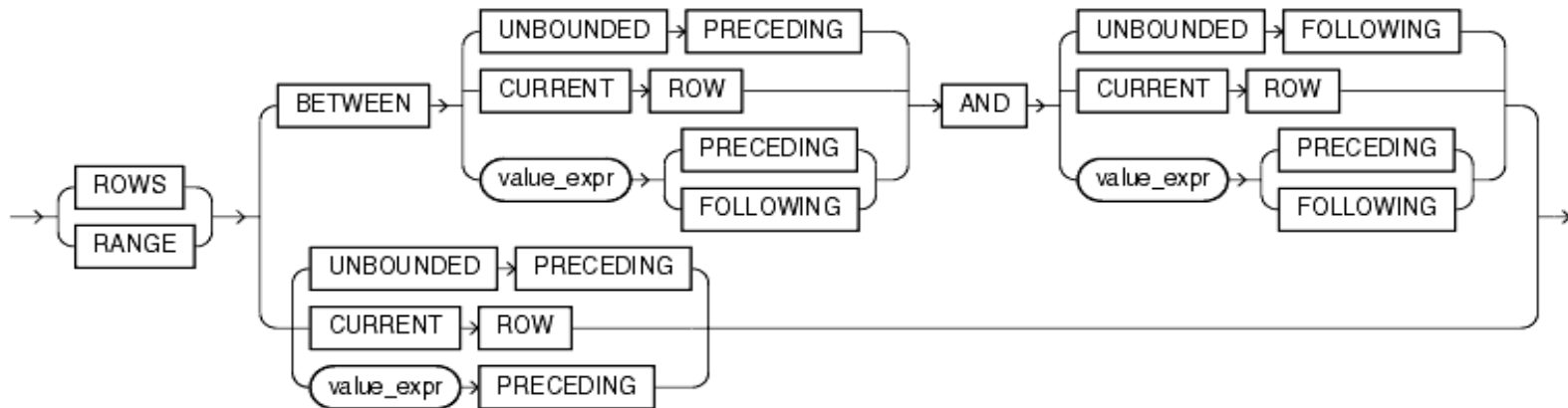
□ *[ROWS or RANGE] BETWEEN <start\_expr> AND <end\_expr>*



- *<start\_expr>* and *<end\_expr>* can be any one of the following
  - UNBOUNDED PRECEDING (default) or
  - CURRENT ROW or
  - *<value\_expr>* PRECEDING or FOLLOWING
- The start point of the window and the end point of the window can finish before the current row or after the current row
- Only start point cannot come after the end point of window

# Window clause in Oracle (Cont.)

□ *[ROWS or RANGE] BETWEEN <start\_expr> AND <end\_expr>*



- The window clause is defined in terms of the current row. But may or may not include the current row.
- For **ROW type windows**, the definition is in terms of row number before or after the current row. **<sql\_expr>** must evaluate to a positive integer.
- For **RANGE type windows**, the definition is in terms of values before or after the current order.

# ROW Type Windows

---

- General syntax for analytic functions with ROW type windows

***Function( ) OVER (PARTITION BY <expr1>***

***ORDER BY <expr2,..>***

***ROWS BETWEEN <start\_expr> AND <end\_expr> )***

*or*

***Function( ) OVER (PARTITION BY <expr1>***

***ORDER BY <expr2,..>***

***ROWS [ <start\_expr> PRECEDING or***

***UNBOUNDED PRECEDING]***

# Example

```
SELECT l.state, t.t_month, s.sales,
SUM(s.sales) OVER (PARTITION BY l.state
ORDER BY t.t_month
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
THREE_DAY_SALE_SUM
FROM sales_f s, locations l, times t
WHERE s.locid=l.locid
AND s.timeid=t.timeid
ORDER BY l.state, t.t_month;
```

**\*\* For ROW type windows, the window clause is in terms of record numbers**

STATE	T_MONTH	SALES	THREE_DAY_SALE_SUM
CA	Feb	22	67
CA	Feb	45	107
CA	Feb	40	120
CA	Jan	35	101
CA	Jan	26	81
CA	Jan	20	56
CA	Mar	10	30
TN	Mar	5	25
TN	Mar	20	25
WI	Feb	10	30
WI	Feb	20	38
WI	Feb	8	58
WI	Jan	30	46
WI	Jan	8	63
WI	Jan	25	43
WI	Mar	10	50
WI	Mar	15	75
WI	Mar	50	65

*partition by state  
order by month*

## Window + Partition by

- Create partitions of rows
- Order within partition
- Create window
- Apply an aggregate function to the window
- One output row per each row

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	3	20
13	1	2	20
13	2	2	40
13	3	3	5

Slaes\_F



# Example: RANGE Type Windows

---

- General syntax for analytic functions with RANGE type windows

*Function( )* **OVER (PARTITION BY <expr1>  
ORDER BY <expr2>  
RANGE BETWEEN <start\_expr> AND <end\_expr>)**

*or*

*Function( )* **OVER (PARTITION BY <expr1>  
ORDER BY <expr2>  
RANGE [<start\_expr> PRECEDING  
or UNBOUNDED PRECEDING]**

- The default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

# RANGE BETWEEN

```
SELECT locid, pid, sales,
       SUM(sales) OVER (PARTITION BY locid
                        ORDER BY pid
                        RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING)
       THREE_DAY_SALE_SUM
FROM sales_f;
```

LOCID	PID	SALES	THREE_DAY_SALE_SUM
1	11	8	148
1	11	25	148
1	11	15	148
1	12	30	176
1	12	20	176
1	12	50	176
1	13	10	128
1	13	8	128
1	13	10	128
2	11	35	138
2	11	22	138
2	11	10	138
2	12	26	198
2	12	45	198
2	13	20	131
2	13	40	131
3	12	20	25
3	13	5	25

**\*\* For RANGE type windows, <exp> is a logical offset.**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	3	20
13	1	2	20
13	2	2	40
13	3	3	5

Slaes\_F

# LEAD and LAG functions

---

- ❑ **LEAD()** has the ability to compute an expression on the next rows (rows which are going to come after the current row) and return the value to the current row.
- ❑ General syntax of LEAD

**LEAD (<sql\_expr>, <offset>, <default>)  
OVER (<analytic\_clause>)**

- <sql\_expr> is the expression to compute from the leading row.
- <offset> is the index of the leading row relative to the current row. It is a positive integer with default 1.
- <default> is the value to return if the <offset> points to a row outside the partition range.

# LEAD and LAG functions (Cont.)

- The syntax of LAG is similar except that the offset for LAG goes into the previous rows

```
SELECT pid, locid, sales,  
       LEAD(sales) OVER (PARTITION BY locid ORDER BY sales DESC NULLS LAST)  
       NEXT_LOWER_SAL,  
       LAG(sales) OVER (PARTITION BY locid ORDER BY sales DESC NULLS LAST)  
       PREV_HIGHTER_SAL  
FROM sales_f  
WHERE locid in (1, 3)  
ORDER BY locid, sales DESC;
```

PID	LOCID	SALES	NEXT_LOWER_SAL	PREV_HIGHTER_SAL
12	1	50	30	0
12	1	30	25	50
11	1	25	20	30
12	1	20	15	25
11	1	15	10	20
13	1	10	8	15
13	1	10	8	15
11	1	8	0	10
13	1	8	0	10
12	3	20	5	0
13	3	5	0	20

# FIRST VALUE and LAST VALUE function

---

## □ General syntax

**FIRST\_VALUE (<sql\_expr>) OVER  
(<analytic\_clause>)**

- <sql\_expr> is computed on the columns of the first record and results are returned.
- FIRST\_VALUE analytic function picks the first record from the partition after doing the ORDER BY.
- LAST\_VALUE function is used in similar context except that it acts on the last record of the partition

```
SELECT locid, f.pid, t.t_date,  
FIRST_VALUE(t.t_date) OVER (PARTITION BY locid ORDER BY t.t_date) FIRST_SALE_DATE,  
t.t_date - FIRST_VALUE(t.t_date) OVER (partition by locid order by t.t_date) DAY_GAP  
FROM sales_f f, times t  
WHERE f.timeid=t.timeid;
```

# Example: FIRST\_VALUE()

---

```
SELECT locid, f.pid, t.t_date,  
FIRST_VALUE(t.t_date) OVER (PARTITION BY locid ORDER BY t.t_date) FIRST_SALE_DATE,  
t.t_date - FIRST_VALUE(t.t_date) OVER (partition by locid order by t.t_date) DAY_GAP  
FROM sales_f f, times t  
WHERE f.timeid=t.timeid;
```

LOCID	PID	T_DATE	FIRST_SALE_DATE	DAY_GAP
1	13	01-FEB-06	01-FEB-06	0
1	12	01-FEB-06	01-FEB-06	0
1	11	01-FEB-06	01-FEB-06	0
1	12	01-FEB-07	01-FEB-06	365
1	13	01-FEB-07	01-FEB-06	365
1	11	01-FEB-07	01-FEB-06	365
1	11	01-JAN-08	01-FEB-06	699
1	13	01-JAN-08	01-FEB-06	699
1	12	01-JAN-08	01-FEB-06	699
2	11	01-FEB-06	01-FEB-06	0
2	13	01-FEB-07	01-FEB-06	365
2	11	01-FEB-07	01-FEB-06	365
2	12	01-FEB-07	01-FEB-06	365
2	12	01-JAN-08	01-FEB-06	699
2	13	01-JAN-08	01-FEB-06	699
2	11	01-JAN-08	01-FEB-06	699
3	12	01-FEB-06	01-FEB-06	0
3	13	01-FEB-06	01-FEB-06	0

# Summary: Analytic Functions

---

- \* AVG
- \* COVAR\_POP
- \* COUNT
- \* DENSE\_RANK
- \* FIRST\_VALUE
- \* LAST
- \* LEAD
- \* MIN
- \* PERCENT\_RANK
- \* PERCENTILE\_DISC
- \* RATIO\_TO\_REPORT
- \* STDDEV\_POP
- \* SUM
- \* VAR\_SAMP
- \* CORR
- \* COVAR\_SAMP
- \* CUME\_DIST
- \* FIRST
- \* LAG
- \* LAST\_VALUE
- \* MAX
- \* NTILE
- \* PERCENTILE\_CONT
- \* RANK
- \* STDDEV
- \* STDDEV\_SAMP
- \* VAR\_POP
- \* VARIANCE

ORACLE All functions:

[http://download.oracle.com/docs/cd/B19306\\_01/server.102/b14200/functions001.htm](http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions001.htm)