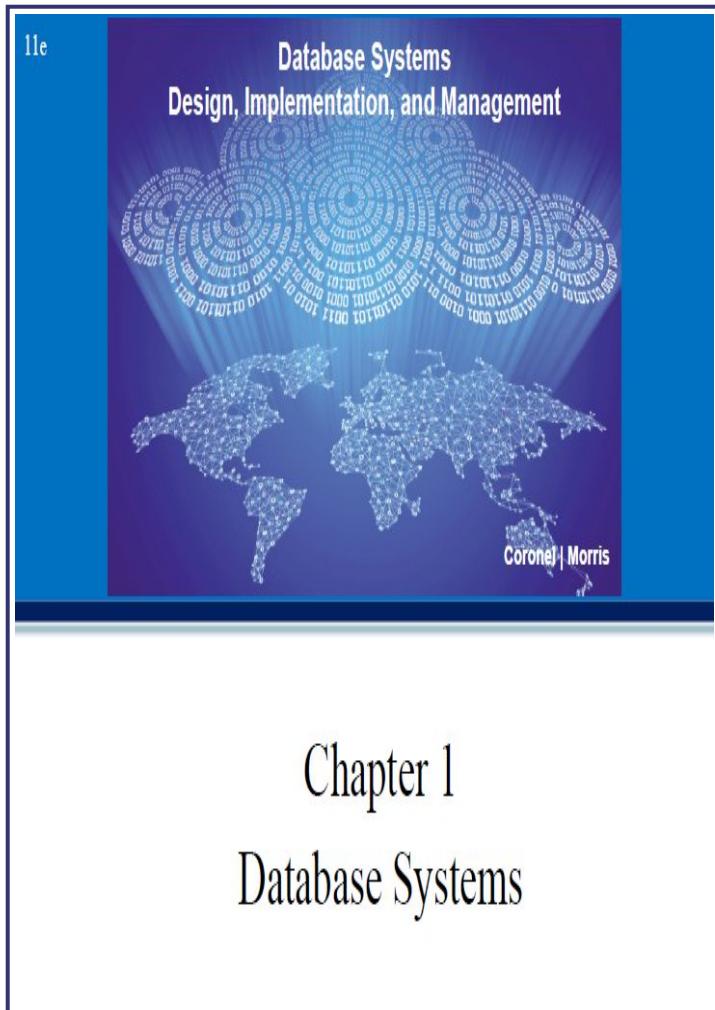


1/30 11:07:13 \*\*\*



# Introduction

# Database systems



## Chapter 1

### Database Systems

# What you will learn:

## Learning Objectives

- In this chapter, you will learn:
  - The difference between data and information
  - What a database is, the various types of databases, and why they are valuable assets for decision making
  - The importance of database design
  - How modern databases evolved from file systems

# What else you will learn:

## Learning Objectives

- In this chapter, you will learn:
  - About flaws in file system data management
  - The main components of the database system
  - The main functions of a database management system (DBMS)

# Data != information!

## Data vs. Information

Data	Information
<ul style="list-style-type: none"><li>▪ Raw facts<ul style="list-style-type: none"><li>▪ Raw data - Not yet been processed to reveal the meaning</li></ul></li><li>▪ Building blocks of information</li><li>▪ <b>Data management</b><ul style="list-style-type: none"><li>▪ Generation, storage, and retrieval of data</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ Produced by processing data</li><li>▪ Reveals the meaning of data</li><li>▪ Enables <b>knowledge</b> creation</li><li>▪ Should be accurate, relevant, and timely to enable good decision making</li></ul>

# DB, DBMS

## Database

- Shared, integrated computer structure that stores a collection of:
  - End-user data - Raw facts of interest to end user
  - **Metadata:** Data about data, which the end-user data are integrated and managed
    - Describe data characteristics and relationships
- **Database management system (DBMS)**
  - Collection of programs
  - Manages the database structure
  - Controls access to data stored in the database

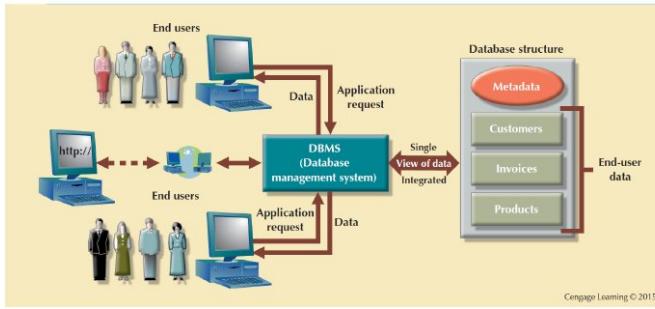
# Why DBMS?

## Role of the DBMS

- Intermediary between the user and the database
- Enables data to be shared
- Presents the end user with an integrated view of the data
- Receives and translates application requests into operations required to fulfill the requests
- Hides database's internal complexity from the application programs and users

# DBMS is a go-between

Figure 1.2 - The DBMS Manages the Interaction between the End User and the Database



# DBMS: advantages

## Advantages of the DBMS

- Better data integration and less data inconsistency
  - **Data inconsistency:** Different versions of the same data appear in different places
- Increased end-user productivity
- Improved:
  - Data sharing
  - Data security
  - Data access
  - Decision making
    - **Data quality:** Promoting accuracy, validity, and timeliness of data

# Types of DBs: based on user count

## Types of Databases

- **Single-user database:** Supports one user at a time
  - **Desktop database:** Runs on PC
- **Multiuser database:** Supports multiple users at the same time
  - **Workgroup databases:** Supports a small number of users or a specific department
  - **Enterprise database:** Supports many users across many departments

# Types of DBs: based on location

## Types of Databases

- **Centralized database:** Data is located at a single site
- **Distributed database:** Data is distributed across different sites
- **Cloud database:** Created and maintained using cloud data services that provide defined performance measures for the database

# Types of DBs: based on content

## Types of Databases

- **General-purpose databases:** Contains a wide variety of data used in multiple disciplines
- **Discipline-specific databases:** Contains data focused on specific subject areas

# Types of DBs: based on data currency

## Types of Databases

- **Operational database:** Designed to support a company's day-to-day operations
- **Analytical database:** Stores historical data and business metrics used exclusively for tactical or strategic decision making
- **Data warehouse:** Stores data in a format optimized for decision support

# Types of DBs [cont'd]

## Types of Databases

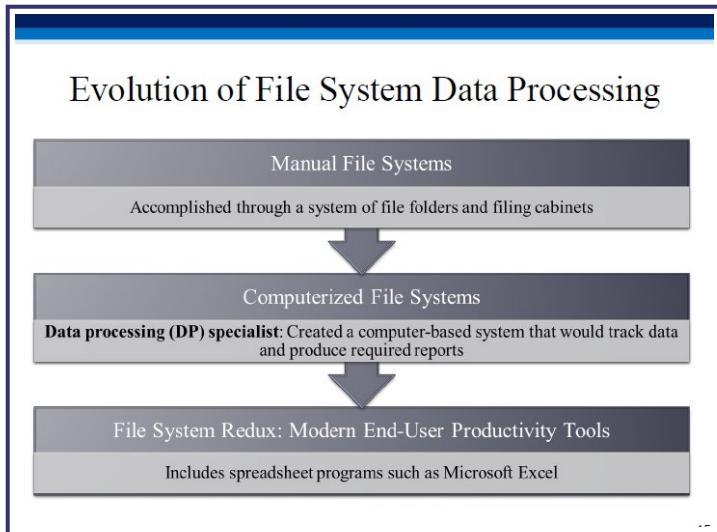
- **Online analytical processing (OLAP)**
  - Enable retrieving, processing, and modeling data from the data warehouse
- **Business intelligence:** Captures and processes business data to generate information that support decision making

# Types of DBs: based on the structure of contained data

## Types of Databases

- **Unstructured data:** It exists in their original state
- **Structured data:** It results from formatting
  - Structure is applied based on type of processing to be performed
- **Semistructured data:** Processed to some extent
- **Extensible Markup Language (XML)**
  - Represents data elements in textual format

# Early DBs: file systems



# File systems

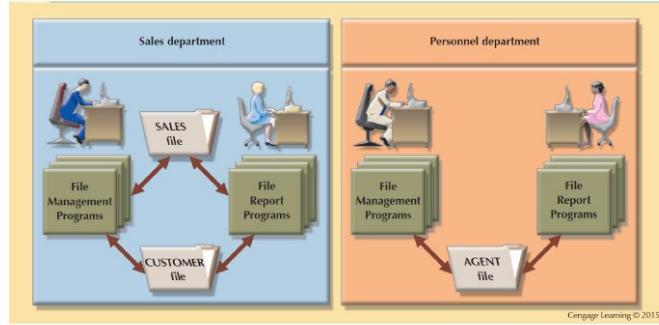
Table 1.2 - Basic File Terminology

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data have little meaning unless they have been organized in some logical manner.
Field	A character or group of characters (alphabetical or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a record for a customer might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

Cengage Learning © 2015

# File system

Figure 1.6 - A Simple File System



# File systems: problems

## Problems with File System Data Processing

Lengthy development times

Difficulty of getting quick answers

Complex system administration

Lack of security and limited data sharing

Extensive programming

# 'Structural' dependence (not a good thing!)

## Structural and Data Dependence

- **Structural dependence:** Access to a file is dependent on its own structure
  - All file system programs are modified to conform to a new file structure
- **Structural independence:** File structure is changed without affecting the application's ability to access the data

# Structural dependence [cont'd]

## Structural and Data Dependence

- Data dependence
  - Data access changes when data storage characteristics change
- Data independence
  - Data storage characteristics is changed without affecting the program's ability to access the data
- Practical significance of data dependence is difference between logical and physical format

# Redundancy of data (again, not a good thing!)

## Data Redundancy

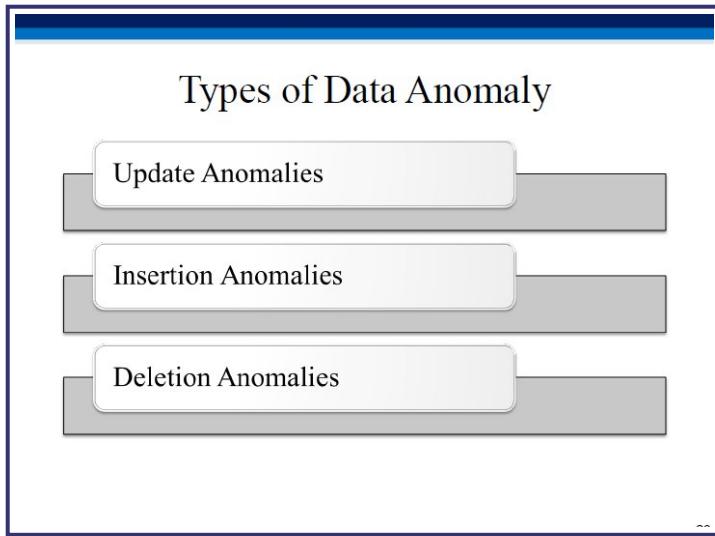
- Unnecessarily storing same data at different places
- **Islands of information:** Scattered data locations
  - Increases the probability of having different versions of the same data

# Why is redundancy not a good thing?

## Data Redundancy Implications

- Poor data security
- Data inconsistency
- Increased likelihood of data-entry errors when complex entries are made in different files
- **Data anomaly:** Develops when not all of the required changes in the redundant data are made successfully

# The three types of data anomalies



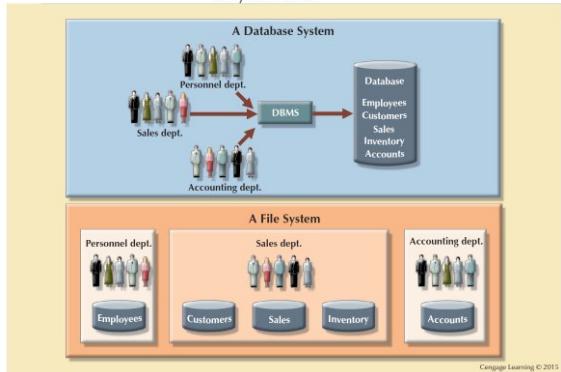
# DB systems

## Database Systems

- Logically related data stored in a single logical data repository
  - Physically distributed among multiple storage facilities
  - DBMS eliminates most of file system's problems
- Current generation DBMS software:
  - Stores data structures, relationships between structures, and access paths
  - Defines, stores, and manages all access paths and components

# DB vs file system

Figure 1.8 - Contrasting Database and File Systems



# DBMS

## DBMS Functions

### Data dictionary management

- **Data dictionary:** Stores definitions of the data elements and their relationships

### Data storage management

- **Performance tuning:** Ensures efficient performance of the database in terms of storage and access speed

### Data transformation and presentation

- Transforms entered data to conform to required data structures

### Security management

- Enforces user security and data privacy

# DBMS [cont'd]

## DBMS Functions

### Multiuser access control

- Sophisticated algorithms ensure that multiple users can access the database concurrently without compromising its integrity

### Backup and recovery management

- Enables recovery of the database after a failure

### Data integrity management

- Minimizes redundancy and maximizes consistency

# DBMS [cont'd]

## DBMS Functions

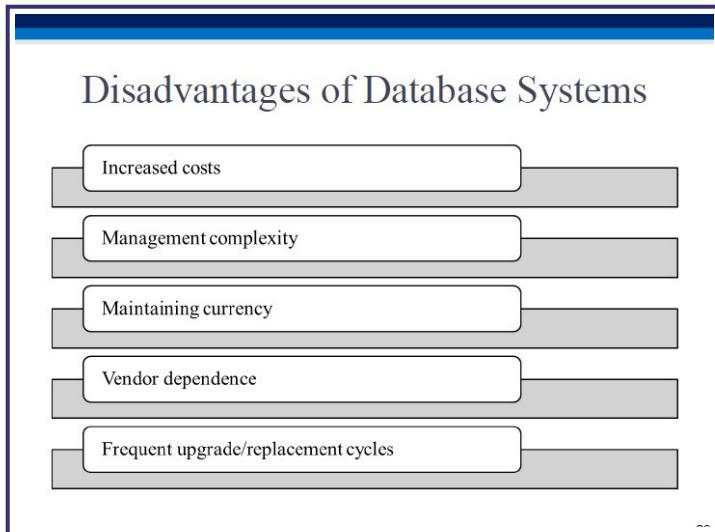
Database access languages and application programming interfaces

- **Query language:** Lets the user specify what must be done without having to specify how
- **Structured Query Language (SQL):** De facto query language and data access standard supported by the majority of DBMS vendors

Database communication interfaces

- Accept end-user requests via multiple, different network environments

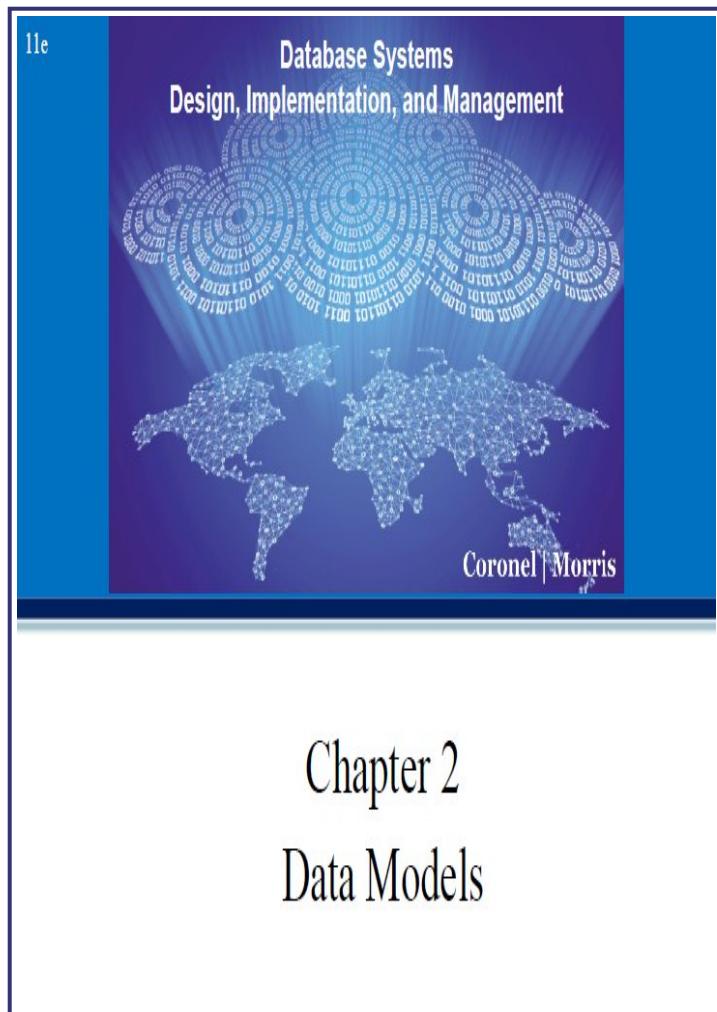
# How DBs could be "bad"



1/39    11:11:34 \*\*\*



# Data Modeling



## Chapter 2

### Data Models

## Learning Objectives

- In this chapter, you will learn:
  - About data modeling and why data models are important
  - About the basic data-modeling building blocks
  - What business rules are and how they influence database design

## Learning Objectives

- In this chapter, you will learn:
  - How the major data models evolved
  - About emerging alternative data models and the need they fulfill
  - How data models can be classified by their level of abstraction

## Data Modeling and Data Models

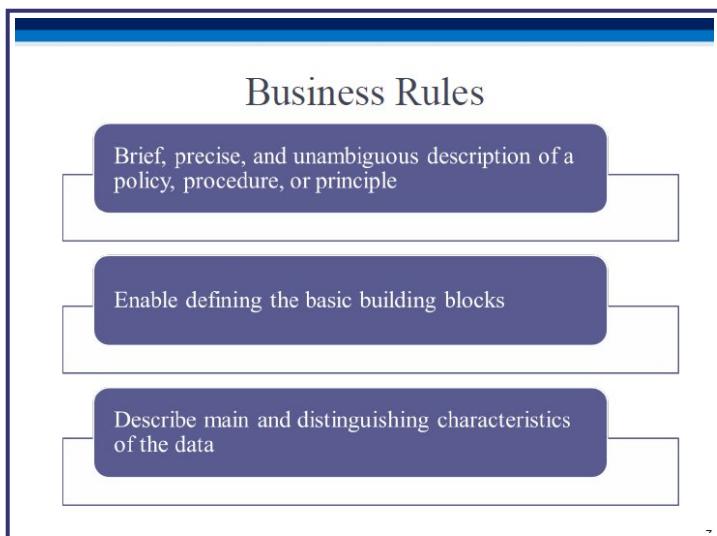
- **Data modeling:** Iterative and progressive process of creating a specific data model for a determined problem domain
- **Data models:** Simple representations of complex real-world data structures
- Useful for supporting a specific problem domain
- **Model -** Abstraction of a real-world object or event

## Importance of Data Models

- Are a communication tool
- Give an overall view of the database
- Organize data for various users
- Are an abstraction for the creation of good database

## Data Model Basic Building Blocks

- **Entity:** Unique and distinct object used to collect and store data
  - **Attribute:** Characteristic of an entity
- **Relationship:** Describes an association among entities
  - **One-to-many (1:M)**
  - **Many-to-many (M:N or M:M)**
  - **One-to-one (1:1)**
- **Constraint:** Set of rules to ensure data integrity





## Reasons for Identifying and Documenting Business Rules

- Help standardize company's view of data
- Communications tool between users and designers
- Allow designer to:
  - Understand the nature, role, scope of data, and business processes
  - Develop appropriate relationship participation rules and constraints
  - Create an accurate data model

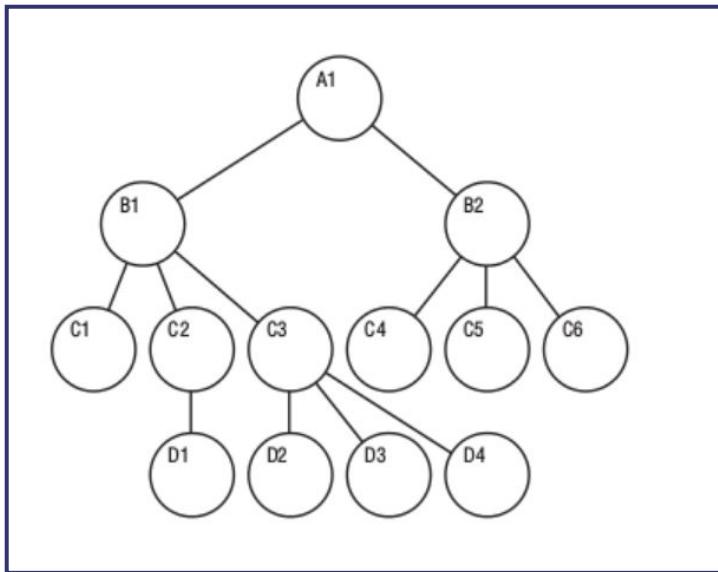
## Translating Business Rules into Data Model Components

- Nouns translate into entities
- Verbs translate into relationships among entities
- Relationships are bidirectional
- Questions to identify the relationship type
  - How many instances of B are related to one instance of A?
  - How many instances of A are related to one instance of B?

## Naming Conventions

- Entity names - Required to:
  - Be descriptive of the objects in the business environment
  - Use terminology that is familiar to the users
- Attribute name - Required to be descriptive of the data represented by the attribute
- Proper naming:
  - Facilitates communication between parties
  - Promotes self-documentation

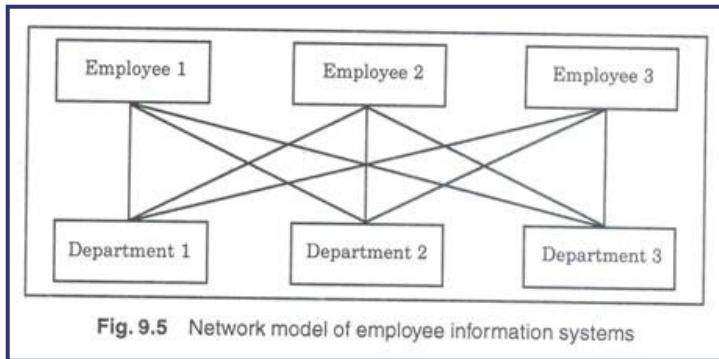
# Hierarchical modeling



At first, data was stored in individual files (transitioned from paper). The next improvement was a 'hierarchical DB model', where data was structured in the form of a tree [similar to a modern filesystem]. Data, in the form of nodes, are linked in a tree-like fashion. To traverse the tree, we need to know the underlying format ('class hierarchy, to make an analogy with classes and objects), and the actual path [eg. to relate A1 and D2, we need to traverse A1->B1->C3>D2].

Hierarchies are good for '1:M' [tree], but not 'M:N' [graph or multiple inheritance].

# Network modeling



A network model is better than a hierarchical one, because it can capture M:N [in addition to the above, another example is 'products and orders'].

## Hierarchical and Network Models

### Hierarchical Models

- Manage large amounts of data for complex manufacturing projects
- Represented by an upside-down tree which contains segments
  - **Segments:** Equivalent of a file system's record type
- Depicts a set of one-to-many (1:M) relationships

### Network Models

- Represent complex data relationships
- Improve database performance and impose a database standard
- Depicts both one-to-many (1:M) and many-to-many (M:N) relationships

## Hierarchical Model

Advantages	Disadvantages
<ul style="list-style-type: none"><li>▪ Promotes data sharing</li><li>▪ Parent/child relationship promotes conceptual simplicity and data integrity</li><li>▪ Database security is provided and enforced by DBMS</li><li>▪ Efficient with 1:M relationships</li></ul>	<ul style="list-style-type: none"><li>▪ Requires knowledge of physical data storage characteristics</li><li>▪ Navigational system requires knowledge of hierarchical path</li><li>▪ Changes in structure require changes in all application programs</li><li>▪ Implementation limitations</li><li>▪ No data definition</li><li>▪ Lack of standards</li></ul>

## Network Model

Advantages	Disadvantages
<ul style="list-style-type: none"><li>▪ Conceptual simplicity</li><li>▪ Handles more relationship types</li><li>▪ Data access is flexible</li><li>▪ Data owner/member relationship promotes data integrity</li><li>▪ Conformance to standards</li><li>▪ Includes data definition language (DDL) and data manipulation language (DML)</li></ul>	<ul style="list-style-type: none"><li>▪ System complexity limits efficiency</li><li>▪ Navigational system yields complex implementation, application development, and management</li><li>▪ Structural changes require changes in all application programs</li></ul>

## Standard Database Concepts

### Schema

- Conceptual organization of the entire database as viewed by the database administrator

### Subschema

- Portion of the database seen by the application programs that produce the desired information from the data within the database

# Data creation, querying

## Schema data definition language (DDL)

- Enables the database administrator to define the schema components

## Data manipulation language (DML)

- Environment in which data can be managed and is used to work with the data in the database

# Relational model

## The Relational Model

- Based on a relation
  - **Relation** or **table**: Matrix composed of intersecting tuple and attribute
    - **Tuple**: Rows
    - **Attribute**: Columns
- Describes a precise set of data manipulation constructs

# Relational model

Relational Model	
Advantages	Disadvantages
<ul style="list-style-type: none"><li>▪ Structural independence is promoted using independent tables</li><li>▪ Tabular view improves conceptual simplicity</li><li>▪ Ad hoc query capability is based on SQL</li><li>▪ Isolates the end user from physical-level details</li><li>▪ Improves implementation and management simplicity</li></ul>	<ul style="list-style-type: none"><li>▪ Requires substantial hardware and system software overhead</li><li>▪ Conceptual simplicity gives untrained people the tools to use a good system poorly</li><li>▪ May promote information problems</li></ul>

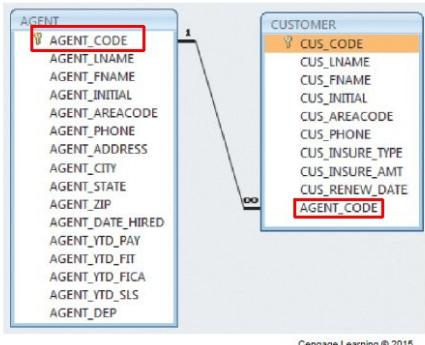
# Relational DBMS

## Relational Database Management System(RDBMS)

- Performs basic functions provided by the hierarchical and network DBMS systems
- Makes the relational data model easier to understand and implement
- Hides the complexities of the relational model from the user

# Relation - BETWEEN entities

Figure 2.2 - A Relational Diagram



Note: this relation is NOT what relational modeling is about!! Here, we relate two entities, via a common attribute (AGENT\_CODE, in our example).

# SQL + RDBMS

## SQL-Based Relational Database Application

- End-user interface
  - Allows end user to interact with the data
- Collection of tables stored in the database
  - Each table is independent from another
  - Rows in different tables are related based on common values in common attributes
- SQL engine
  - Executes all queries

# E-R

## The Entity Relationship Model

- Graphical representation of entities and their relationships in a database structure
- **Entity relationship diagram (ERD)**
  - Uses graphic representations to model database components
- **Entity instance or entity occurrence**
  - Rows in the relational table
- **Connectivity:** Term used to label the relationship types

# E-R

## Entity Relationship Model

### Advantages

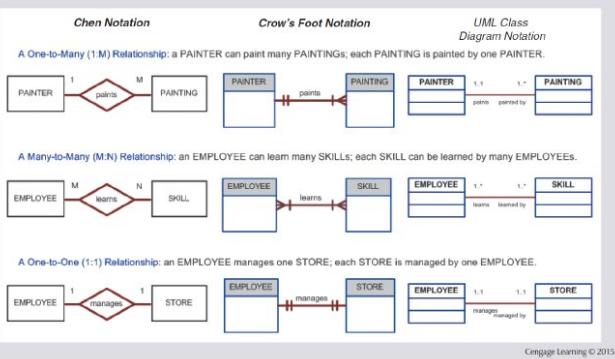
- Visual modeling yields conceptual simplicity
- Visual representation makes it an effective communication tool
- Is integrated with the dominant relational model

### Disadvantages

- Limited constraint representation
- Limited relationship representation
- No data manipulation language
- Loss of information content occurs when attributes are removed from entities to avoid crowded displays

# Notations

Figure 2.3 - The ER Model Notations



# Notations - more..

Additional reading: [here](#) is information on, and comparison between, four ER notations: Chen, Crow, Rein85, IDEFIX.

# O-O databases

Also called 'object stores', these dbs offer(ed) a way to store ("persist") objects on disk. The objects (entity instances) are instanced from classes (entities), like with standard OO programming practice.

Advantages:

- 'cleaner' design - objects mimic real-world counterparts
- inheritance and encapsulation possible
- richer datatypes (attributes) available
- good for CAD, multimedia..

Drawbacks:

- harder to query (compared to relational DBs) - no straightforward way to build and traverse relations between objects
- relations are simpler in certain situations

The RDBMS community collectively ignored this development..

# O-R databases

These are a compromise between RDBs and OODBs - they feature an O-O front-end over a relational architecture. Interfacing applications do so in an O-O way, and queries/modifications are translated to/from relational form ("ORM").

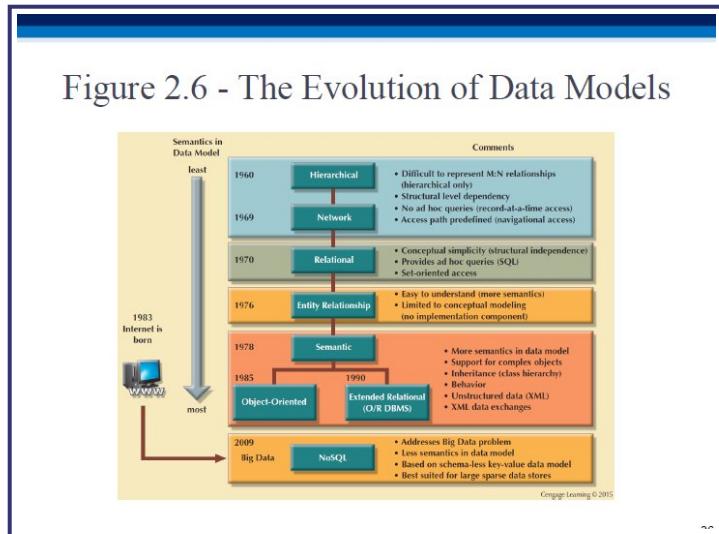
Benefits:

- easy to access the data from an O-O application
- queries can be simpler (can use objects' structure)

Drawback:

- performance can be poor on account of the two-way translation

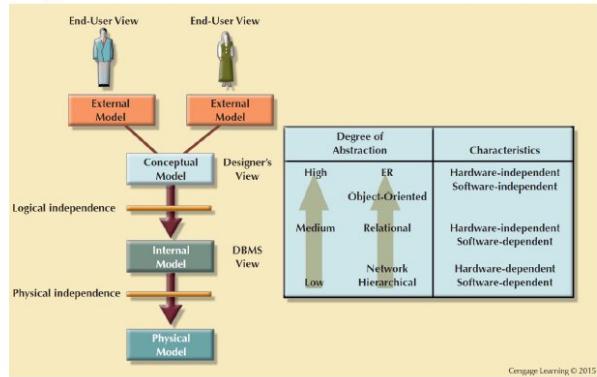
# Data models: hierarchical => => NoSQL



Data models have evolved - from 'hierarchical' (very rigid) to 'NoSQL' (VERY flexible).

# Layered data abstraction

Figure 2.7 - Data Abstraction Levels



# External model

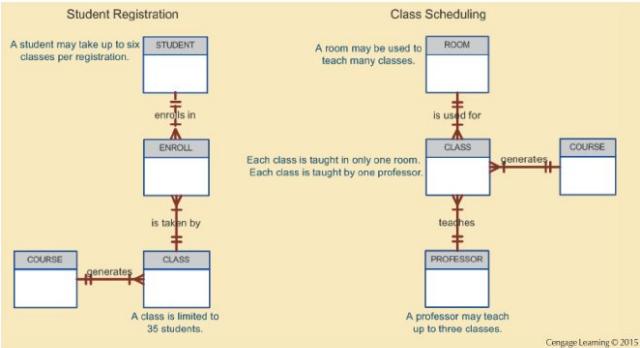
## The External Model

- End users' view of the data environment
- ER diagrams are used to represent the external views
- **External schema:** Specific representation of an external view

An external model is a collection of 'fragmented', 'from the stakeholders' POV', modeling of a database.

# External model

Figure 2.8 - External Models for Tiny College



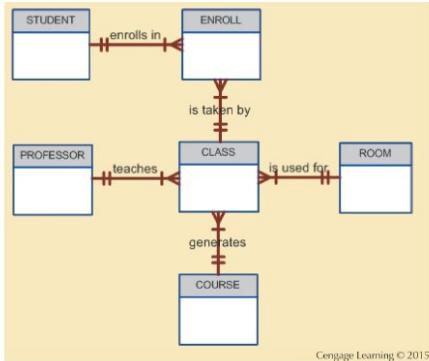
# Conceptual model

## The Conceptual Model

- Represents a global view of the entire database by the entire organization
- **Conceptual schema:** Basis for the identification and high-level description of the main data objects
- Has a macro-level view of data environment
- Is software and hardware independent
- **Logical design:** Task of creating a conceptual data model

# Conceptual model

Figure 2.9 - Conceptual Model for Tiny College



A conceptual model unifies the external views into a cohesive one.

# Internal model

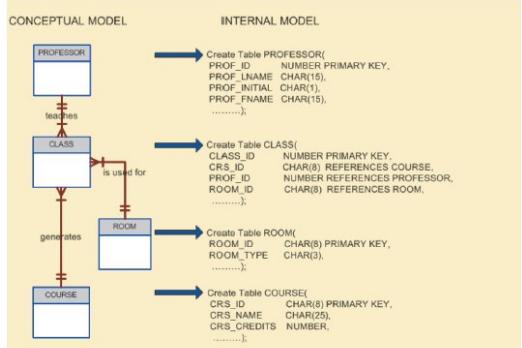
## The Internal Model

- Representing database as seen by the DBMS  
mapping conceptual model to the DBMS
- **Internal schema:** Specific representation of an internal model
  - Uses the database constructs supported by the chosen database
- Is software dependent and hardware independent
- **Logical independence:** Changing internal model without affecting the conceptual model

An internal model specifies what type of modeling (eg. relational, NoSQL...) to use for storing the data.

# Internal model

Figure 2.10 - Internal Model for Tiny College



# Physical model

## The Physical Model

- Operates at lowest level of abstraction
- Describes the way data are saved on storage media such as disks or tapes
- Requires the definition of physical storage and data access methods
- Relational model aimed at logical level
  - Does not require physical-level details
- **Physical independence:** Changes in physical model do not affect internal model

The physical model specifies actual data storage specifics (file format, APIs...).

1/34    11:12:30 \*\*\*



# ER

## Entity Relationship Model (ERM)

- Basis of an entity relationship diagram (ERD)
- ERD depicts the:
  - Conceptual database as viewed by end user
  - Database's main components
    - Entities
    - Attributes
    - Relationships
- Entity - Refers to the entity set and not to a single entity occurrence

## Attributes

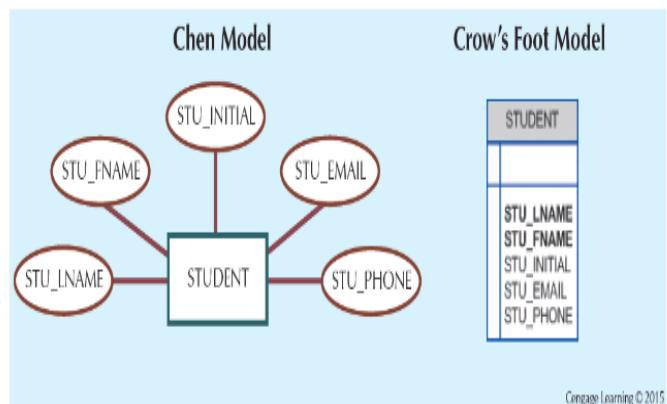
- Characteristics of entities
- **Required attribute:** Must have a value, cannot be left empty
- **Optional attribute:** Does not require a value, can be left empty
- Domain - Set of possible values for a given attribute
- **Identifiers:** One or more attributes that uniquely identify each entity instance

© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

4

An identifier is also called a KEY, or PRIMARY KEY - this is one of the 'key' concepts in all of database theory!! We'll talk much more about keys later.

Figure 4.1 - The Attributes of the Student Entity: Chen and Crow's Foot



Cengage Learning © 2015

© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

5

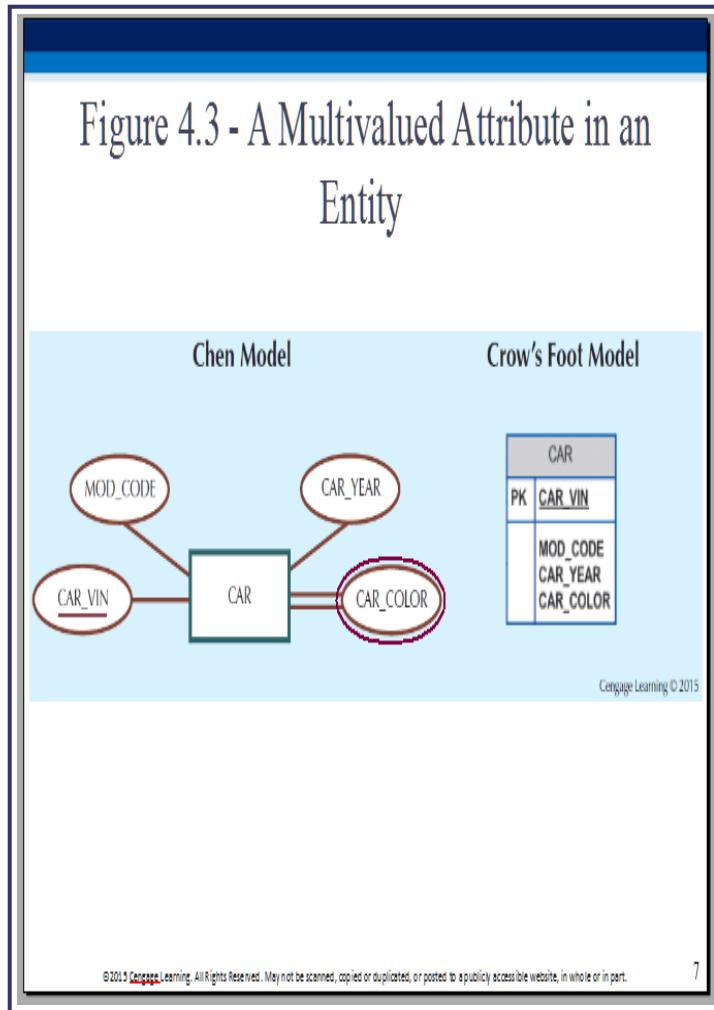
## Attributes

- **Composite identifier:** Primary key composed of more than one attribute
- ~~Compound~~ **Composite attribute:** Attribute that can be subdivided to yield additional attributes
- **Simple attribute:** Attribute that cannot be subdivided
- **Single-valued attribute:** Attribute that has only a single value
- **Multivalued attributes:** Attributes that have many values

© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

6

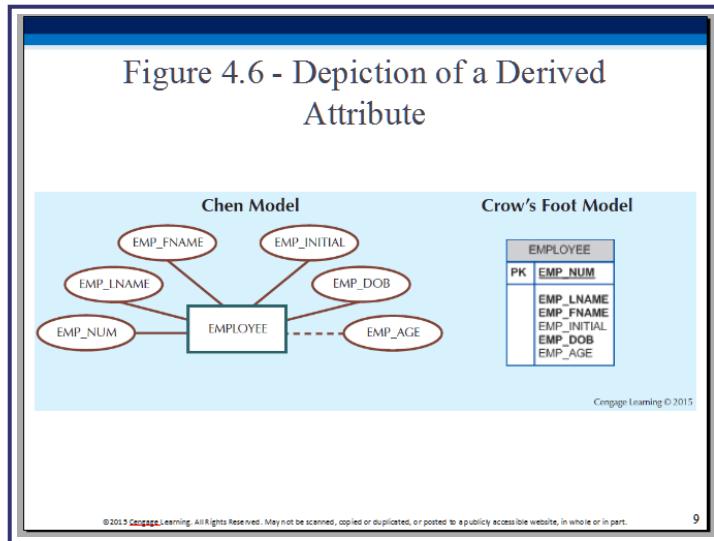
FYI - here is a page on the various types of attributes.



In Crow's Foot notation, 'bold' attributes are 'required' (can't be null).

## Attributes

- **Multivalued attributes:** Attributes that have many values and require creating:
  - Several new attributes, one for each component of the original multivalued attribute
  - A new entity composed of the original multivalued attribute's components
- **Derived attribute:** Attribute whose value is calculated from other attributes
  - Derived using an algorithm



9

Table 4.2 - Advantages and Disadvantages of Storing Derived Attributes

	STORED	NOT STORED
Advantage	Saves CPU processing cycles Saves data access time Data value is readily available Can be used to keep track of historical data	Saves storage space Computation always yields current value
Disadvantage	Requires constant maintenance to ensure derived value is current, especially if any values used in the calculation change	Uses CPU processing cycles Increases data access time Adds coding complexity to queries

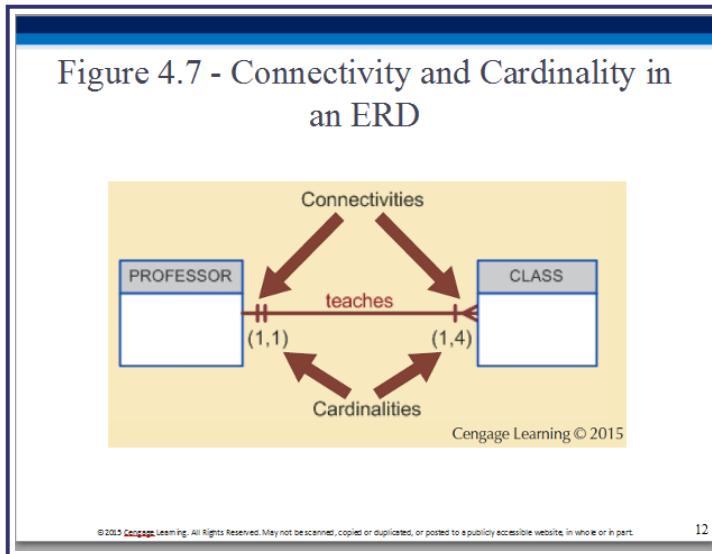
Cengage Learning © 2015

## Relationships

- Association between entities that always operate in both directions
- **Participants:** Entities that participate in a relationship
- **Connectivity:** Describes the relationship classification
- **Cardinality:** Expresses the minimum and maximum number of entity occurrences associated with one occurrence of related entity

©2011 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Connectivity vs cardinality



Connectivity: 1:1, 1:M or M:N (three diff ways by which two entities are related).

Cardinality: (min,max) for 1:1, 1:M or M:N (eg. 1:1 can have (1,0) as its cardinality, 1:M can have (0,4) as its cardinality). Sometimes, min is called 'modality' (and max is cardinality). The 'inside' symbols denotes min, and the outside ones, max.

Confusingly, the # rows in a table is ALSO called table's cardinality (and, # of columns is called the table's degree).

Also confusingly, 1:1, 1:M, M:N are called 'cardinality ratios'!

# 'Can I exist apart from you?'

The diagram is a comparison chart titled 'Existence Dependence' at the top. It features two columns: 'Existence dependence' on the left and 'Existence independence' on the right. The 'Existence dependence' column contains a bulleted list: 'Entity exists in the database only when it is associated with another related entity occurrence'. The 'Existence independence' column also contains a bulleted list: 'Entity exists apart from all of its related entities' and 'Referred to as a **strong entity** or **regular entity**'. At the bottom of the slide, there is a copyright notice: '©2011 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.' and the number '13'.

Existence Dependence	Existence independence
Existence dependence	Existence independence
<ul style="list-style-type: none"><li>Entity exists in the database only when it is associated with another related entity occurrence</li></ul>	<ul style="list-style-type: none"><li>Entity exists apart from all of its related entities</li><li>Referred to as a <b>strong entity</b> or <b>regular entity</b></li></ul>

Existence independence implies a strong entity; but, existence dependence (alone, ie. by itself) does NOT imply a weak entity (there needs to be one more condition, based on 'relationship strength', for it to become 'weak').

In other words, **we need to look at where the FK in the dependent entity is located.**

# Existence dependence

An entity B is "existent dependent" on another entity A, if, a row in B can only exist when its FK is NOT NULL, ie. a corresponding entry exists in A.

Eg. if A is EMPLOYEE and B is DEPENDENT, a dependent (eg. child) in B can only exist if there is a corresponding employee (eg. Dad) in A. THIS ALONE DOES NOT MAKE 'B' A WEAK ENTITY!

# Weak vs strong relationship

Again, it's all about the FK [WHERE it goes], in the dependent entity!

The diagram is titled "Relationship Strength" and contains two main sections: "Weak (non-identifying) relationship" and "Strong (identifying) relationships".

**Weak (non-identifying) relationship**

- Primary key of the related entity does not contain a primary key component of the parent entity

**Strong (identifying) relationships**

- Primary key of the related entity contains a primary key component of the parent entity

At the bottom left, there is a small copyright notice: "© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part." At the bottom right, the number "14" is displayed.

Figure 4.8 - A Weak (Non-Identifying) Relationship between COURSE and CLASS

COURSE	
PK	CRS_CODE
	DEPT_CODE
	CRS_DESCRIPTION
	CRS_CREDIT

CLASS	
PK	CLASS_CODE
FK1	CRS_CODE
	CLASS_SECTION
	CLASS_TIME
	ROOM_CODE
	PROF_NUM

generates

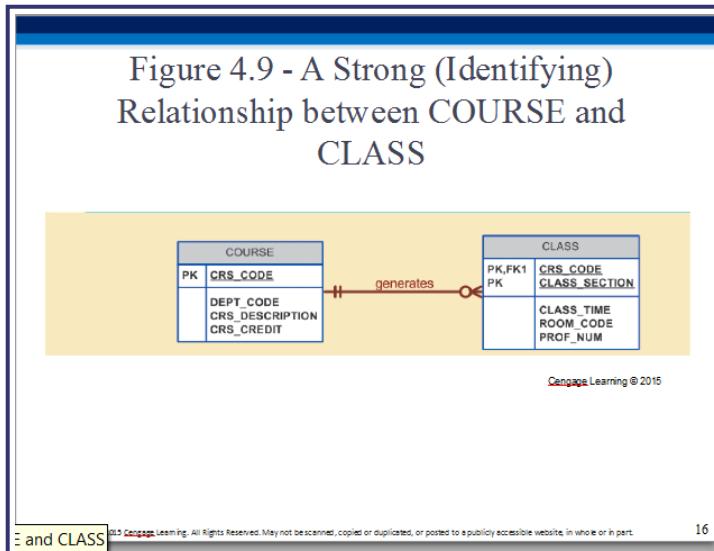
Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

15

So, here, CLASS is \*\*not\*\* a weak entity.

# Strong ("common PK") course-class relation



CLASS is now a weak entity (because: it is existence dependent, AND has a strong relationship).

# Weak entity [two conditions]

## Weak Entity

- Conditions
  - Existence-dependent
  - Has a primary key that is partially or totally derived from parent entity in the relationship
- Database designer determines whether an entity is weak based on business rules

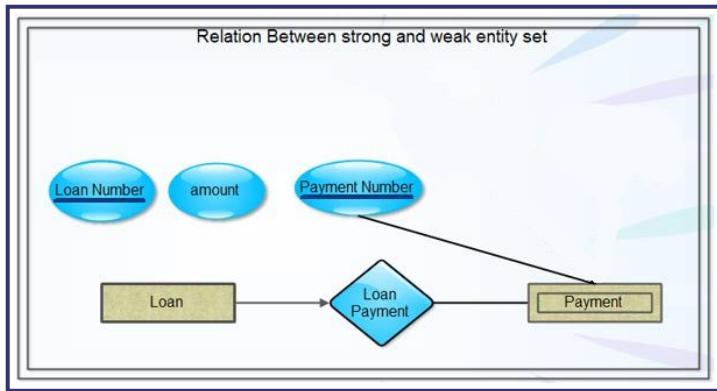
A weak entity needs to satisfy two conditions: existence dependence, strong (identifying/owning) relationship with a parent.

Note that a weak entity implies existence dependence, but existence dependence does not imply a weak entity!

Note too that a weak entity implies a strong ("owning" or "identifying") relationship.

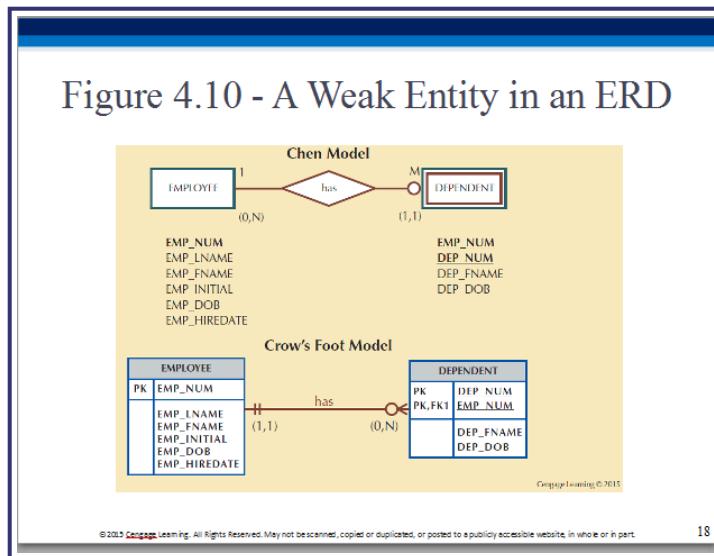
Removing the controlling (owning) entity's key from a weak entity's PK will result in \*\*duplicates\*\* for remaining PK(s) - THAT is what makes it 'weak'.

# Weak entity - example



Payment cannot exist independent of Loan, AND needs  
Loan's key to be part of its own key, so it is a weak entity.

# Weak entity



18

# Weak entity

Figure 4.11 - A Weak Entity

Table name: EMPLOYEE      Database name: Ch04\_ShortCo

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIREDATE
1001	Calliente	Jeanine	J	12-Mar-64	25-May-97
1002	Smithson	William	K	23-Nov-70	28-May-97
1003	Washington	Herman	H	15-Aug-68	26-May-97
1004	Chee	Lydia	B	23-Mar-74	15-Oct-98
1005	Johnson	Melanie		28-Sep-66	20-Dec-98
1006	Ortega	Jorge	G	12-Jul-79	05-Jan-02
1007	O'Donnell	Peter	D	10-Jun-71	23-Jun-02
1008	Brzinski	Barbara	A	12-Feb-70	01-Nov-03

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

19

## Relationship Participation

### Optional participation

- One entity occurrence does not require a corresponding entity occurrence in a particular relationship

### Mandatory participation

- One entity occurrence requires a corresponding entity occurrence in a particular relationship

**Table 4.3 - Crow's Foot Symbols**

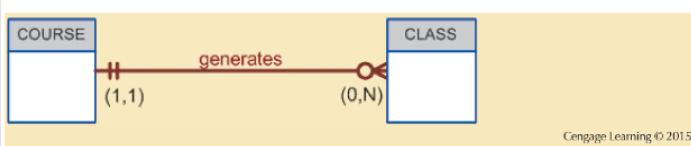
CROW'S FOOT SYMBOLS	CARDINALITY	COMMENT
	(0,N)	Zero or many; the "many" side is optional.
	(1,N)	One or many; the "many" side is mandatory.
	(1,1)	One and only one; the "1" side is mandatory.
	(0,1)	Zero or one; the "1" side is optional.

Cengage Learning © 2015

© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

21

Figure 4.13 - CLASS is Optional to COURSE

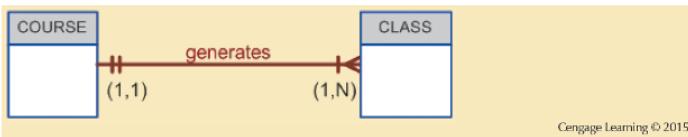


Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

22

Figure 4.14 - COURSE and CLASS in a Mandatory Relationship



Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

23

## Relationship Degree

- Indicates the number of entities or participants associated with a relationship
- **Unary relationship:** Association is maintained within a single entity
  - **Recursive relationship:** Relationship exists between occurrences of the same entity set
- **Binary relationship:** Two entities are associated
- **Ternary relationship:** Three entities are associated

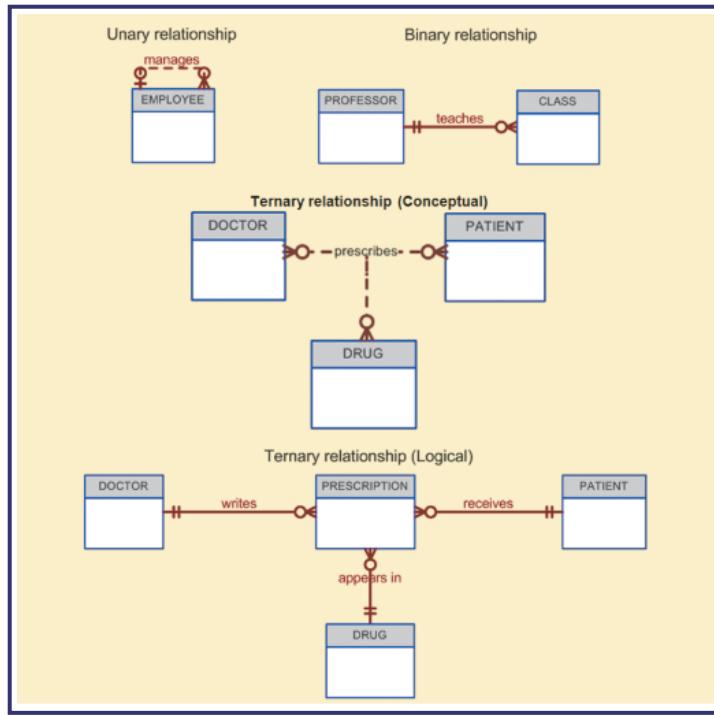
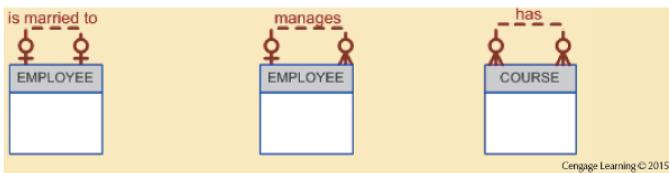


Figure 4.17 - An ER Representation of Recursive Relationships



Cengage Learning © 2015

© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

26

# Bridge entities

## Associative Entities

- Also known as composite or bridge entities
- Used to represent an M:N relationship between two or more entities
- Is in a 1:M relationship with the parent entities
  - Composed of the primary key attributes of each parent entity
- May also contain additional attributes that play no role in connective process

©2012 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

27

Figure 4.23 - Converting the M:N Relationship into Two 1:M Relationships

Table name: STUDENT

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Database name: Ch04\_CollegeTry

Table name: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

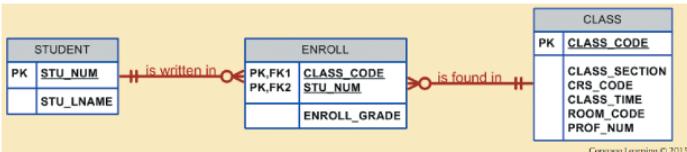
Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

28

Figure 4.25 - A Composite Entity in an ERD



©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

29

# Putting together an ERD

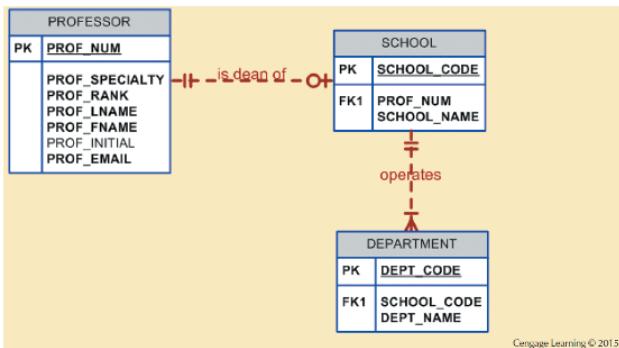
## Developing an ER Diagram

- Create a detailed narrative of the organization's description of operations
- Identify business rules based on the descriptions
- Identify main entities and relationships from the business rules
- Develop the initial ERD
- Identify the attributes and primary keys that adequately describe entities
- Revise and review ERD

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

30

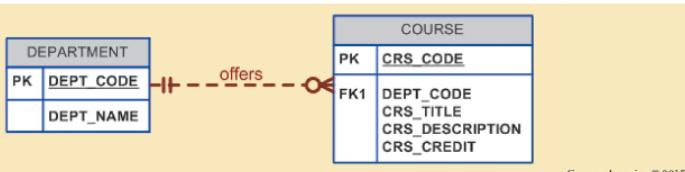
Figure 4.26 - The First Tiny College ERD Segment



©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

31

Figure 4.27 - The Second Tiny College ERD Segment

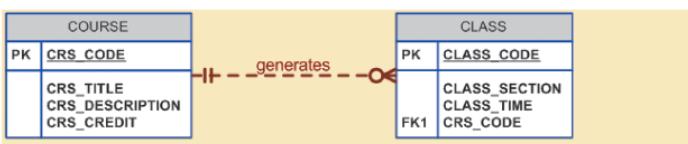


Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

32

Figure 4.28 - The Third Tiny College ERD Segment

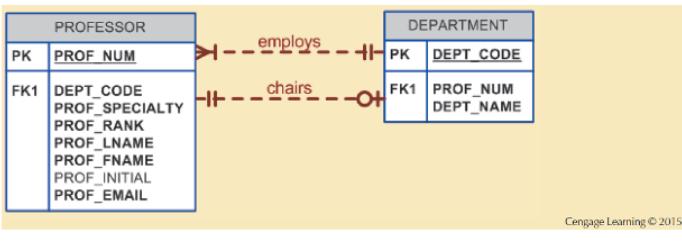


Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

33

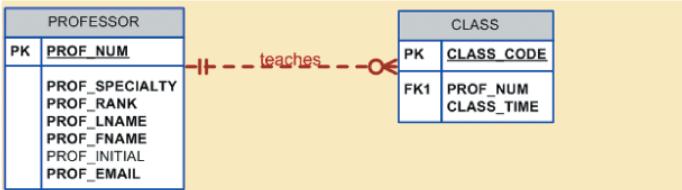
Figure 4.29 - The Fourth Tiny College ERD Segment



Cengage Learning © 2015

34

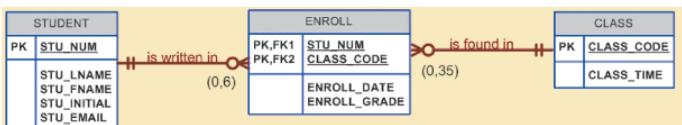
Figure 4.30 - The Fifth Tiny College ERD Segment



Cengage Learning © 2015

35

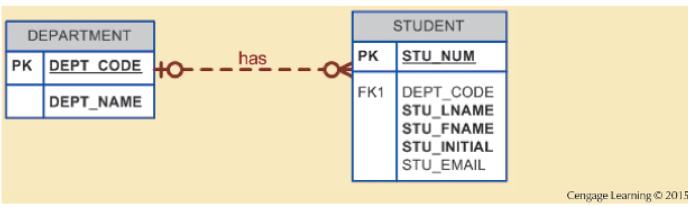
Figure 4.31 - The Sixth Tiny College ERD Segment



Cengage Learning © 2015

36

Figure 4.32 - The Seventh Tiny College  
ERD Segment

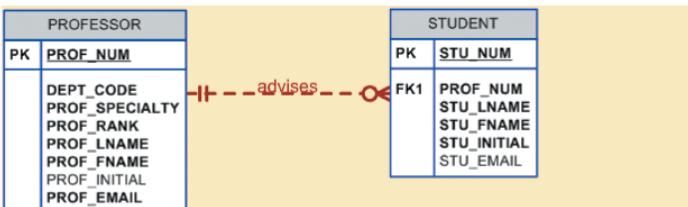


Cengage Learning © 2015

© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

37

Figure 4.33 - The Eighth Tiny College ERD  
Segment

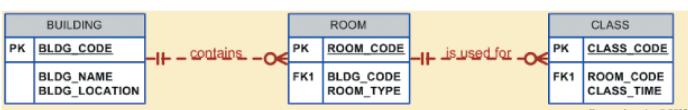


Cengage Learning © 2015

© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

38

Figure 4.34 - The Ninth Tiny College ERD  
Segment



Cengage Learning © 2015

© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

39

# List of entities, relationships, connectivities

Table 4.4 - Components of the ERM

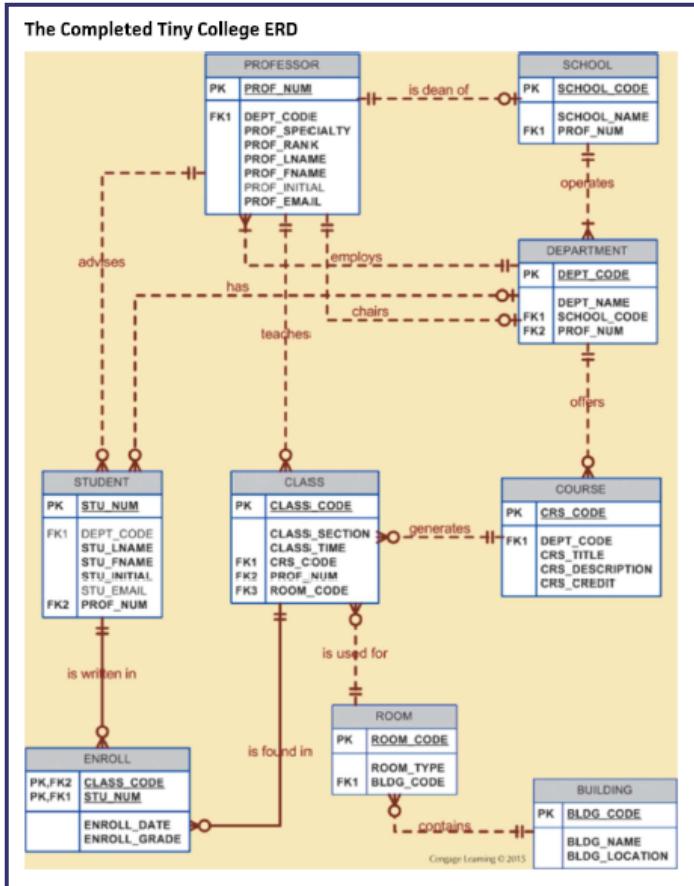
ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrolls in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS

Note: ENROLL is the composite entity that implements the M:N relationship "STUDENT enrolls in CLASS."

Cengage Learning © 2015

# The full schema

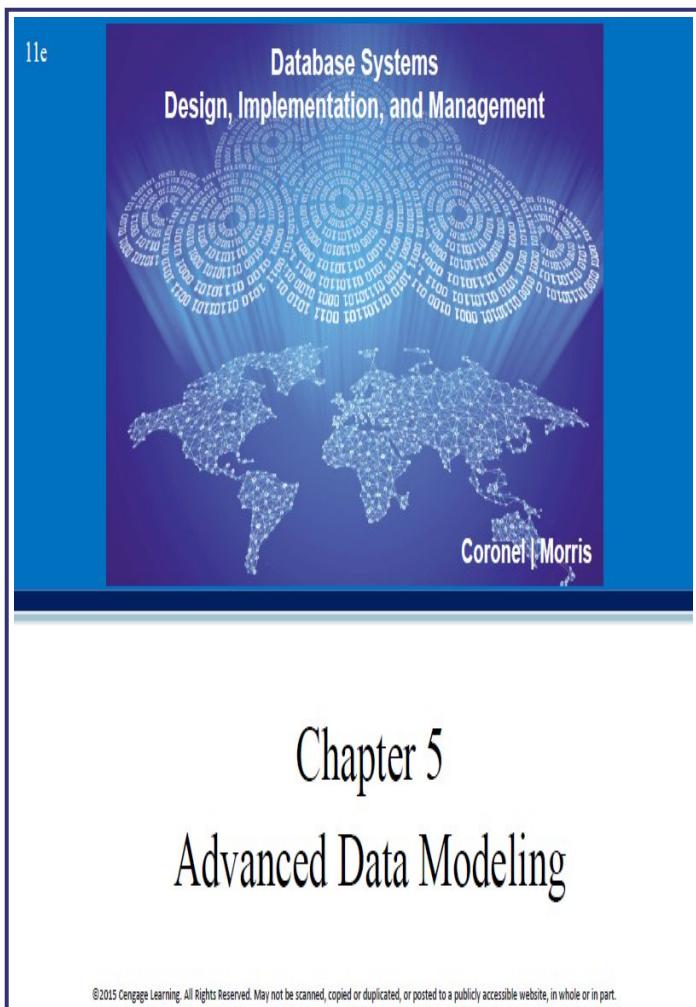
"All together now!"



1/13 11:14:31 \*\*\*



# Extended ER ("EER")



## Chapter 5

### Advanced Data Modeling

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

## Extended Entity Relationship Model (EERM)

- Result of adding more semantic constructs to the original entity relationship (ER) model
- **EER diagram (EERD):** Uses the EER model

## Entity Supertypes and Subtypes

- **Entity supertype:** Generic entity type related to one or more entity subtypes
  - Contains common characteristics
- **Entity subtype:** Contains unique characteristics of each entity subtype
- Criteria to determine the usage
  - There must be different, identifiable kinds of the entity in the user's environment
  - The different kinds of instances should each have one or more attributes that are unique to that kind of instance

## Specialization Hierarchy

- Depicts arrangement of higher-level entity supertypes and lower-level entity subtypes
- Relationships are described in terms of “is-a” relationships
- Subtype exists within the context of a supertype
- Every subtype has one supertype to which it is directly related
- Supertype can have many subtypes

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

5

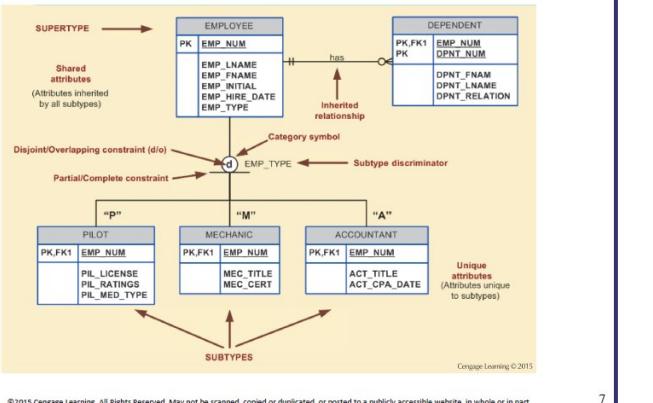
## Specialization Hierarchy

- Provides the means to:
  - Support attribute inheritance
  - Define a special supertype attribute known as the subtype discriminator
  - Define disjoint/overlapping constraints and complete/partial constraints

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

6

Figure 5.2 - Specialization Hierarchy



©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

## Inheritance

- Enables an entity subtype to inherit attributes and relationships of the supertype
- All entity subtypes inherit their primary key attribute from their supertype
- At the implementation level, supertype and its subtype(s) maintain a 1:1 relationship
- Entity subtypes inherit all relationships in which supertype entity participates
- Lower-level subtypes inherit all attributes and relationships from its upper-level supertypes

## Subtype Discriminator

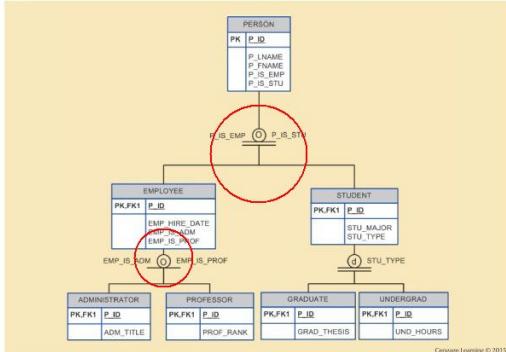
- Attribute in the supertype entity that determines to which entity subtype the supertype occurrence is related
- Default comparison condition is the equality comparison

# Disjoint vs overlapping subtypes

## Disjoint and Overlapping Constraints

- **Disjoint subtypes:** Contain a unique subset of the supertype entity set
  - Known as **nonoverlapping subtypes**
  - Implementation is based on the value of the subtype discriminator attribute in the supertype
- **Overlapping subtypes:** Contain nonunique subsets of the supertype entity set
  - Implementation requires the use of one discriminator attribute for each subtype

Figure 5.4 - Specialization Hierarchy with Overlapping Subtypes



©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

## Completeness Constraint

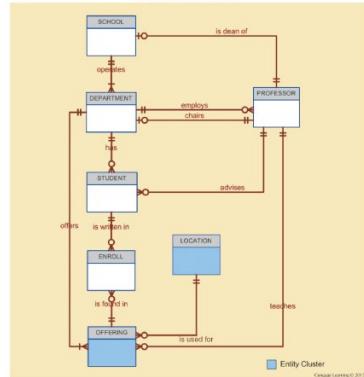
- Specifies whether each supertype occurrence must also be a member of at least one subtype
- Types
  - **Partial completeness:** Not every supertype occurrence is a member of a subtype
  - **Total completeness:** Every supertype occurrence must be a member of any

**Table 5.2 - Specialization Hierarchy Constraint Scenarios**

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique. 
Total 	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique. 

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Figure 5.5 - Tiny College ERD Using Entity Clusters



©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

1/35    11:14:53 \*\*\*



# Relational Modeling

# Logical view: 'relation'

## A Logical View of Data

- Relational database model enables logical representation of the data and its relationships
- Logical simplicity yields simple and effective database design methodologies
- Facilitated by the creation of data relationships based on a logical construct called a relation

# Relational tables

Table 3.1 - Characteristics of a Relational Table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row ( <b>tuple</b> ) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each intersection of a row and column represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the <b>attribute domain</b> .
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or combination of attributes that uniquely identifies each row.

Cengage Learning © 2015

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

4

# Keys

## Keys

- Consist of one or more attributes that determine other attributes
- Used to:
  - Ensure that each row in a table is uniquely identifiable
  - Establish relationships among tables and to ensure the integrity of the data
- **Primary key (PK):** Attribute or combination of attributes that uniquely identifies any given row

# "determines"

## Determination

- State in which knowing the value of one attribute makes it possible to determine the value of another
- Is the basis for establishing the role of a key
- Based on the relationships among the attributes

# Determinants determine dependents [via] dependencies :)

Dependencies

- **Functional dependence:** Value of one or more attributes determines the value of one or more other attributes
  - **Determinant:** Attribute whose value determines another
  - **Dependent:** Attribute whose value is determined by the other attribute
- **Full functional dependence:** Entire collection of attributes in the determinant is necessary for the relationship

© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

7

'Full' functional dependence is a "good" thing.

# Functional dependency

STU\_ID[determinant]  $\rightarrow$  [functionally determines]  
STU\_LNAME[dependent]

STU\_ID,STU\_LNAME  $\rightarrow$  GPA is NOT a 'full functional dependency' because the determinant contains an extra (unwanted) attr (STU\_LNAME)

STU\_LNAME,STU\_FNAME  $\rightarrow$  GPA is a 'full functional dependency' (assuming lastname,firstname is unique)

Solemnly swear: "The key, the whole key, and nothing but the key, so help me Codd." :) :)

# Composite key; entity integrity

## Types of Keys

- **Composite key:** Key that is composed of more than one attribute
- **Key attribute:** Attribute that is a part of a key
- **Entity integrity:** Condition in which each row in the table has its own unique identity
  - All of the values in the primary key must be unique
  - No key attribute in the primary key can contain a null

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

8

A table 'cannot not' have entity integrity!!

# Nulls; referential integrity

## Types of Keys

- **Null:** Absence of any data value that could represent:
  - An unknown attribute value
  - A known, but missing, attribute value
  - A inapplicable condition
- **Referential integrity:** Every reference to an entity instance by another entity instance is valid

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

9

Whereas entity integrity has to do with a single table, referential integrity relates to two tables (loosely, 'don't allow invalid pointers').

# Types (categories) of keys

Table 3.3 - Relational Database Keys

KEY TYPE	DEFINITION
Superkey	An attribute or combination of attributes that uniquely identifies each row in a table
Candidate key	A minimal (irreducible) superkey; a superkey that does not contain a subset of attributes that is itself a superkey
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row; cannot contain null entries
Foreign key	An attribute or combination of attributes in one table whose values must either match the primary key in another table or be null
Secondary key	An attribute or combination of attributes used strictly for data retrieval purposes

Cengage Learning © 2015

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

10

# Keys: many types

- \* primary (foreign) keys are a subset of candidate keys are a subset of superkeys (note - superkeys could be 'wasteful', ie. contain superfluous, non-needed attrs)
- \* simple keys vs compound keys vs composite keys
- \* natural keys - keys that are created from real-world entities (eg. for a US resident, their SSN could be a natural key)
- \* surrogate keys (just make up brand new unique keys)
- \* secondary, or 'alternate' keys

You can read a bit more keys [here](#).

# Example relation

**Figure 3.2 - An Example of a Simple Relational Database**

Table name: PRODUCT      Database name: Ch03\_SaleCo  
 Primary key: PROD\_CODE  
 Foreign key: VEND\_CODE

PROD_CODE	PROD_DESCRPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UY	Houselite chain saw, 16-in. bar	189.99	4	235
GER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR  
 Primary key: VEND\_CODE  
 Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

Cengage Learning © 2015

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

# Nulls - avoid where possible!

## Ways to Handle Nulls

- **Flags:** Special codes used to indicate the absence of some value
- NOT NULL constraint - Placed on a column to ensure that every row in the table has a value for that column
- UNIQUE constraint - Restriction placed on a column to ensure that no duplicate values exist for that column

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

15

In RL, NULLs can't be entirely avoided (look [here](#), for 'interpreted as any of the following').

# Relational 'algebra' [fun with one, two or more tables]

## Relational Algebra

- Theoretical way of manipulating table contents using relational operators
- **Relvar:** Variable that holds a relation
  - Heading contains the names of the attributes and the body contains the relation
- Relational operators have the property of closure
  - **Closure:** Use of relational algebra operators on existing relations produces new relations

© 2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

16

What if a table were a datatype (similar to an int, Vec3D, ComplexNumber, etc)?! Specifically, what operations could be performed on them (eg. similar to addition, square root on doubles)?!

# Operations on tables [table(s) in, table out, ie. "closure"]

There are (only) EIGHT 'relational set operators' (defined by Ed Codd, at IBM, in 1970), which are all used to operate ("perform relational algebra") on tables: Select, Project, Union, Intersect, Difference, Product, Join, Divide.

This is no exaggeration: these operators are the basis for SQL and the entire relational DB industry!

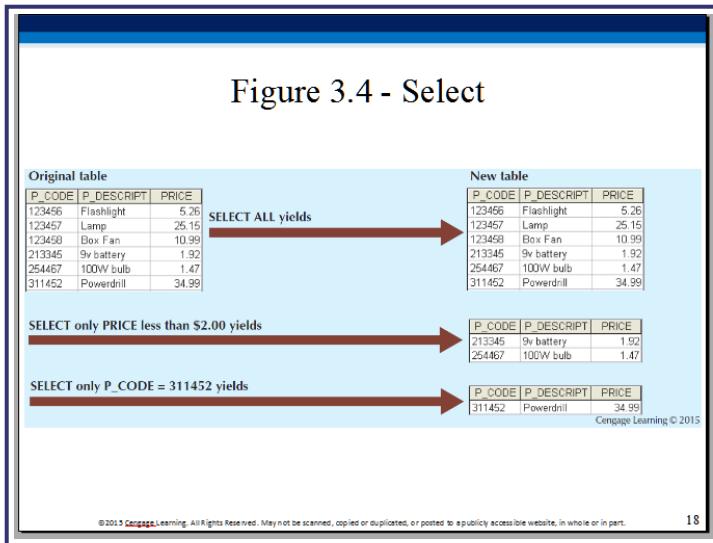
# SELECT; PROJECT; UNION; INTERSECT

## Relational Set Operators

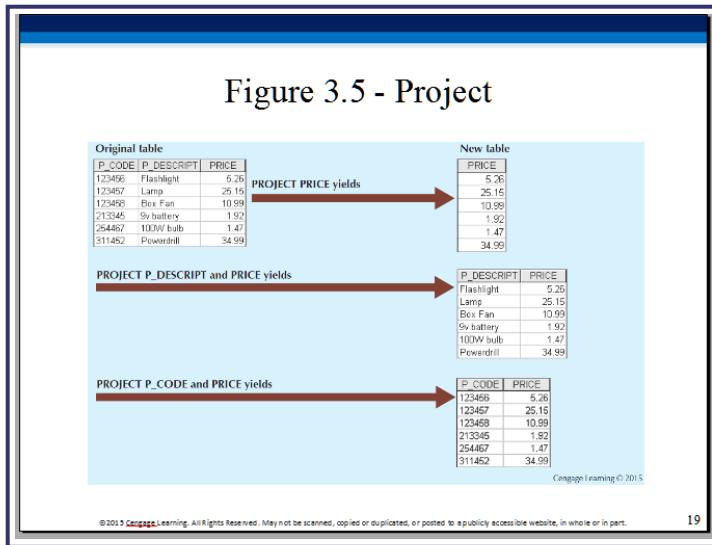
- Select (Restrict)**
  - Unary operator that yields a horizontal subset of a table
- Project**
  - Unary operator that yields a vertical subset of a table
- Union**
  - Combines all rows from two tables, excluding duplicate rows
  - **Union-compatible:** Tables share the same number of columns, and their corresponding columns share compatible domains
- Intersect**
  - Yields only the rows that appear in both tables
  - Tables must be union-compatible to yield valid results

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

# SELECT [outputs a subset of rows]



# PROJECT [outputs a subset of cols]



# UNION [eqvt to 'cat a b > c']

Figure 3.6 - Union

The diagram illustrates the result of a UNION query. On the left, two separate tables are shown: one with 6 rows and another with 3 rows. An arrow labeled "yields" points from the UNION operation to the resulting table on the right, which contains all 9 rows from both tables combined.

P_CODE	P_DESCRPT	PRICE
123456	Flashlight	5.20
123457	Lamp	25.15
123459	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

UNION

P_CODE	P_DESCRPT	PRICE
345670	Microwave	160.00
345679	Dishwasher	500.00

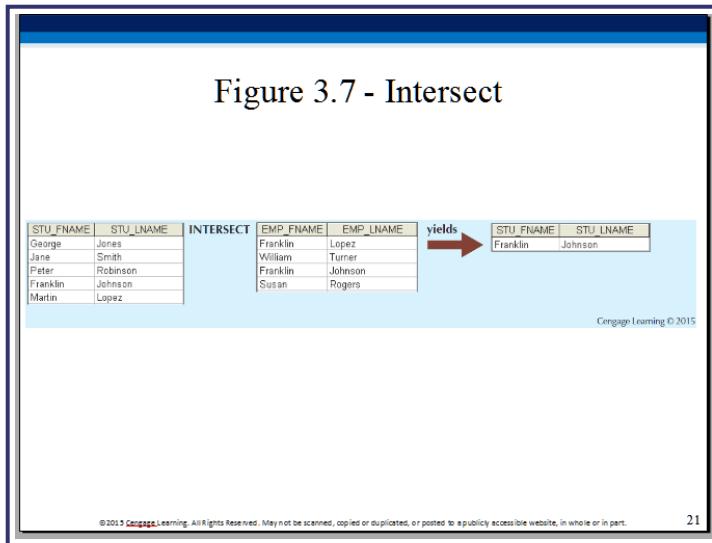
yields

P_CODE	P_DESCRPT	PRICE
123456	Flashlight	5.20
123457	Lamp	25.15
123459	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99
345670	Microwave	160.00
345679	Dishwasher	500.00

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

20

# INTERSECT [rows common to a and b]



# Difference; Product

## Relational Set Operators

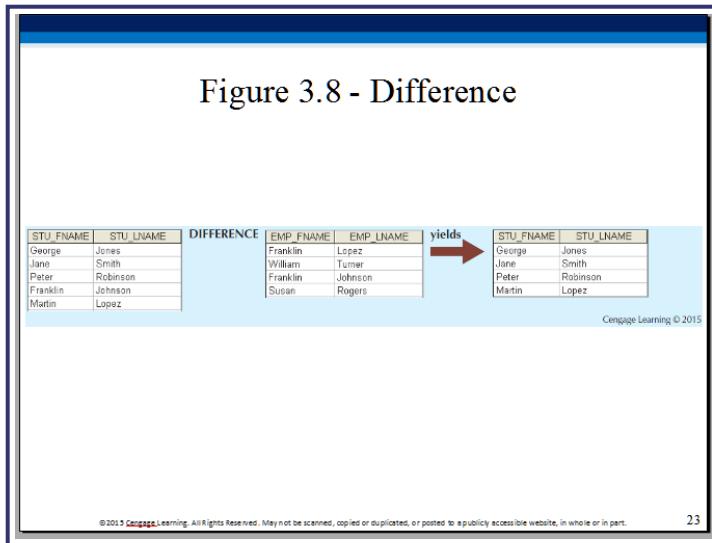
### ▪ Difference

- Yields all rows in one table that are not found in the other table
- Tables must be union-compatible to yield valid results

### ▪ Product

- Yields all possible pairs of rows from two tables

# Difference [a - b]



# Product [multiply rows, add columns]

Figure 3.9 - Product

The diagram illustrates the concept of multiplying rows and adding columns. On the left, a table titled "PRODUCT" shows five items: Flashlight, Lamp, Box Fan, 9v battery, and Powerdrill. Each item has a unique product code, description, price, and is located in a specific store (W, K, Z), aisle (5, 9, 6), and shelf (5, 9, 6). An arrow labeled "yields" points to the right, where a larger table shows the result of multiplying each row by the number of stores, aisles, and shelves. This results in 15 rows for each item, with columns for P\_CODE, P\_DESCRPT, PRICE, STORE, AISLE, and SHELF. The expanded table contains 75 rows in total.

P_CODE	P_DESCRPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.35	23	W	5
123457	Lamp	25.15	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92			
254467	100W bulb	1.47			
311452	Powerdrill	34.99			

P_CODE	P_DESCRPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.35	23	W	5
123456	Flashlight	5.35	24	K	9
123456	Flashlight	5.35	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

Cengage Learning © 2015

24

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# JOIN (several kinds); DIVIDE (?!)

## Relational Set Operators

### ▪ Join

- Allows information to be intelligently combined from two or more tables

### ▪ Divide

- Uses one 2-column table as the dividend and one single-column table as the divisor
- Output is a single column that contains all values from the second column of the dividend that are associated with every row in the divisor

# JOIN

## Types of Joins

- **Natural join:** Links tables by selecting only the rows with common values in their common attributes
  - **Join columns:** Common columns
- **Equijoin:** Links tables on the basis of an equality condition that compares specified columns of each table
- **Theta join:** Extension of natural join, denoted by adding a theta subscript after the JOIN symbol

# JOIN [cont'd]

## Types of Joins

- **Inner join:** Only returns matched records from the tables that are being joined
- **Outer join:** Matched pairs are retained and unmatched values in the other table are left null
  - **Left outer join:** Yields all of the rows in the first table, including those that do not have a matching value in the second table
  - **Right outer join:** Yields all of the rows in the second table, including those that do not have matching values in the first table

# Tables to illustrate JOIN operations

Figure 3.10 - Two Tables That Will Be Used in JOIN Illustrations

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	vWalker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: CUSTOMER

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

Table name: AGENT

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

29

# Natural join

A natural join links tables by selecting from two tables, only those rows that have common (identical) values for common attributes.

These three steps result in a natural join: create product, select, project.

# Natural join: product

Cartesian product of the two tables (product of rows, juxtaposition of columns):

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER	AGENT_CODE	AGENT	AGENT_PHONE
1132445	Walker	32145	231	125	6152439987	
1132445	Walker	32145	231	167	6153426778	
1132445	Walker	32145	231	231	6152431124	
1132445	Walker	32145	231	333	9041234445	
1217782	Adares	32145	125	125	6152439987	
1217782	Adares	32145	125	167	6153426778	
1217782	Adares	32145	125	231	6152431124	
1217782	Adares	32145	125	333	9041234445	
1312243	Rakowski	34129	167	125	6152439987	
1312243	Rakowski	34129	167	167	6153426778	
1312243	Rakowski	34129	167	231	6152431124	
1312243	Rakowski	34129	167	333	9041234445	
1321242	Rodriguez	37134	125	125	6152439987	
1321242	Rodriguez	37134	125	167	6153426778	
1321242	Rodriguez	37134	125	231	6152431124	
1321242	Rodriguez	37134	125	333	9041234445	
1542311	Smithson	37134	421	125	6152439987	
1542311	Smithson	37134	421	167	6153426778	
1542311	Smithson	37134	421	231	6152431124	
1542311	Smithson	37134	421	333	9041234445	
1657399	Vanloo	32145	231	125	6152439987	
1657399	Vanloo	32145	231	167	6153426778	
1657399	Vanloo	32145	231	231	6152431124	
1657399	Vanloo	32145	231	333	9041234445	

Cengage Learning © 2015

# Natural join: select

Select only rows with identical values in the common (joining) columns:

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER_AGENT_CODE	AGENT_AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124

Cengage Learning © 2015

# Natural join: project

Project away, ie. remove one of the two duplicate columns:

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

Cengage Learning © 2015

Result (the table above): natural join.

# Left outer join

Output all rows of the left (CUSTOMER) table, including ones for which there are no matching values in the join column in the other (AGENT) table:

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER_AGENT_CODE	AGENT_AGENT_CODE	AGENT_PHONE
1217782	Adrees	32145	125	125	6152438887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	WWalker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
1542311	Smithson	37134	421		

Cengage Learning © 2015

Note that an outer join is an "inner join plus" [it is NOT an opposite of inner join].

# Right outer join, full outer join

Output all rows of the right (AGENT) table, including ones for which there are no matching values in the join column in the other (CUSTOMER) table:

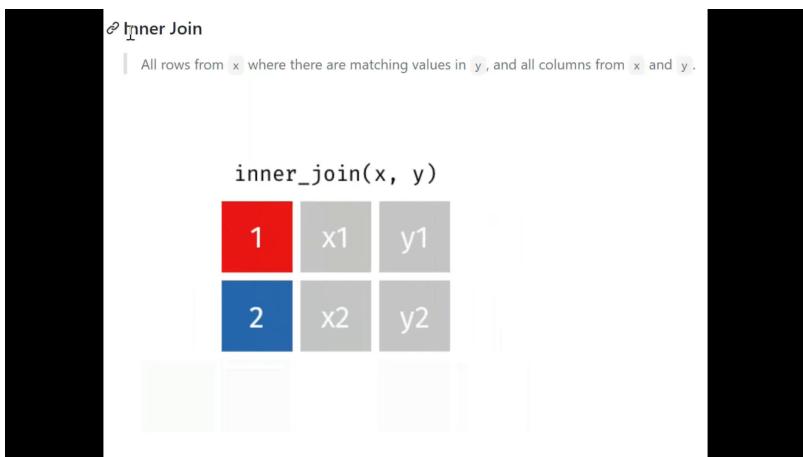
CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER_AGENT_CODE	AGENT_AGENT_CODE	AGENT_PHONE
1217782	Adanes	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439897
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
				333	9041234445

Cengage Learning © 2015

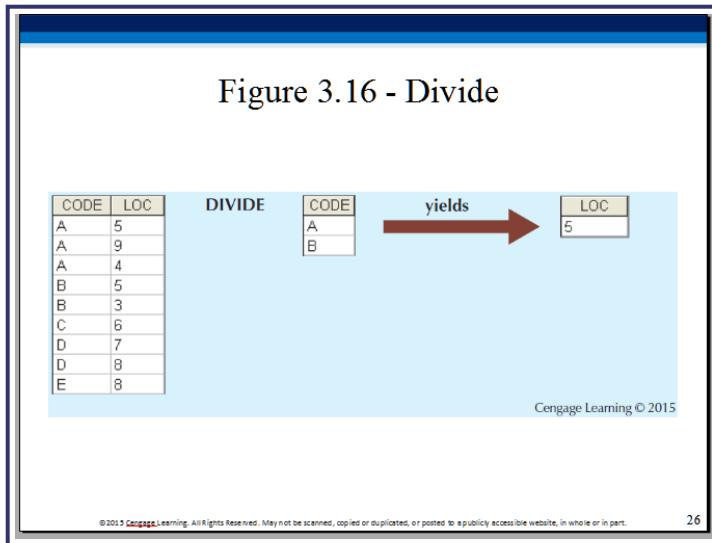
Outer joins are useful in exposing missing information [in our example, customers who don't seem to have an agent, and, agents who don't seem to have customers].

A 'full outer join' is a union of left outer join and right outer join - output all the rows from both tables, including ones for which there are no matches in the other table - this could result in nulls on the left side of some rows, as well as nulls on the right side of others.

This clip shows the various types of joins [thanks, Yash Gupta, for sending this]:



# DIVIDE



We're dividing by A and B in the divisor (bottom) table. There's (A,5) and (B,5) in the dividend (top) table, so we output 5 as the result; if the dividend were to contain (A,9) and (B,9) also, then we'd output 5 9 as the result.

# Dictionaries [hold metadata]

## Data Dictionary and the System Catalog

- **Data dictionary:** Description of all tables in the database created by the user and designer
- **System catalog:** System data dictionary that describes all objects within the database
- Homonyms and synonyms must be avoided to lessen confusion
  - **Homonym:** Same name is used to label different attributes
  - **Synonym:** Different names are used to describe the same attribute

©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

30

A data dictionary is metadata about tables (only); a system catalog, that includes (is a superset of, although confusingly, the two are conflated in RL) the data dictionary, and more.

1/30 11:16:17 \*\*\*



# (De/)Normalization



# On tap

## Learning Objectives

- In this chapter, students will learn:
  - What normalization is and what role it plays in the database design process
  - About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
  - How normal forms can be transformed from lower normal forms to higher normal forms
  - That normalization and ER modeling are used concurrently to produce a good database design
  - That some situations require denormalization to generate information efficiently

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# The goal of normalization

Loosely speaking:



[<http://rovingcrafters.com/>]

# Goal: reduce redundancies, anomalies

## Normalization

- Evaluating and correcting table structures to minimize data redundancies
- Reduces data anomalies
- Assigns attributes to tables based on determination
- Normal forms
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Higher normal forms → cleaner designs

## Normalization

- Structural point of view of normal forms
  - Higher normal forms are better than lower normal forms
- Properly designed 3NF structures meet the requirement of fourth normal form (4NF)
- **Denormalization:** Produces a lower normal form
  - Results in increased performance and greater data redundancy

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Normalization is a design step

## Need for Normalization

- Used while designing a new database structure
  - Analyzes the relationship among the attributes within each entity
  - Determines if the structure can be improved
- Improves the existing data structure and creates an appropriate database design

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# A construction company db

Employees of the construction company work on projects. Each employee has an ID, name, job title and corresponding hourly rate.

Each project has a number, name and assigned employees. An employee can be assigned to more than one project.

The company bills clients for projects, based on hours worked by employees.

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	46.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
		105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.6
		107	Maria D. Alonso	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.0
		101	John G. News *	Database Designer	105.00	66.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Copyright © Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

# Report

The construction company periodically generates a report like so:

PROJECT NUMBER	PROJECT NAME	EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CLASS	CHARGE/HOUR	HOURS BILLED	TOTAL CHARGE
15	Evergreen	103	Judy E. Arbough	Elec. Engineer	\$ 84.50	23.8	\$ 2,011.10
		101	John G. News *	Database Designer	\$ 105.00	19.4	\$ 2,045.00
		105	Alice K. Johnson *	Database Designer	\$ 105.00	35.7	\$ 3,748.50
		106	William Smithfield	Programmer	\$ 35.75	11.6	\$ 405.45
		102	David K. Senior	Systems Analyst	\$ 96.75	23.8	\$ 2,302.65
				<b>Subtotal</b>			<b>\$10,549.70</b>
18	Amber Wave	114	Annelise Jones	Applications Designer	\$ 48.10	24.6	\$ 1,183.26
		118	James J. Frommer	General Support	\$ 18.36	45.3	\$ 831.71
		104	Annie K. Rameiras *	Systems Analyst	\$ 96.75	32.4	\$ 3,134.70
		112	Dariene M. Smithson	DSS Analyst	\$ 45.95	44.0	\$ 2,021.80
				<b>Subtotal</b>			<b>\$ 7,171.47</b>
22	Rolling Tide	105	Alice K. Johnson	Database Designer	\$ 105.00	64.7	\$ 6,793.50
		104	Annie K. Rameiras	Systems Analyst	\$ 96.75	48.4	\$ 4,668.70
		113	Delbert K. Jembroid *	Applications Designer	\$ 48.10	21.6	\$ 1,135.16
		111	Geoff B. Wahash	Clerical Support	\$ 26.87	22.0	\$ 591.14
		106	William Smithfield	Programmer	\$ 35.75	12.8	\$ 457.60
				<b>Subtotal</b>			<b>\$13,660.10</b>
25	Starlight	107	Maria D. Alonso	Programmer	\$ 35.75	24.6	\$ 879.45
		115	Travis B. Bewangi	Systems Analyst	\$ 96.75	45.8	\$ 4,431.15
		101	John G. News *	Database Designer	\$ 105.00	56.3	\$ 5,911.50
		114	Annelise Jones	Applications Designer	\$ 48.10	33.1	\$ 1,551.11
		108	Ralph B. Washington	Systems Analyst	\$ 96.75	23.6	\$ 2,283.30
		118	James J. Frommer	General Support	\$ 18.36	10.3	\$ 185.98
		112	Dariene M. Smithson	DSS Analyst	\$ 45.95	41.4	\$ 1,902.33
				<b>Subtotal</b>			<b>\$17,559.82</b>
				<b>Total</b>			<b>\$48,941.09</b>

Note: A \* indicates the project leader.

Cengage Learning © 2015

# Issues with our db

Here is our table again:

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.9
18	Amber Wave	114	Anneise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.6
25	Starflight	107	Maria D. Alonso	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.0
		101	John G. News *	Database Designer	105.00	56.3
		114	Anneise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

There are numerous issues:

- the PROJ\_NUM attr could be used as a PK (or part of a PK, along with PROJ\_NAME) but it contains nulls
- possibilities for data inconsistencies exist, eg. if someone's name or title is misspelled
- the redundancies that exist, could lead to insertion anomalies (eg. a new employee needs to be assigned to some project, even a fake one), update anomalies (eg. if an employee's JOB\_CLASS changes, it has to be modified multiple times), deletion anomalies (eg. if a project has just one employee and that employee leaves, deleting the lone employee record would lead to the project itself getting deleted!)
- data redundancy leads to wasted storage space

We have 'repeating groups' (for each project, we list all details about each employee) - our table is un-normalized, ie. is in '1NF' :)

So, we need to clean up the design!

# Objectives: what do we want?

## Normalization Process

- Objective is to ensure that each table conforms to the concept of well-formed relations
  - Each table represents a single subject
  - No data item will be unnecessarily stored in more than one table **wholly; nothing but**
  - All nonprime attributes in a table are dependent  on the primary key
  - Each table is void of insertion, update, and deletion anomalies

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Normal forms

Normalization is a systematic process that yields progressively higher 'normal forms' (NFs) for each entity (table) in our db. We want **at least** 3NF for each table; in RL, we stop **at** 3NF.

Table 6.2 - Normal Forms

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	6.3.1
Second normal form (2NF)	1NF and no partial dependencies	6.3.2
Third normal form (3NF)	2NF and no transitive dependencies	6.3.3
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	6.6.1
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	6.6.2

Cengage Learning © 2015

# The process

## Normalization Process

- Ensures that all tables are in at least 3NF
- Higher forms are not likely to be encountered in business environment
- Works one relation at a time
- Starts by:
  - Identifying the dependencies of a relation (table)
  - Progressively breaking the relation into new set of relations

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Normalization how-to, in one sentence: **work on one relation (table) at a time: identify dependencies, then 'normalize' - progressively break it down into smaller relations (tables), based on the dependencies we identify in the original relation so that "only the PK, the whole PK and nothing but the PK" acts as a determinant!** But how?? Details follow..

# Functional dependence, determination

## Functional Dependence Concepts

Concept	Definition
Functional dependence	The attribute B is fully functionally dependent on the attribute A if each value of A determines one and only one value of B.
Functional dependence (Generalized definition)	Attribute A determines attribute B if all of the rows in the table that agree in value for attribute A also agree in value for attribute B.
Fully functional dependence (composite key)	If attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A.

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Partial dependency, transitive dependency

## Types of Functional Dependencies

- **Partial dependency:** Functional dependence in which the determinant is only part of the primary key
  - Assumption - One candidate key
  - Straight forward
  - Easy to identify
- **Transitive dependency:** An attribute functionally depends on another nonkey attribute

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

If  $(A,B)$  is a primary key, we have **partial** dependence if  $(A,B) \rightarrow (C,D)$  and  $B \rightarrow C$  [ $C$  is only partially dependent on the PK, ie. we only need  $B$  to determine  $C$ ]. In other words, a part of an existing PK is acting like a PK on its own.

If  $X$  is a primary key, we have a **transitive** dependency if  $X \rightarrow Y$  and  $Y \rightarrow Z$  [ $Z$  is transitively dependent on  $X$ , not directly so]. In other words, a non-PK (regular attr) is acting like a PK.

# 0NF->1NF: eliminate repeating groups

## Conversion to First Normal Form

- **Repeating group:** Group of multiple entries of same type can exist for any single key attribute occurrence
  - Existence proves the presence of data redundancies
- Enable reducing data redundancies
- Steps
  - Eliminate the repeating groups
  - Identify the primary key
  - Identify all dependencies

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

In other words, "fill in the blanks" so that there are no nulls.  
Now we have a relation (table), with a value in each cell.

Further, identify the PK! In our example, it is  
(PROJ\_NUM,EMP\_NUM).

# 0NF->1NF [cont'd]

## Conversion to First Normal Form

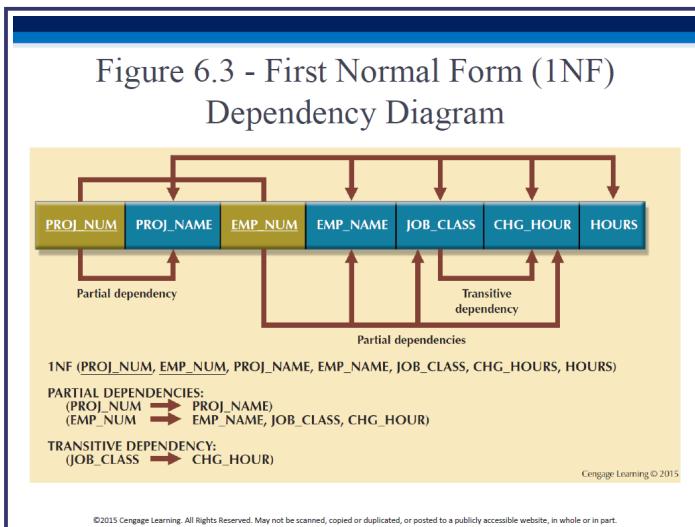
- **Dependency diagram:** Depicts all dependencies found within given table structure
  - Helps to get an overview of all relationships among table's attributes
  - Makes it less likely that an important dependency will be overlooked

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Create a **dependency diagram**, showing relationships (dependencies) between the attributes - this will help us systematically normalize the table.

# Dependency diagram

Indicate full dependencies on the top, and partial and transitive dependencies on the bottom. "Top good, bottom bad". Also, color the PK components in a different color (and underline them). Result:



PROJ\_NAME has only a partial dependency on the PK (since it is only dependent on PROJ\_NUM, which is just a part of the PK).

CHG\_HOUR is dependent on JOB\_CLASS, which is a non-prime attribute that is itself dependent on EMP\_NUM. So  $\text{JOB\_CLASS} \rightarrow \text{CHG\_HOUR}$  is a signaling dependency, indicating a  $\text{EMP\_NUM} \rightarrow \text{CHG\_HOUR}$  transitive dependency.

# 0NF->1NF [cont'd]

## Conversion to First Normal Form

- 1NF describes tabular format in which:
  - All key attributes are defined
  - There are no repeating groups in the table
  - All attributes are dependent on the primary key
- All relational tables satisfy 1NF requirements
- Some tables contain partial dependencies
  - Subject to data redundancies and various anomalies

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 1NF->2NF: remove partial dependencies

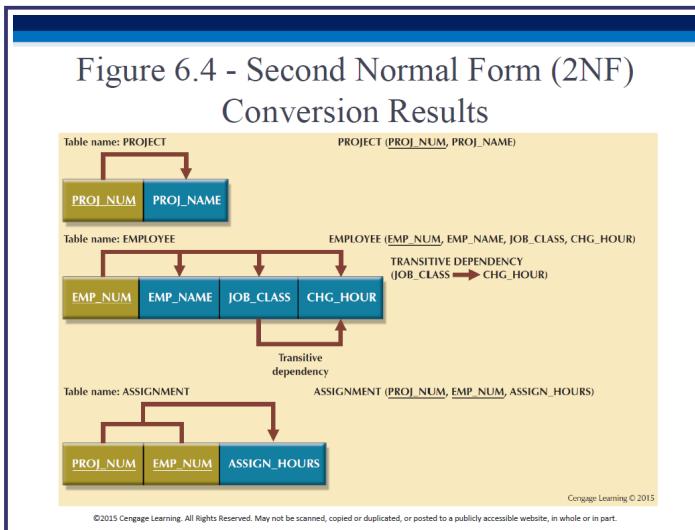
## Conversion to Second Normal Form

- Steps
  - Make new tables to eliminate partial dependencies
  - Reassign corresponding dependent attributes
- Table is in 2NF when it:
  - Is in 1NF
  - Includes no partial dependencies

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 1NF->2NF [cont'd]

We eliminate partial dependencies by creating separate tables of such dependencies, and removing the dependent attributes from the starter table.



# 2NF->3NF: remove transitive dependencies

We promote the non-prime keys that masquerade as PKs, into actual PKs (give them their own tables).

Whether we eliminate partial dependencies (to create 2NF) or transitive ones (to create 3NF), we follow the same process: create a new relation for each 'problem' dependency!

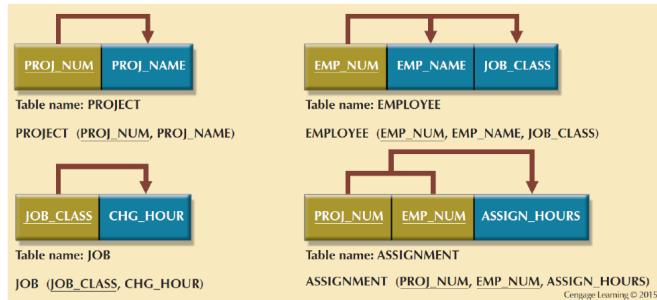
## Conversion to Third Normal Form

- Steps
  - Make new tables to eliminate transitive dependencies
    - **Determinant:** Any attribute whose value determines other values within a row
    - Reassign corresponding dependent attributes
  - Table is in 3NF when it:
    - Is in 2NF
    - Contains no transitive dependencies

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 2NF->3NF [cont'd]

Figure 6.5 - Third Normal Form (3NF)  
Conversion Results



# 'Good' tables

We can create a better DB by doing the following augmentations, to the 3NF model we just created:

- evaluate PKs - create a JOB\_CODE
- evaluate naming conventions - eg. JOB\_CHG\_HOUR
- refine attr atomicity, eg. EMP\_NAME
- identify new attrs, eg. EMP\_HIREDATE
- identify new relationships, PROJECT can have EMP\_NUM as FK [to be able to record a project's (always sole) manager]
- refine PKs for data granularity, eg. ASSIGN\_NUM
- maintain historical accuracy [duplicate data], eg. store JOB\_CHG\_HOUR in ASSIGNMENT
- evaluate derived attrs, eg. ASSIGN\_CHARGE

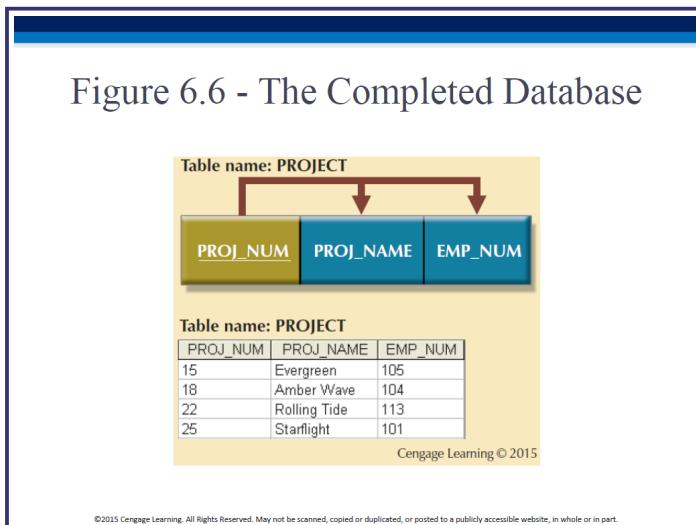
## Requirements for Good Normalized Set of Tables

- Evaluate PK assignments and naming conventions
- Refine attribute atomicity
  - **Atomic attribute:** Cannot be further subdivided
  - **Atomicity:** Characteristic of an atomic attribute
- Identify new attributes and new relationships
- Refine primary keys as required for data granularity
  - **Granularity:** Level of detail represented by the values stored in a table's row
- Maintain historical accuracy and evaluate using derived attributes

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Final result

Here is the result of making the "extra" changes to our 3NF form:



# Final result [cont'd]

Figure 6.6 - The Completed Database

Table name: JOB		
Database name: Ch06_ConstructCo		
JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	36.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.75
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.35

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Final result [cont'd]

Figure 6.6 - The Completed Database

Table name: ASSIGNMENT						
ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-14 15	103	2.6	84.50	219.70	
1002	04-Mar-14 18	110	1.4	18.36	25.70	
1003	05-Mar-14 15	101	3.6	100.00	278.00	
1004	05-Mar-14 22	113	2.5	48.10	120.25	
1005	05-Mar-14 15	103	1.9	84.50	160.55	
1006	05-Mar-14 25	111	4.2	96.75	246.25	
1007	05-Mar-14 22	105	5.2	100.00	254.00	
1008	05-Mar-14 25	101	1.7	105.00	178.90	
1009	05-Mar-14 15	105	2.0	105.00	210.00	
1010	06-Mar-14 15	102	3.8	96.75	307.65	
1011	06-Mar-14 22	104	2.8	96.75	251.55	
1012	06-Mar-14 15	101	2.3	105.00	241.50	
1013	06-Mar-14 25	114	1.8	48.10	86.56	
1014	06-Mar-14 22	111	4.0	26.87	107.48	
1015	06-Mar-14 25	114	3.4	48.10	168.54	
1016	06-Mar-14 18	112	1.2	45.95	55.14	
1017	06-Mar-14 18	118	2.0	10.36	36.72	
1018	06-Mar-14 18	104	2.6	96.75	251.55	
1019	06-Mar-14 15	103	3.0	84.50	253.50	
1020	07-Mar-14 22	105	2.7	105.00	283.50	
1021	08-Mar-14 25	108	4.2	96.75	406.95	
1022	07-Mar-14 25	114	5.8	48.10	278.98	
1023	07-Mar-14 22	106	2.4	35.75	85.00	

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Final result [cont'd]

Figure 6.6 - The Completed Database

Table name: EMPLOYEE						Database name: Ch06_ConstructCo	
EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE		
<b>Table name: EMPLOYEE</b>							
101	News	John	G	08-Nov-00	502		
102	Senior	David	H	12-Jul-99	601		
103	Arbrough	June	E	01-Dec-97	603		
104	Ramoras	Annie	K	15-Nov-88	601		
105	Johnson	Alice	K	01-Feb-94	602		
106	Shirey	William		22-Jun-95	500		
107	Auren	Maria	D	10-Oct-94	500		
108	Washington	Ralph	B	22-Aug-99	601		
109	Smith	Larry	W	18-Jul-99	601		
110	Olenko	Gerald	A	11-Dec-96	605		
111	Wabash	Geoff	B	04-Apr-89	606		
112	Smitsion	Darlene	M	23-Oct-95	607		
113	Joinmod	Delbert	K	15-Nov-94	508		
114	Jones	Annelise		20-Aug-91	608		
115	Brewingt	Travis	B	25-Jan-90	601		
116	Patt	Gerald	L	08-Mar-95	610		
117	Williamson	Angie	H	19-Jun-94	609		
118	Frommer	James	J	04-Jan-06	510		

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Normalization: summary

- \* 1NF: eliminate repeating groups (partial:y, transitive:y)
  - \* 2NF: eliminate redundant data (partial:n, transitive:y)
  - \* 3NF: eliminate fields not dependent on key fields (partial:n, transitive:n)
- 

Here is more, on normalization.

---

# Denormalization

## Denormalization

- Design goals
  - Creation of normalized relations
  - Processing requirements and speed
- Number of database tables expands when tables are decomposed to conform to normalization requirements
- Joining a larger number of tables:
  - Takes additional input/output (I/O) operations and processing logic
  - Reduces system speed

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Denormalization [cont'd]

## Denormalization

- Defects in unnormalized tables
  - Data updates are less efficient because tables are larger
  - Indexing is more cumbersome
  - No simple strategies for creating virtual tables known as views

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.



# SQL

# Where did SQL come from?

As mentioned earlier, SQL is an implementation of Ed Codd's relational set operators:

1. SELECT [formerly known as RESTRICT]
2. PROJECT
3. JOIN
4. PRODUCT
5. UNION
6. INTERSECT
7. DIFFERENCE
8. DIVIDE

Interestingly, Ed's ideas were ignored by IBM, his employer. Only after Oracle debuted (with SQL support right off the bat!) did IBM create DB2, its first relational DB.

## Structured Query Language (SQL)

- Categories of SQL function
  - Data definition language (DDL)
  - Data manipulation language (DML)
- Nonprocedural language with basic command vocabulary set of less than 100 words
- Differences in SQL dialects are minor

[prev](#) [next](#) [page \(right arrow\)](#)

## Table 7.1 - SQL Data Definition Command

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Cengage Learning © 2015

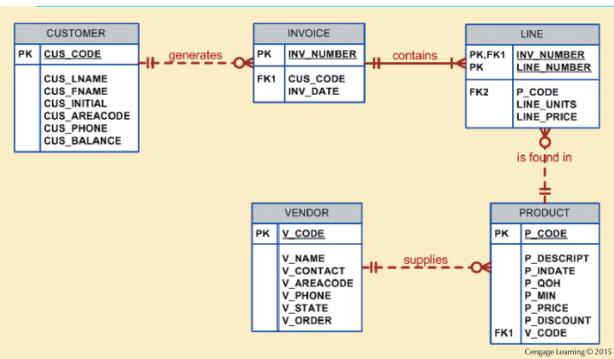
View next page (Right Arrow)

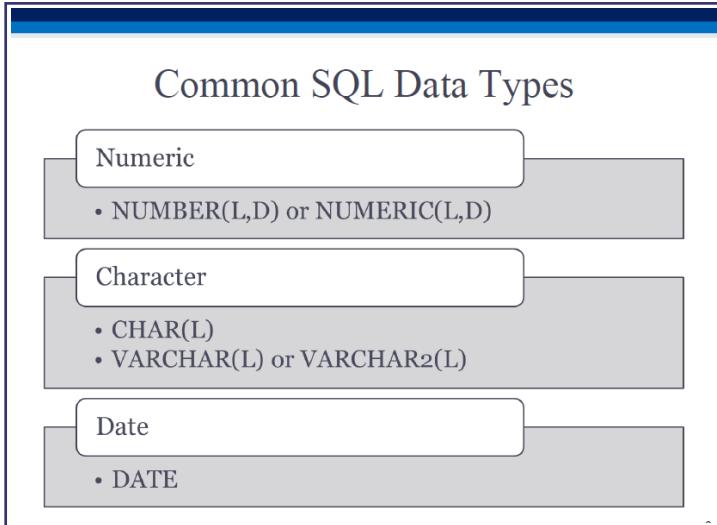
## Table 7.2 - SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
<b>Comparison operators</b>	
=, <, >, <=, >=, <>	Used in conditional expressions
<b>Logical operators</b>	
AND/OR/NOT	Used in conditional expressions
<b>Special operators</b>	
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
<b>Aggregate functions</b>	
Used with SELECT to return mathematical summaries on columns	
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Cengage Learning © 2015

Figure 7.1 - The Database Model





char vs VARCHAR (reserved for future use by SQL - so don't use!) vs VARCHAR2 [versus NVARCHAR2 (for Unicode)]:

[http://www.orafaq.com/faq/what\\_is\\_the\\_difference\\_between\\_varchar\\_varchar2\\_and\\_char\\_data\\_types](http://www.orafaq.com/faq/what_is_the_difference_between_varchar_varchar2_and_char_data_types)

## Creating Table Structures

- Use one line per column (attribute) definition
- Use spaces to line up attribute characteristics and constraints
- Table and attribute names are capitalized
- Features of table creating command sequence
  - NOT NULL specification
  - UNIQUE specification
- Syntax to create table
  - CREATE TABLE tablename();

## Primary Key and Foreign Key

- Primary key attributes contain both a NOT NULL and a UNIQUE specification
- RDBMS will automatically enforce referential integrity for foreign keys
- Command sequence ends with semicolon
- ANSI SQL allows use of following clauses to cover CASCADE, SET NULL, or SET DEFAULT
  - ON DELETE and ON UPDATE

```
CREATE TABLE PRODUCT (
    P_CODE      VARCHAR(10)      NOT NULL      UNIQUE,
    P_DESCRPT   VARCHAR(35)      NOT NULL,
    P_INDATE    DATE            NOT NULL,
    P_QOH       SMALLINT        NOT NULL,
    P_MIN       SMALLINT        NOT NULL,
    P_PRICE     NUMBER(8,2)      NOT NULL,
    P_DISCOUNT  NUMBER(5,2)      NOT NULL,
    V_CODE      INTEGER,
    PRIMARY KEY (P_CODE),
    FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE
    CASCADE);
```

## SQL Constraints

### NOT NULL

- Ensures that column does not accept nulls

### UNIQUE

- Ensures that all values in column are unique

### DEFAULT

- Assigns value to attribute when a new row is added to table

### CHECK

- Validates data when attribute value is entered

Plus: ON UPDATE CASCADE and ON DELETE CASCADE - both affect [change] a secondary table (that has an FK), when a change is made in the primary table (with the corresponding PK). ON UPDATE will update the values in the secondary table when corresp. values in the primary table are changed; ON DELETE will delete rows in the secondary table, when linked rows are deleted in the primary table.

## Data Manipulation Commands

INSERT, SELECT, COMMIT, UPDATE, ROLLBACK, and DELETE.

### INSERT: Command to insert data into table

- Syntax - INSERT INTO tablename VALUES();
- Used to add table rows with NULL and NOT NULL attributes

### COMMIT: Command to save changes

- Syntax - COMMIT [WORK];
- Ensures database update integrity

```
INSERT INTO VENDOR VALUES (21225,'Bryson, Inc.:'Smithson';615'223-3234';TN';Y');
INSERT INTO VENDOR VALUES (21226,'Superloo, Inc.';Flushing';904'215-8995';FL';N');

INSERT INTO PRODUCT VALUES ('11QER/31';Power painter, 15 psi., 3-nozzle';03-Nov-13';8,5,109.99,0.00,25595);
INSERT INTO PRODUCT VALUES ('13-Q2/P2';7.25-in. pwr. saw blade';13-Dec-13';32,15,14.99,0.05, 21344);

INSERT INTO PRODUCT(P_CODE, P_DESCRPT) VALUES ('BRT-345';Titanium drill bit');18-Oct-13';75, 10, 4.50, 0.06, NULL);

INSERT INTO PRODUCT(P_CODE, P_DESCRPT) VALUES ('BRT-345';Titanium drill bit');
```

Vaguely similar to parameter passing/matching during a function call [where arguments need to match in position, type, count].

## Data Manipulation Commands

### SELECT: Command to list the contents

- Syntax - `SELECT columnlist FROM tablename;`
- **Wildcard character(\*)**: Substitute for other characters/command

### UPDATE: Command to modify data

- Syntax - `UPDATE tablename SET columnname = expression [, columnname = expression] [WHERE conditionlist];`

```
SELECT * FROM
PRODUCT;

SELECT P_CODE, P_DESCRIP, P_INDATE, P_QOH, P_MIN, P_PRICE, P_DISCOUNT,
V_CODE
FROM PRODUCT;

UPDATE PRODUCT
SET P_INDATE =
'18-JAN-2014'
WHERE P_CODE = '13-Q2/P2';

UPDATE PRODUCT
SET P_INDATE = '18-JAN-2014', P_PRICE = 17.99, P_MIN
= 10
WHERE P_CODE = '13-Q2/P2';
```

SELECT operates on 1 or more columns, and 0 or more rows from 1 or more tables - like many SQL commands, it is **set-oriented** and **non procedural**.

UPDATE modifies one or more columns of a table (on all rows, or on specific rows based on a condition specified by WHERE). [Here](#) is more.

## Data Manipulation Commands

### WHERE condition

- Specifies the rows to be selected

### ROLLBACK: Command to restore the database

- Syntax - ROLLBACK;
- Undoes the changes since last COMMIT command

### DELETE: Command to delete

- Syntax - DELETE FROM *tablename*
  - [WHERE *conditionlist*];

```
DELETE FROM    PRODUCT  
WHERE          P_CODE = 'BRT-345';
```

Another ex: DELETE FROM PRODUCT WHERE P\_MIN=5; (can be any condition on any attribute).

Q: what would DELETE do, if there isn't a WHERE condition?

## Inserting Table Rows with a SELECT Subquery

- Syntax
  - `INSERT INTO tablename SELECT columnlist FROM tablename`
- Used to add multiple rows using another table as source
- SELECT command - Acts as a subquery and is executed first
  - **Subquery:** Query embedded/nested inside another query

```
INSERT INTO      SELECT      FROM
        columnlist    tablename;
```

## Selecting Rows Using Conditional Restrictions

- Following syntax enables to specify which rows to select
  - `SELECT columnlist`
  - `FROM tablelist`
  - `[WHERE conditionlist];`
- Used to select partial table contents by placing restrictions on the rows
- Optional WHERE clause
  - Adds conditional restrictions to the SELECT statement

```
SELECT    columnlist
FROM      tablelist
[WHERE    conditionlist
];
```

## Comparison Operators

- Add conditional restrictions on selected table contents
- Used on:
  - Character attributes
  - Dates

**Table 7.6 - Comparison Operators**

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Cengage Learning © 2015

## Comparison Operators: Computed Columns and Column Aliases

- SQL accepts any valid expressions/formulas in the computed columns
- **Alias:** Alternate name given to a column or table in any SQL statement to improve the readability
- Computed column, an alias, and date arithmetic can be used in a single query

```
SELECT      P_DESCRPT, P_QOH, P_PRICE, P_QOH *  
            P_PRICE  
FROM        PRODUCT
```

P_DESCRPT	P_QOH	P_PRICE	Expr1
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT      P_DESCRPT, P_QOH, P_PRICE, P_QOH * P_PRICE AS TOTVALUE  
FROM        PRODUCT;
```

P_DESCRPT	P_QOH	P_PRICE	TOTVALUE
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT      P_CODE, P_INDATE, P_INDATE + 90 AS EXPDATE
FROM        PRODUCT;
```

## Arithmetic operators

- **The Rule of Precedence:** Establish the order in which computations are completed
- Perform:
  - Operations within parentheses
  - Power operations
  - Multiplications and divisions
  - Additions and subtractions

**Table 7.7 - The Arithmetic Operators**

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
<sup>^</sup>	Raise to the power of (some applications use $^{**}$ instead of $^$ )

Cengage Learning © 2015

Figure 7.12 - Selected PRODUCT Table  
Attributes: The logical OR

P_DESCRPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-13	14.99	21344
9.00-in. pwr. saw blade	13-Nov-13	17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-13	4.99	21344
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288

Cengage Learning © 2015

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM   PRODUCT
WHERE  V_CODE = 21344 OR V_CODE = 24288;
```

Figure 7.13 - Selected PRODUCT Table  
Attributes: The Logical AND

P_DESCRPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231

Cengage Learning © 2015

```
SELECT      P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       P_PRICE < 50
AND         P_INDATE > '15-Jan-2014';
```

Figure 7.14 - Selected PRODUCT Table  
Attributes: The Logical AND and OR

P_DESCRPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231

Cengage Learning © 2015

```
SELECT      P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       (P_PRICE < 50 AND P_INDATE > '15-Jan-2014')
OR          V_CODE = 24288;
```

## Special Operators

### BETWEEN

- Checks whether attribute value is within a range

### IS NULL

- Checks whether attribute value is null

### LIKE

- Checks whether attribute value matches given string pattern

### IN

- Checks whether attribute value matches any value within a value list

### EXISTS

- Checks if subquery returns any rows

```
SELECT      *
FROM        PRODUCT
WHERE       P_PRICE BETWEEN 50.00 AND 100.00;
```

```
SELECT      P_CODE, P_DESCRIP, V_CODE
FROM        PRODUCT
WHERE       V_CODE IS NULL;
```

- % means any and all *following* or *preceding* characters are eligible. For example:
  - 'J%e' includes Johnson, Jones, Jernigan, July, and J-231Q.
  - 'Jo%' includes Johnson and Jones.
  - '%n' includes Johnson and Jernigan.
- \_ means any *one* character may be substituted for the underscore. For example:
  - '\_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.
  - '\_23\_56-678\_' includes 123-156-6781, 123-256-6782, and 823-956-6788.
  - '\_o\_es' includes Jones, Cones, Cokes, totes, and roles.

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE  
FROM VENDOR  
WHERE V_CONTACT LIKE 'Smith%';
```

Another example for pattern-matching: load [this page](#), and enter and execute:

```
SELECT * FROM Customers WHERE CustomerName LIKE 'C%';
```

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE = 21344
OR          V_CODE = 24288;

SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);

SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

## 'EXISTS'

The EXISTS clause is used with a query, and returns TRUE if the subquery results in any output (non-zero # of rows being returned), or FALSE if the subquery results in no data. The rest of the query (the 'main' query) will (or will not) run, based on EXIST's output - if EXISTS returns false, the main query will get skipped.

You can loosely think of EXISTS as "ONLY WHEN". We use it to 'defensively' update (insert, modify, delete) parts of a table (after we determine it is updatable).

You can also think of 'WHERE EXISTS' as "such that there exists". Eg. in the first example below (SELECT \* FROM VENDOR..), it reads as "Find all vendors such that there exists records for them in the PRODUCT TABLE (via V\_CODE) where P\_QOH<=P\_MIN" [in other words, we are looking for vendors we need to re-order from].

In the second example (INSERT INTO CONTACTS..), read it as "get the supplier ID and name for all suppliers such that there exists order IDs for them, then insert them into the contacts table".

Exercise: what does #3 below do?

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

```
INSERT INTO contacts
(contact_id, contact_name)
SELECT supplier_id, supplier_name
FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

```
UPDATE suppliers
SET supplier_name = (SELECT customers.name
                      FROM customers
                      WHERE customers.customer_id = suppliers.supplier_id)
WHERE EXISTS (SELECT customers.name
              FROM customers
              WHERE customers.customer_id = suppliers.supplier_id);
```

```
DELETE FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

```
SELECT *
FROM suppliers
WHERE EXISTS (SELECT *
               FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

This SQL EXISTS condition example will return all records from the suppliers table where there is at least one record in the orders table with the same supplier\_id.

```
SELECT *
FROM suppliers
WHERE NOT EXISTS (SELECT *
                     FROM orders
                    WHERE suppliers.supplier_id = orders.supplier_id);
```

This SQL EXISTS example will return all records from the suppliers table where there are no records in the orders table for the given supplier\_id.

## Advanced Data Definition Commands

- **ALTER TABLE** command: To make changes in the table structure
- Keywords use with the command
  - ADD - Adds a column
  - MODIFY - Changes column characteristics
  - DROP - Deletes a column
- Used to:
  - Add table constraints
  - Remove table constraints

## Changing Column's Data Type

- ALTER can be used to change data type
- Some RDBMSs do not permit changes to data types unless column is empty
- Syntax –
  - `ALTER TABLE tablename MODIFY (columnname(datatype));`

```
ALTER TABLE PRODUCT  MODIFY (V_CODE CHAR(5));
```

## Changing Column's Data Characteristics

- Use ALTER to change data characteristics
- Changes in column's characteristics are permitted if changes do not alter the existing data type
- Syntax
  - `ALTER TABLE tablename MODIFY (columnname(characterstic));`

```
ALTER TABLE PRODUCT MODIFY (P_PRICE DECIMAL(9,2));
```

## Adding Column, Dropping Column

- Adding a column
  - Use ALTER and ADD
  - Do not include the NOT NULL clause for new column
- Dropping a column
  - Use ALTER and DROP
  - Some RDBMSs impose restrictions on the deletion of an attribute

```
ALTER TABLE PRODUCT ADD (P_SALECODE CHAR(1));
```

```
ALTER TABLE VENDOR DROP COLUMN V_ORDER;
```

## Advanced Data Updates

- UPDATE command updates only data in existing rows
- If a relationship is established between entries and existing columns, the relationship can assign values to appropriate slots
- Arithmetic operators are useful in data updates
- In Oracle, ROLLBACK command undoes changes made by last two UPDATE statements

```
UPDATE      PRODUCT
SET         P_SALECODE = '2'
WHERE       P_CODE = '1546-QQ2';

UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_CODE IN ('2232/QWE', '2232/QTY');

UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_INDATE >= '16-Jan-2014' AND P_INDATE <='10-Feb-2014';

UPDATE      PRODUCT
SET         P_PRICE = P_PRICE * 1.10
WHERE       P_PRICE < 50.00;
```

## Table update in 'bulk'

Sometimes we can update a table, by filling it with output from a query. The table to be filled in has to exist first (so we need to create it if necessary), and have type-compatible columns that can receive values from our data-fetching query.

The table creation and updating can happen in separate steps, or even be combined for compactness of expression.

### Copying Parts of Tables

- SQL permits copying contents of selected table columns
  - Data need not be reentered manually into newly created table(s)
- Table structure is created
- Rows are added to new table using rows from another table

```
CREATE TABLE PART(
    PART_CODE          CHAR(8),
    PART_DESCRPT      CHAR(35),
    PART_PRICE        DECIMAL(8,2),
    V_CODE            INTEGER,
    PRIMARY KEY (PART_CODE));

INSERT INTO PART      (PART_CODE, PART_DESCRPT, PART_PRICE, V_CODE)
SELECT              P_CODE, P_DESCRPT, P_PRICE, V_CODE FROM PRODUCT;

CREATE TABLE PART AS
SELECT    P_CODE AS PART_CODE, P_DESCRPT AS
          PART_DESCRPT, P_PRICE AS PART_PRICE,
          V_CODE
FROM     PRODUCT;
```

## Adding Primary and Foreign Key Designations

- **ALTER TABLE** command
  - Followed by a keyword that produces the specific change one wants to make
  - Options include ADD, MODIFY, and DROP
- Syntax to add or modify columns
  - **ALTER TABLE *tablename***
    - {ADD | MODIFY} ( *columnname datatype* [ {ADD | MODIFY} *columnname datatype*] );
  - **ALTER TABLE *tablename***
    - ADD *constraint* [ ADD *constraint* ] ;

```
ALTER TABLE PART
ADD PRIMARY KEY (PART_CODE);
```

```
ALTER TABLE PART
ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Oftentimes the need to do this occurs when a table (eg. PART) is created via bulk-update, from another table (eg. PRODUCT).

## Deleting a Table from the Database

- **DROP TABLE:** Deletes table from database
- Syntax - `DROP TABLE tablename;`
- Can drop a table only if it is not the one side of any relationship
- RDBMS generates a foreign key integrity violation error message if the table is dropped

# Ordering, unique entries, aggregate ops..

## Additional SELECT Query Keywords

- Logical operators work well in the query environment
- SQL provides useful functions that:
  - Counts
  - Find minimum and maximum values
  - Calculate averages
- SQL allows user to limit queries to entries:
  - Having no duplicates
  - Whose duplicates may be grouped

# ORDER BY

## Ordering a Listing

- **ORDER BY** clause is useful when listing order is important
- Syntax - `SELECT columnlist  
FROM tablelist  
[WHERE conditionlist]  
[ORDER BY columnlist [ASC | DESC]];`
- **Cascading order sequence:** Multilevel ordered sequence
  - Created by listing several attributes after the ORDER BY clause

Below is an example of ORDER BY, where by default, values are listed (sorted) in ascending order. To list the values in descending order, we'd do this: ORDER BY P\_PRICE DESC;

```
SELECT P.CODE, P.DESCRIPT, P.INDATE, P.PRICE
FROM PRODUCT
ORDER BY P.PRICE;
```

P_CODE	P_DESCRIP	P_INDATE	P_PRICE
54776-2T	Rd-lst fl, 18-in.	15-Dec-13	4.99
PVC230RT	PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.67
SM-16277	1.25-in. metal screw, 25	01-Mar-14	6.99
SM-23116	2.5-in. wd. screw, 50	24-Feb-14	8.45
23109-HB	Claw hammer	20-Jan-14	9.95
23114-AA	Sledge hammer, 12 lb.	02-Jan-14	14.40
13-020P2	7.25-in. pwr. saw blade	13-Dec-13	14.99
14-01A3	9.00-in. pwr. saw blade	13-Nov-13	17.49
22306-CPO	BSD cordless drill, 1/2-in.	20-Jan-14	30.95
1566-G00	Hrd. cloth, 14-in., 2x50	15-Jan-14	39.95
1556-GW0	Hrd. cloth, 12-in., 3x50	15-Jan-14	49.99
2232QME	BSD jigsaw, 5-in. blade	24-Dec-13	99.67
2232GTY	BSD jigsaw, 12-in. blade	30-Dec-13	109.92
116ER21	Power painter, 15 psi., 3-nozzle	03-Nov-13	109.99
WR3TT3	Steel matting, 4x9x1.6", 5' mesh	17-Jan-14	119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-14	255.99

Cengage Learning © 2015

The sequence (listing) obtained by specifying several comma-separated attributes for ORDER BY is called a cascading order sequence.

```
SELECT EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM EMPLOYEE
ORDER BY EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diantre	Jorge	D	615	890-4567
Genkazi	Leighla	W	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220

Cengage Learning © 2015

# DISTINCT

The 'DISTINCT' keyword is used to count unique/distinct occurrences of an attribute:

## Listing Unique Values

- **DISTINCT** clause: Produces list of values that are unique
- Syntax - `SELECT DISTINCT columnlist  
FROM tablelist;`
- Access places nulls at the top of the list
  - Oracle places it at the bottom
  - Placement of nulls does not affect list contents

## Example of 'DISTINCT' usage

```
SELECT DISTINCT V_CODE  
FROM PRODUCT;
```

V_CODE
21225
21231
21344
23119
24288
25595

Cengage Learning © 2015

# AGGREGATE functions

COUNT, MIN, MAX, SUM, AVG are all functions that operate on a numerical attr/column, and produce a single scalar result (not a table).

Table 7.8 - Some Basic SQL Aggerate Functions

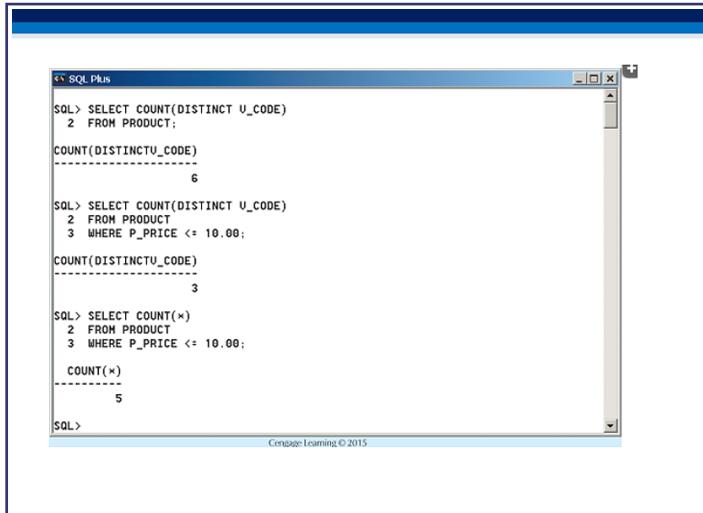
FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

Cengage Learning © 2015

# Aggregate function examples

How many different vendors supply our products? How many supply cheap (PRICE < 10) products?

Note the third query below: COUNT(\*) counts the # of rows returned by a query (rows where the product costs < 10 units). In contrast, count() counts the # of non-null values of the column.



The screenshot shows a window titled "SQL\*Plus" with three SQL queries entered and their results displayed:

```
SQL> SELECT COUNT(DISTINCT V_CODE)
  2  FROM PRODUCT;
COUNT(DISTINCTV_CODE)
-----
6

SQL> SELECT COUNT(DISTINCT V_CODE)
  2  FROM PRODUCT
  3  WHERE P_PRICE <= 10.00;
COUNT(DISTINCTV_CODE)
-----
3

SQL> SELECT COUNT(*)
  2  FROM PRODUCT
  3  WHERE P_PRICE <= 10.00;
COUNT(*)
-----
5
```

The window has a standard title bar and a scroll bar on the right side. The footer of the window displays "Cengage Learning © 2015".

# MAX, MIN

What is the value for the most expensive item in the product table? Least expensive?

What is the most expensive item (details)? We can't do: WHERE P\_PRICE = MAX(P\_PRICE). This is because MAX() can only be used in a SELECT statement.

The screenshot shows the Oracle SQL Plus interface. The command window contains the following SQL statements:

```
SQL> SELECT MAX(P_PRICE)
  2  FROM PRODUCT;
MAX(P_PRICE)
-----
256.99

SQL> SELECT MIN(P_PRICE)
  2  FROM PRODUCT;
MIN(P_PRICE)
-----
4.99

SQL> SELECT P_CODE, P_DESCRIP, P_PRICE
  2  FROM PRODUCT
  3  WHERE P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);
P_CODE      P_DESCRIP          P_PRICE
-----  -----
89-WRE-Q    Hicut chain saw, 16 in.   256.99

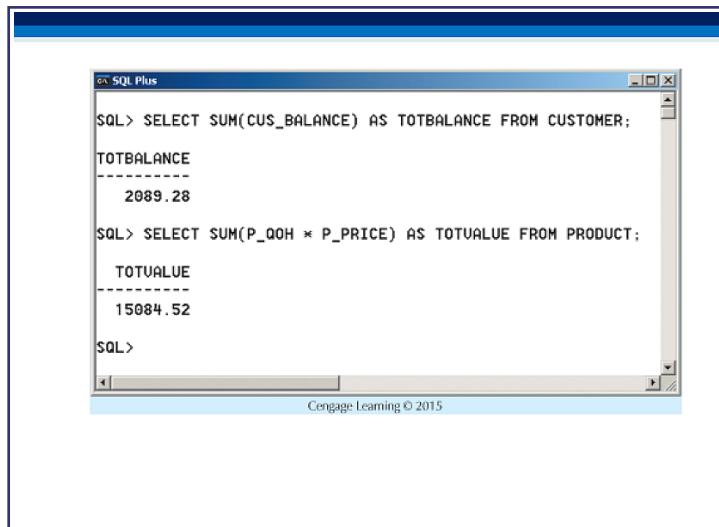
SQL>
```

Below the command window, a results window displays the output of the last query:

P_CODE	P_DESCRIP	P_PRICE
89-WRE-Q	Hicut chain saw, 16 in.	256.99

At the bottom of the interface, there is a footer bar with the text "Copyright © 2015" and a status bar at the very bottom.

# SUM



The screenshot shows an Oracle SQL Plus window with a blue title bar and a white body. It contains the following SQL statements and their results:

```
SQL> SELECT SUM(CUS_BALANCE) AS TOTBALANCE FROM CUSTOMER;
TOTBALANCE
-----
2089.28

SQL> SELECT SUM(P_QOH * P_PRICE) AS TOTVALUE FROM PRODUCT;
TOTVALUE
-----
15084.52

SQL>
```

In the bottom right corner of the window, there is a small watermark that reads "Cengage Learning © 2015".

# AVG

The screenshot shows a window titled "SQL Plus" with the following content:

```
SQL> SELECT AVG(P_PRICE) FROM PRODUCT;
AUG(P_PRICE)
-----
56.42125

SQL> SELECT P_CODE, P_DESCRIP, P_QOH, P_PRICE, V_CODE
  2  FROM PRODUCT
  3 WHERE P_PRICE > (SELECT AVG(P_PRICE) FROM PRODUCT)
  4 ORDER BY P_PRICE DESC;

P_CODE      P_DESCRIP          P_QOH    P_PRICE      V_CODE
-----  -----
69-WRE-Q    Hicut chain saw, 16 in.        11    256.99    24288
WR3/TT3     Steel matting, 4'x8'x1/6", .5" mesh   18    119.95    25595
110ER/31    Power painter, 15 psi., 3-nozzle       8    109.99    25595
2232/QTV    B\&D jigsaw, 12-in. blade        8    109.92    24288
2232/QWE    B\&D jigsaw, 8-in. blade       6    99.87    24288

SQL>
```

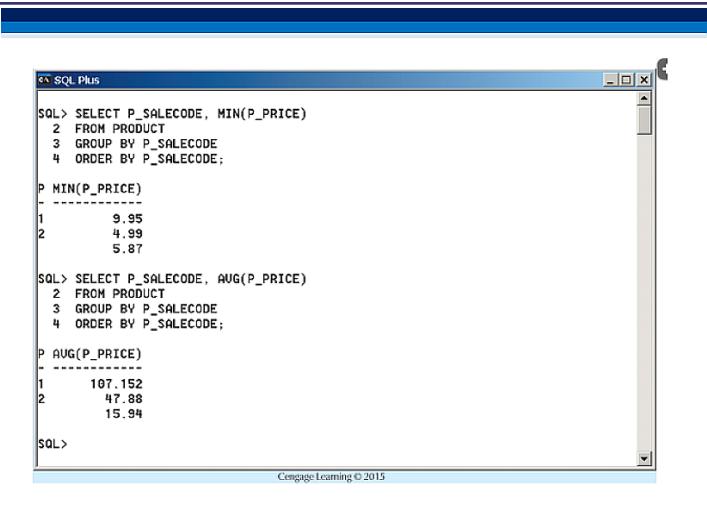
Cengage Learning © 2015

# GROUP BY (itemizing)

## Grouping Data

- Frequency distributions created by **GROUP BY** clause within SELECT statement
- GROUP BY can **only** be used in concert with an aggregate function
- Syntax - `SELECT columnlist  
FROM tablelist  
[WHERE conditionlist]  
[GROUP BY columnlist]  
[HAVING conditionlist]  
[ORDER BY columnlist [ASC | DESC]]; ..`

## 'GROUP BY' example



The screenshot shows a window titled "SQL\*Plus" with two SQL queries displayed. The first query is:

```
SQL> SELECT P_SALECODE, MIN(P_PRICE)
  2  FROM PRODUCT
  3  GROUP BY P_SALECODE;
  4  ORDER BY P_SALECODE;
```

The output is:

P_SALECODE	MIN(P_PRICE)
1	9.95
2	4.99
	5.87

The second query is:

```
SQL> SELECT P_SALECODE, AVG(P_PRICE)
  2  FROM PRODUCT
  3  GROUP BY P_SALECODE;
  4  ORDER BY P_SALECODE;
```

The output is:

P_SALECODE	AVG(P_PRICE)
1	107.152
2	47.88
	15.94

SQL>

You can think of 'GROUP BY' to mean "CATEGORIZE BY" or "ITEMIZE BY": rather than just a single MIN, MAX, SUM, COUNT or AVG, we're asking for a value PER OCCURRENCE of a GROUP BY value, eg. minimum price PER VENDOR, max GPA PER DEPARTMENT, average earnings PER MAJOR, etc. Specifically, when used with SUMO, GROUP BY is used to request subtotals.

The screenshot shows an SQL Plus window with the following content:

```
SQL> SELECT U_CODE, P_CODE, P_DESCRIP, P_PRICE
  2  FROM PRODUCT
  3 GROUP BY U_CODE;
SELECT U_CODE, P_CODE, P_DESCRIP, P_PRICE
      *
ERROR at line 1:
ORA-00979: not a GROUP BY expression

SQL> SELECT U_CODE, COUNT(DISTINCT P_CODE)
  2  FROM PRODUCT
  3 GROUP BY U_CODE;
U_CODE COUNT(DISTINCTP_CODE)
-----
21225          2
21231          1
21344          3
23119          2
24288          3
25595          3
              2

7 rows selected.

SQL>
```

A note at the bottom of the window reads: "Contains Learning料2015".

'GROUP BY' used without an aggregate function is meaningless, since there is no aggregate value (MIN, COUNT etc.) to itemize.

# Store receipt

Here is a grocery store receipt where subtotals are displayed itemized, aggregated by product types DELI, GROCERY, MIXED NUTS and PRODUCE (presumably using SUMQ, together with GROUP BY):



While you're at it, see what other DATA you can spot in the receipt! Even a single trip to the store can generate a LOT of data.

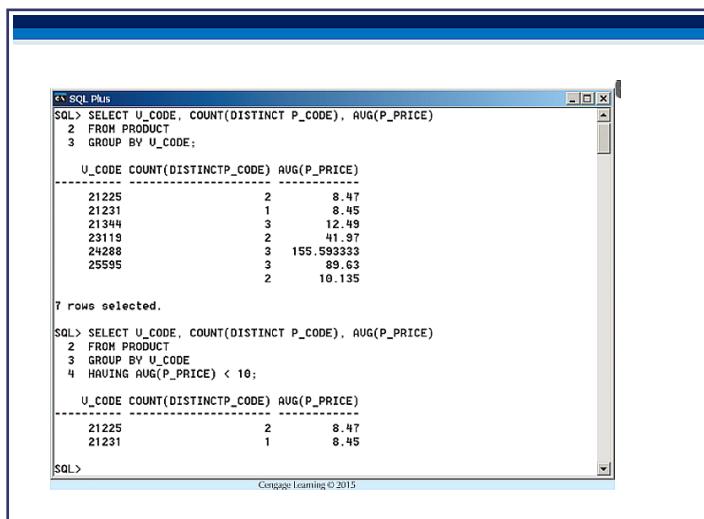
# HAVING

The HAVING clause is used to filter rows in a GROUP BY specification (only those rows HAVING met the specified condition will appear in the result).

Note that HAVING can only occur in a query that has a GROUP BY, which in turn can only occur when there is an aggregate function (MIN, MAX, SUM, COUNT or AVG).

## HAVING Clause

- Extension of GROUP BY feature
- Applied to output of GROUP BY operation
- Used in conjunction with GROUP BY clause in second SQL command set
- Similar to WHERE clause in SELECT statement



SQL> SELECT U\_CODE, COUNT(DISTINCT P\_CODE), AVG(P\_PRICE)  
2 FROM PRODUCT  
3 GROUP BY U\_CODE;

U_CODE	COUNT(DISTINCTP_CODE)	AVG(P_PRICE)
21225	2	8.47
21231	1	8.45
21344	3	12.49
23119	2	41.97
24288	3	155.593333
25595	3	89.63
	2	10.135

7 rows selected.

SQL> SELECT U\_CODE, COUNT(DISTINCT P\_CODE), AVG(P\_PRICE)  
2 FROM PRODUCT  
3 GROUP BY U\_CODE  
4 HAVING AVG(P\_PRICE) < 10;

U_CODE	COUNT(DISTINCTP_CODE)	AVG(P_PRICE)
21225	2	8.47
21231	1	8.45

SQL>

# Table joins

## Joining Database Tables

- Performed when data are retrieved from more than one table at a time
  - Equality comparison between foreign key and primary key of related tables
- Tables are joined by listing tables in FROM clause of SELECT statement
  - DBMS creates Cartesian product of every table in the FROM clause

## Joining Tables With an Alias

- Alias identifies the source table from which data are taken
- Any legal table name can be used as alias
- Add alias after table name in FROM clause
  - FROM tablename alias, eg. 'FROM PRODUCT P'

```
SELECT P_DESCRIP, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

P_DESCRIP	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
Claw hammer	9.95	Bryson, Inc.	Smithson	615	223-3234
1.25-in. metal screw, 25	6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	8.45	D&E Supply	Singh	615	228-3245
7.25-in. pwr. saw blade	14.99	Gomez Bros.	Ortega	615	889-2546
9.00-in. pwr. saw blade	17.49	Gomez Bros.	Ortega	615	889-2546
Re-tail file, 18-in. fine	4.99	Gomez Bros.	Ortega	615	889-2546
Hrd. cloth, 1/4-in., 2x50	39.95	RandsSets Ltd.	Anderson	901	678-3998
Hrd. cloth, 1/2-in., 3x50	43.99	RandsSets Ltd.	Anderson	901	678-3998
B&D jigsaw, 12-in. blade	109.92	ORDVA, Inc.	Hakford	615	898-1234
B&D jigsaw, 8-in. blade	99.87	ORDVA, Inc.	Hakford	615	898-1234
Hicut chain saw, 16 in.	255.99	ORDVA, Inc.	Hakford	615	898-1234
Power painter, 15 psi., 3-nozzle	109.99	Rubicon Systems	Orton	904	456-0092
B&D cordless drill, 1/2-in.	38.95	Rubicon Systems	Orton	904	456-0092
Steel matting, 4'x8'x1/8", 5 <sup>th</sup> mesh	119.95	Rubicon Systems	Orton	904	456-0092

Cengage Learning © 2015

```
ORDER BY PRODUCT.P_PRICE;
```

```
SELECT      CUS_LNAME, INVOICE.INV_NUMBER, INV_DATE, P_DESCRPT
FROM        CUSTOMER, INVOICE, LINE, PRODUCT
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND         INV_DATE > '2021-09-01'
AND         LINE.P_CODE = PRODUCT.P_CODE
AND         CUSTOMER.CUS_CODE = 10014
ORDER BY    INV_NUMBER;
```

Joining n tables will need (n-1) join conditions.

## 'Recursive join' example

Need different aliases for the table being queried so that we can use such aliases as namespaces for attributes.

### Recursive Joins

- **Recursive query:** Table is joined to itself using alias
- Use aliases to differentiate the table from itself

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_AREA_CODE	EMP_PHONE	EMP_MGR
100 Mr.	Kolnycz	George	D		15-Mar-42	15-Mar-85 615	224-5459		
101 Ms.	Levitt	Rhonda	G		19-Mar-65	25-Apr-88 615	224-4472	100	
102 Mr.	Vandem	Rhett			14-Nov-59	20-Dec-90 601	675-2993	100	
103 Ms.	Jones	Anne	M		16-Oct-74	20-Aug-94 615	698-3458	100	
104 Mr.	Lange	John	P		06-Mar-71	20-Oct-94 601	504-4430	105	
105 Mr.	Watson	Robert	D		14-Mar-75	08-Nov-98 615	698-3220		
106 Mrs.	Smith	Jeanine	K		12-Feb-69	05-Jan-99 615	324-7893	105	
107 Mr.	Dante	Jorge	D		21-Aug-74	03-Mar-94 615	698-4587	105	
108 Mr.	Wesenberg	Paul	R		14-Feb-66	19-Nov-92 615	897-4358		
109 Mr.	Smith	George	K		18-Jun-61	14-Apr-99 601	504-3339	108	
110 Mrs.	Orenzzi	Leigh-ja	W		19-May-70	01-Dec-90 601	698-0003	108	
111 Mr.	Washington	Rupert	E		03-Jan-66	21-Jun-93 615	698-4925	105	
112 Mr.	Johnson	Edward	E		14-May-61	01-Dec-93 615	898-4287	109	
113 Ms.	Smythe	Melanie	P		15-Sep-70	11-May-99 615	324-9006	105	
114 Ms.	Brendon	Merle	G		02-Nov-55	15-Nov-79 601	882-0845	108	
115 Mrs.	Saranda	Hermine	R		25-Jul-72	23-Apr-93 615	324-5505	105	
116 Mr.	Smith	George	A		08-Nov-65	10-Dec-98 615	698-2984	108	

Copyright © Cengage Learning 2015

```
SELECT      E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME
FROM        EMP E, EMP M
WHERE       E.EMP_MGR=M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

EMP_NUM	E.EMP_LNAME	EMP_MGR	M.EMP_LNAME
112	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diane	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wiesenbach
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach

Cengage Learning © 2015

Sooo... are joins 'evil'? Not really :)

# SQL is declarative, not imperative

Now that you're familiar with SQL syntax, you will appreciate knowing how it compares with a regular programming language such as C++. Shown below is such a comparison, using C# which lets us write code in an imperative (command-oriented) manner as well a declarative (result-oriented) one [this is from a book on functional programming]:

## **Listing 1.3 Imperative data processing (C#)**

```
List<string> res = new List<string>();
foreach(Product p in Products) {
    if (p.UnitPrice > 75.0M) {
        res.Add(String.Format("{0} - ${1}",
            p.ProductName, p.UnitPrice));
    }
}
return res;
#1 Create resulting list
#2 Iterate over products
#3 Add information to list of results
```

You'll probably need to read the code carefully to understand what it does, but that's not the only aspect we want to improve. The code is written as a sequence of some basic imperative commands. For example, the first statement creates new list (#1), the second iterates over all products (#2) and a later one adds element to the list (#3). However, we'd like to be able to describe the problem at a higher level. In more abstract terms, the code just filters a collection and returns some information about every returned product.

In C# 3.0, we can write the same code using query expression syntax. This version is closer to our real goal—it uses the same idea of filtering and transforming the data. You can see the code in listing 1.4.

## **Listing 1.4 Declarative data processing (C#)**

```
var res = from p in Products
          where p.UnitPrice > 75.0M
          select String.Format("{0} - ${1}",
            p.ProductName, p.UnitPrice);
return res;
#1 Filter products using predicate
#2 Return information about product
```

The expression that calculates the result (`res`) is composed from basic operators such as `where` or `select`. These operators take other expressions as an argument, because they need to know exactly what we want to filter or select as a result.

©Manning Publications Co.

# Ways to 'do' SQL..

SQL is a data creation+manipulation language, so it's best learned HANDS ON (not just by looking at slides and reading about the syntax) - you need access to a relational database where you can **create tables, enter data in them and do queries on the data** (tables ← data ← queries).

There are three ways to get your hands on a DB: use a browser page [a server runs the DB software, you simply access it from a page]; install a DB locally on your laptop/tablet/phone; use a 'cloud-based' DB [this is similar to, but more powerful than, accessing a DB via a web page].

## 1. Browser-based SQL IDEs

An easy way to practice running SQL queries is to use browser-based interfaces (nothing to download, no login needed) to create/query databases. Here are some sites that provide this form of access:

- \* SQL Tutorial: <http://www.w3schools.com/sql/default.asp>
- \* ideone: <http://www.ideone.com>
- \* sqlfiddle: <http://sqlfiddle.com/>
- \* Code School's Try SQL: <http://campus.codeschool.com/courses/try-sql/contents>
- \* SQLZOO: <http://sqlzoo.net/> - has extensive tutorials
- \* another tutorial: <http://www.sqltutorial.org/>
- \* w3resource: <http://www.w3resource.com/sql-exercises/> - more tutorials
- \* Khan Academy: <https://www.khanacademy.org/computer-programming/new/sql>
- \* 'Online SQL interpreter': <https://sql-js.github.io/sql.js/examples/GUI/> - JavaScript-based!

Below are examples for three of the above links.

Bring up the w3schools **Tryit Editor**, enter and run the following pair of sql command sets one at a time (these commands run on a set of pre-installed (by w3schools.com) **collection of tables** called the **Northwind database**):

```
SELECT City FROM Customers
WHERE Country="Germany";

SELECT City, CustomerID FROM Customers
WHERE Country="Germany" AND CustomerID>60;
```

In **ideone**, select SQLite (for the choice of language), then enter and run this:

```
-- warmup
create table tble(str varchar(20));
insert into tble values ('Hello world!'),('SQL is fun!!');
select * from tble;
```

```
-- recipes
CREATE TABLE recipes (
    recipe_id INT NOT NULL,
    recipe_name VARCHAR(30) NOT NULL,
    PRIMARY KEY (recipe_id),
    UNIQUE (recipe_name)
);
```

```
INSERT INTO recipes
(recipe_id, recipe_name)
VALUES
(1, "Tacos"),
(2, "Tomato Soup"),
(3, "Grilled Cheese");
```

```
-- quick check
SELECT recipe_name
FROM recipes;
```

```
-- ingredients
CREATE TABLE ingredients (
    ingredient_id INT NOT NULL,
    ingredient_name VARCHAR(30) NOT NULL,
    ingredient_price INT NOT NULL,
    PRIMARY KEY (ingredient_id),
    UNIQUE (ingredient_name)
);
```

```
INSERT INTO ingredients
(ingredient_id, ingredient_name, ingredient_price)
VALUES
(1, "Beef", 5),
(2, "Lettuce", 1),
(3, "Tomatoes", 2),
(4, "Taco Shell", 2),
(5, "Cheese", 3),
(6, "Milk", 1),
(7, "Bread", 2);
```

```
-- recipe_ingredients
CREATE TABLE recipe_ingredients (
recipe_id int NOT NULL,
ingredient_id INT NOT NULL,
amount INT NOT NULL,
PRIMARY KEY (recipe_id,ingredient_id)
);

INSERT INTO recipe_ingredients
(recipe_id, ingredient_id, amount)
VALUES
(1,1,1),
(1,2,2),
(1,3,2),
(1,4,3),
(1,5,1),
(2,3,2),
(2,6,1),
(3,5,1),
(3,7,2);

SELECT *
FROM recipes
ORDER BY recipe_id;
```

Here is the recipes db table and its queries again, this time in sqlfiddle. After the page loads (with our SQL commands above, filled in!), click 'Build Schema' on the left, then click 'Run SQL' on the right [and wait for a few seconds for the output].

## 2. Locally installed DBs

You can install Oracle on your machine, or install the light-weight but powerful [DB Browser for SQLite, aka SQLite Browser](#). On Windows, this offers a 'notebook' environment: <https://sqlnotebook.com/> [a mix of SQL and non-SQL commands]. It is also OK to use Postgres, or MySQL; or you could use [some other DB...](#)

Here is a quickstart guide for SQLite Browser - we create a table called CosineTable, and fill it with rows containing [angle, cos(angle)] pairs. And here is another clip that shows how to create a simple sqlite DB and query it.

The second clip above, also shows that the .db file that is created, contains data in a SQLite-native binary format [which is [public and well-documented](#)]. FYI, there are sqlite API calls in multiple languages that let you access the (binary) data using functions/methods that accept SQL commands as regular strings. This is mixed-language

programming that lets us accomplish a lot, because we leverage the power of Java, Python etc., as well as that of SQL. Eg. here's how it works in Python:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
    print row # or we could email this, put it up on an LED
display, buy/sell the cheap/pricey stocks..
```

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)  
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)  
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)  
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)

You could also write a binary reader+parser in C++, Java etc. and extract the sqlite-native data that way - in other words, you'd never lose your data if you store it in a sqlite .db file!

As you can see, sqlite is **very** popular :)

### 3. Cloud-based DBs

You can also work with a relational DB via the cloud, eg. using Amazon's **RDS** offering. RDS offers a way to create databases such as Amazon's own Aurora, Oracle, SQL Server, MariaDB, MySQL, Postgres.

1/50    11:17:06 \*\*\*



# More SQL

# Ch.8

The book cover for "Database Systems: Design, Implementation, and Management" (11th edition) by Coronel and Morris. The title is at the top, followed by a circular graphic of binary digits. Below is a world map made of binary code. The authors' names are at the bottom.

Chapter 8

Advanced SQL

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Objectives

## Learning Objectives

- In this chapter, the student will learn:
  - How to use the advanced SQL JOIN operator syntax
  - About the different types of subqueries and correlated queries
  - How to use SQL functions to manipulate dates, strings, and other data
  - About the relational set operators UNION, UNION ALL, INTERSECT, and MINUS

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2

# Objectives

## Learning Objectives

- In this chapter, the student will learn:
  - How to create and use views and updatable views
  - How to create and use triggers and stored procedures
  - How to create embedded SQL

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

3

# The 'JOIN' operation

Recall that we looked at examples of joining - entries from **two tables**, and entries from a **single table**. These joins were based on 'join conditions'.

It is also possible to join tables using the 'JOIN' keyword..

# JOIN conditions

Note that JOINs can be based on != (aka <>), >, <, >= and <= as well, in addition to equality. Eg. to list all students who will be getting a 'A' (uses two inequality comparisons indirectly):

```
//  
http://www.comp.nus.edu.sg/~ooibc/courses/sql/dml\_query\_join.htm  
SELECT a.name, a.score  
FROM student_scores a, grade_class b  
WHERE b.grade = 'A' AND a.score BETWEEN  
b.low_end AND b.high_end;
```

## SQL Join Operators

- Relational join operation merges rows from two tables and returns rows with one of the following
  - Natural join - Have common values in common columns
  - Equality or inequality - Meet a given join condition
  - **Outer join**: Have common values in common columns or have no matching values
  - **Inner join**: Only rows that meet a given criterion are selected

# Ways to specify JOIN conditions

Table 8.1 - SQL Join Expression Styles			
JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
CROSS	CROSS JOIN	SELECT * FROM T1, T2	Returns the Cartesian product of T1 and T2 (old style)
		SELECT * FROM T1 CROSS JOIN T2	Returns the Cartesian product of T1 and T2
INNER	Old-style JOIN	SELECT * FROM T1, T2 WHERE T1.C1=T2.C1	Returns only the rows that meet the join condition in the WHERE clause (old style); only rows with matching values are selected
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns; the matching columns must have the same names and similar data types
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1=T2.C1	Returns only the rows that meet the join condition indicated in the ON clause

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Outer vs inner vs full ('both') JOINS

Table 8.1 - SQL Join Expression Styles			
JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
OUTER	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Full (left+right outer) JOIN example

For example, the following query lists the product code, vendor code, and vendor name for all products and includes all product rows (products without matching vendors) as well as all vendor rows (vendors without matching products):

```
SELECT P.CODE, V.CODE, V.NAME
FROM VENDOR FULL JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

The SQL code and its results are shown in Figure 8.12.

**FIGURE 8.12** FULL JOIN results

P.CODE	V.CODE	V.NAME
10000001	22500	Rudolph Systems
12345672	21345	Gomez Bros.
14-34153	21346	Damek Bros.
1545-0000	21347	Elmer J. Fudd Ltd.
1558-0001	20110	Rabbitz Ltd.
2233	20200	Orville, Inc.
2232/00E	20200	Orville, Inc.
2238/QPQ	25595	Rudolph Systems
2345-0000	21225	Gomez Bros.
54778-21	21346	Gomez Bros.
89-00000	21225	Gomez Bros., Inc.
SH-10277	21225	Brosco, Inc.
SW-23116	21211	Dale Supply
WB/113	22547	Dome Supply
	22548	Dome Supply, Inc.
	24000	Brikkman Bros.
	25541	Dwail Supplies
	25541	Dwail, Inc.
29114-AH		
P022801		
		21 rows selected.
		SQL>

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 'SELECT' subqueries

SELECT SUBQUERY EXAMPLES	EXPLANATION
INSERT INTO PRODUCT SELECT * FROM P;	Inserts all rows from Table P into the PRODUCT table. Both tables must have the same attributes. The subquery returns all rows from Table P.
UPDATE PRODUCT SET P_PRICE = (SELECT AVG(P_PRICE) FROM PRODUCT) WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Updates the product price to the average product price, but only for products provided by vendors who have an area code equal to 615. The first subquery returns the average price; the second subquery returns the list of vendors with an area code equal to 615.
DELETE FROM PRODUCT WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Deletes the PRODUCT table rows provided by vendors with an area code equal to 615. The subquery returns the list of vendor codes with an area code equal to 615.

Cengage Learning © 2015

## Subqueries and Correlated Queries

- Subquery is a query inside another query
- Subquery can return:
  - One single value - One column and one row
  - A list of values - One column and multiple rows
  - A virtual table - Multicolumn, multirow set of values
  - No value - Output of the outer query might result in an error or a null empty set

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 'WHERE' subqueries

## WHERE Subqueries

- Uses inner SELECT subquery on the right side of a WHERE comparison expression
- Value generated by the subquery must be of a comparable data type
- If the query returns more than a single value, the DBMS will generate an error
- Can be used in combination with joins

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

9

# WHERE subquery example

FIGURE 8.13 WHERE subquery example

```
SQL> SELECT P_CODE, P_PRICE FROM PRODUCT
  2 WHERE P_PRICE >= (SELECT AVG(P_PRICE) FROM PRODUCT);
P_CODE          P_PRICE
-----        -----
112ER/31         109.99
22232/TIV         109.92
2232/TME         95.87
80-URE-Q         255.99
UR0/TT0         119.95

SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2 FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
  3           JOIN LINE USING (INV_NUMBER)
  4           JOIN PRODUCT USING (P_CODE)
  5 WHERE P_CODE IN (SELECT P_CODE FROM PRODUCT WHERE P_DESCRIFT = 'Claw hammer');
CUS_CODE CUS_LNAME      CUS_FNAME
-----  -----
10011 Dunne            Leona
10014 Orlando          Myron
```

# IN, HAVING subqueries

## IN and HAVING Subqueries

- IN subqueries
  - Used to compare a single attribute to a list of values
- HAVING subqueries
  - HAVING clause restricts the output of a GROUP BY query by applying conditional criteria to the grouped rows

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

11

# 'IN' subqueries

Compare against a LIST of values..

FIGURE  
8.14 IN subquery example

The screenshot shows the Oracle SQL\*Plus interface. The title bar says "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area displays the following SQL query and its results:

```
SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2  FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
  3  JOIN LINE USING (INV_NUMBER)
  4  JOIN PRODUCT USING (P_CODE)
  5 WHERE P_CODE IN (SELECT P_CODE FROM PRODUCT
  6   WHERE P_DESCRIP LIKE '%hammer%' OR P_DESCRIP LIKE '%saw%');

  CUS_CODE CUS_LNAME      CUS_FNAME
  -----  -----
  10011 Duane          Leona
  10012 Smith          Kathy
  10014 Orlando        Byron
  10015 O'Brian        Amy
```

The results show four customer records whose purchased products either contain the word "hammer" or "saw" in their descriptions.

# 'HAVING' subqueries

As we saw earlier, this restricts the results of a GROUP BY clause. Eg. here's how to list all products sold, whose totals are greater than the average quantity sold:

FIGURE 8.15 HAVING subquery example

The screenshot shows an Oracle SQL\*Plus window. The query is:

```
SQL> SELECT P_CODE, SUM(LINE_UNITS)
  2  FROM LINE
  3  GROUP BY P_CODE
  4  HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS) FROM LINE);
```

The output is:

P_CODE	SUM(LINE_UNITS)
13-Q2/P2	8
23109-ID	5
54778-2T	6
PU023DRT	17
SM-18277	3
WR9/TT9	9

6 rows selected.

SQL>

# ALL, ANY (inequality comparisons)

Recall that 'IN' is an equality comparison against a list. To do **inequality comparison of a value against a list of values** (eg. need to be greater than ALL, need to be less than ANY..), use ALL, ANY.

## Multirow Subquery Operators: ANY and ALL

- ALL operator
  - Allows comparison of a single value with a list of values returned by the first subquery
  - Uses a comparison operator other than equals
- ANY operator
  - Allows comparison of a single value to a list of values and selects only the rows for which the value is greater than or less than any value in the list

# ALL, ANY

Eg. "which products do we own [in our store], whose value is more than ALL other products's values supplied by vendors in Florida?"

FIGURE 8.16 Multirow subquery operator example

```

SQL> SELECT P_CODE, P_QUOH*P_PRICE
  2  FROM PRODUCT
  3  WHERE P_QUOH*P_PRICE > ALL
  4  (SELECT P_QUOH*P_PRICE FROM PRODUCT
  5  WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_STATE = 'FL'));

```

P_CODE	P_QUOH*P_PRICE
B9-WHE-Q	2820.89

Note that 'greater than ALL' is eqvt to 'greater than the largest of'. 'ALL' is used to select rows [plural in general] that comparison-succeed against all values in a list.

Another powerful operator is the ANY multirow operator (the near cousin of the ALL multirow operator). The ANY operator allows you to compare a single value to a list of values, selecting only the rows for which the inventory cost is greater than any value of the list or less than any value of the list. You could use the equal to ANY operator, which would be the equivalent of the IN operator.

'ANY' is used to select rows [plural in general] that comparison-succeed with any value in a list.

Note that ' $= \text{ANY}(\text{list of values})$ ' is equivalent to the ' $\text{IN}$ ' operator (which is itself equivalent to multiple  $=$  conditions joined by ORs). So the following are all equivalent, for a given value of 'M':

**(M==6) OR (M==8) OR (M==10)**

**M IN (6,8,10)**

**M = ANY (6,8,10)**

So loosely speaking, ALL is equivalent to AND, and ANY is equivalent to OR.

# 'FROM' subqueries

A SELECT query that appears in FROM, creates a \*\*virtual table\*\* against which the main query can run.

## FROM Subqueries

- FROM clause:
  - Specifies the tables from which the data will be drawn
  - Can use SELECT subquery

# FROM subquery example

All customers who bought both specified products

FIGURE 8.17 FROM subquery example

```
+ Oracle SQL*Plus
File Edit Preferences Help
SQL> SELECT DISTINCT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME
  2  FROM CUSTOMER,
  3  (SELECT INVOICE.CUS_CODE
  4   FROM INVOICE
  5   WHERE INVOICE.P_CODE = '19-627-02') CP1,
  6  (SELECT INVOICE.CUS_CODE
  7   FROM INVOICE
  8   WHERE INVOICE.P_CODE = '2019-00') CP2
  9  WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE AND
10    CP1.CUS_CODE = CP2.CUS_CODE;
CUS_CODE CUS_LNAME
-----10014 Orlando
SQL> |
```

# Attribute list subqueries

These subqueries determine what columns get output by the main query - they can be actual (existing) columns or computed columns or results of aggregate functions.

These are also known as 'column subqueries' or 'inline subqueries'.

## Attribute List Subqueries

- SELECT statement uses attribute list to indicate what columns to project in the resulting set
- Inline subquery
  - Subquery expression included in the attribute list that must return one value
- Column alias cannot be used in attribute list computation if alias is defined in the same attribute list

# Attribute subquery example

FIGURE 8.18 Inline subquery example

The screenshot shows the Oracle SQL\*Plus interface. The command window contains the following SQL query:

```
SQL> SELECT P_CODE, P_PRICE, (SELECT AVG(P_PRICE) FROM PRODUCT) AS AVEPRICE,
2          P_PRICE-(SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF
3     FROM PRODUCT;
```

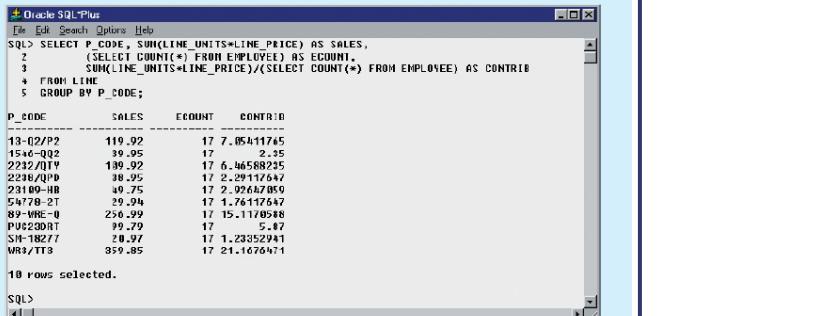
The results window displays the output of the query, which includes columns P\_CODE, P\_PRICE, AVEPRICE, and DIFF. The data shows various product codes and their prices relative to the average price of 56.42125.

P_CODE	P_PRICE	AVEPRICE	DIFF
11QEVR31	189.99	56.42125	133.56875
13-Q2/P2	1h.90	56.42125	-41.53125
14-Q1/L3	17.49	56.42125	-38.93125
15AB-02	39.99	56.42125	-17.47125
15AB-0011	46.99	56.42125	-10.44875
2222/01W	189.99	56.42125	133.56875
2232/0ME	99.97	56.42125	43.44875
2238/0FD	38.95	56.42125	-17.47125
23109-HI	9.95	56.42125	-46.47125
23114-AA	14.4	56.42125	-42.02125
5477B-2T	4.99	56.42125	-51.43125
89-00000	250.0	56.42125	193.56875
PV0239MT	5.97	56.42125	-50.44875
SII-1M2Z77	6.99	56.42125	-49.43125
SU-23116	8.95	56.42125	-47.07125
VR3/113	119.95	56.42125	63.52875

16 ROWS Selected.

# Another attribute subquery example

FIGURE 8.19 Another example of an inline subquery



The screenshot shows the Oracle SQL\*Plus interface with a query window. The query is:

```
SQL> SELECT P_CODE, SUM(LINE_UNITS*LINE_PRICE) AS SALES,
  2      (SELECT COUNT(*) FROM EMPLOYEE) AS ECOUNT,
  3      SUM(LINE_UNITS*LINE_PRICE)/(SELECT COUNT(*) FROM EMPLOYEE) AS CONTRIB
  4  FROM LINE
  5 GROUP BY P_CODE;
```

The output shows 10 rows of data:

P_CODE	SALES	ECOUNT	CONTRIB
13-02/P2	119.92	17	7.05411745
1546-Q02	39.95	17	2.35
2232/J01Y	119.92	17	6.46588235
2238/Q/PB	38.95	17	2.29117647
23100-HB	49.75	17	2.92627059
5478/E-0T	29.91	17	1.76117647
6939-E-0	26.99	17	15.117647
P02200RT	99.79	17	5.74
SM-1RZ77	78.97	17	1.23352941
VR3/TT3	359.85	17	21.1676471

10 rows selected.

SQL>

# Correlated subqueries

## Correlated Subquery

- Executes once for each row in the outer query
- Inner query references a column of the outer subquery
- Can be used with the EXISTS special operator

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

21

In a correlated subquery, the inner (sub) query is repeatedly run, for each row of the outer query! The inner is said to be (co-)related with the outer query when it references a column in the outer query's table. This is in effect, like a double (nested) 'for' loop..

Here is the Wikipedia entry on correlated subqueries. This is the example shown there [select employees who make more than the average salary for their department]:

```
SELECT employee_number, name
  FROM employees AS Bob
 WHERE salary > (
   SELECT AVG(salary)
     FROM employees
    WHERE department = Bob.department);
```

In the above, the outer query "passes in", for each employee (each row), the employee's dept. [which the inner query refers to as Bob.department]. The inner query selects all salaries for that dept., computes the average, compares it with the passed-in employee's salary; if the test passes, the outer query selects the employee's # and name.

# Correlated subqueries [cont'd]

Until now, all subqueries you have learned execute independently. That is, each subquery in a command sequence executes in a serial fashion, one after another. The inner subquery executes first; its output is used by the outer query, which then executes until the last outer query executes (the first SQL statement in the code).

In contrast, a **correlated subquery** is a subquery that executes once for each row in the outer query. That process is similar to the typical nested loop in a programming language. For example:

```
FOR X = 1 TO 2
  FOR Y = 1 TO 3
    PRINT "X = "X, "Y = "Y
  END
END
```

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of that of the subqueries as you have already seen. The query is called a *correlated subquery* because the inner query is *related* to the outer query by the fact that the inner query references a column of the outer subquery.

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Correlated subquery examples

To see the correlated subquery in action, suppose that you want to know all product sales in which the units sold value is greater than the average units sold value *for that product* (as opposed to the average for *all* products). In that case, the following procedure must be completed:

1. Compute the average units sold for a product.
2. Compare the average computed in Step 1 to the units sold in each sale row, and then select only the rows in which the number of units sold is greater.

The following correlated query completes the preceding two-step process:

```
SELECT INV_NUMBER, P_CODE, LINE_UNITS
FROM LINE LS
WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
                         FROM LINE LA
                         WHERE LA.P_CODE = LS.P_CODE);
```

```
SQL> SELECT INV_NUMBER, P_CODE, LINE_UNITS
  2  FROM LINE LS
  3 WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
  4                           FROM LINE LA
  5                           WHERE LA.P_CODE = LS.P_CODE);

INV_NUMBER P_CODE      LINE_UNITS
----- ----- -----
1003 13-Q2/P2          5
1004 54778-2T          3
1004 23109-HB          2
1005 PUC23DRT          12

SQL> SELECT INV_NUMBER, P_CODE, LINE_UNITS,
  2    (SELECT AVG(LINE_UNITS) FROM LINE LX WHERE LX.P_CODE = LS.P_CODE) AS AVG
  3  FROM LINE LS
  4 WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
  5                           FROM LINE LA
  6                           WHERE LA.P_CODE = LS.P_CODE);

INV_NUMBER P_CODE      LINE_UNITS      AVG
----- ----- ----- -----
1003 13-Q2/P2          5  2.66666667
1004 54778-2T          3   2
1004 23109-HB          2   1.25
1005 PUC23DRT          12   8.5
```

In the top query and its result in Figure 8.14, note that the LINE table is used more than once, so you must use table aliases. In this case, the inner query computes the average units sold of the product that matches the P\_CODE of the outer query P\_CODE. That is, the inner query runs once, using the first product code found in the outer LINE table, and returns the average sale for that product. When the number of units sold in the outer LINE row is greater than the average computed, the row is added to the output. Then the inner query runs again, this time using the second product code found in the outer LINE table. The process repeats until the inner query has run for all rows in the outer LINE table. In this case, the inner query will be repeated as many times as there are rows in the outer query.

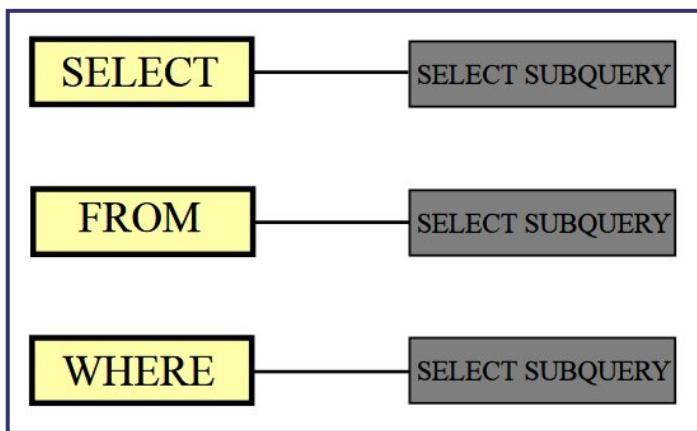
To verify the results and to provide an example of how you can combine subqueries, you can add a correlated inline subquery to the previous query. (See the second query and its results in Figure 8.14.) As you can see, the new query contains a correlated inline subquery that computes the average units sold for each product. You not only get an answer, you can also verify that the answer is correct.

**In the second query above, we have TWO correlated subqueries (that are identical), both of which need to run for every row of the main query.**

# Queries: summary

We looked at several variations of queries and subqueries (SELECT, WHERE, HAVING, IN..).

Most interestingly, a SELECT subquery can appear at the top (SELECT), middle (FROM) or bottom (WHERE) of a parent query, which provides a flexible way to express complex logic (since such subqueries can be recursively nested):



# SQL functions

## SQL Functions

- Functions always use a numerical, date, or string value
- Value may be part of a command or may be an attribute located in a table
- Function may appear anywhere in an SQL statement where a value or an attribute can be used

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

24

# SQL functions [cont'd]

SQL Functions

- Date and time functions
- Numeric functions
- String functions
- Conversion functions

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# UNION, INTERSECTION, DIFFERENCE

## Relational Set Operators

- SQL data manipulation commands are set-oriented
  - **Set-oriented:** Operate over entire sets of rows and columns at once
- UNION, INTERSECT, and Except (MINUS) work properly when relations are union-compatible
  - **Union-compatible:** Number of attributes are the same and their corresponding data types are alike
- UNION
  - Combines rows from two or more queries without including duplicate rows

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

26

# UNION, INTERSECTION, DIFFERENCE

## [cont'd]

### Relational Set Operators

- Syntax - query UNION query
- UNION ALL
  - Produces a relation that retains duplicate rows
  - Can be used to unite more than two queries
- INTERSECT
  - Combines rows from two queries, returning only the rows that appear in both sets
  - Syntax - query INTERSECT query

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, in whole or in part.

27

# UNION, INTERSECTION, DIFFERENCE

## [cont'd]

### Relational Set Operators

- EXCEPT (MINUS)
  - Combines rows from two queries and returns only the rows that appear in the first set
  - Syntax
    - query EXCEPT query
    - query MINUS query
- Syntax alternatives
  - IN and NOT IN subqueries can be used in place of INTERSECT

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

28

# VIEWS

## Virtual Tables: Creating a View

- **View:** Virtual table based on a SELECT query
- **Base tables:** Tables on which the view is based
- **CREATE VIEW** statement: Data definition command that stores the subquery specification in the data dictionary
  - CREATE VIEW command
    - CREATE VIEW viewname AS SELECT query

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

29

# VIEW example

Creating a Virtual Table with the CREATE VIEW Command

The screenshot shows a window titled "SQL Plus". Inside, the following SQL commands are run:

```
SQL> CREATE VIEW PRICEGT50 AS
  2    SELECT P_DESCRPT, P_QOH, P_PRICE
  3    FROM PRODUCT
  4   WHERE P_PRICE > 50.00;

View created.

SQL> SELECT * FROM PRICEGT50;
```

The output of the second command is a table:

P_DESCRPT	P_QOH	P_PRICE
Power painter, 15 psi., 3-nozzle	8	109.99
B&D jigsaw, 12-in. blade	8	109.92
B&D jigsaw, 8-in. blade	6	99.87
Hicut chain saw, 16 in.	11	256.99
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95

At the bottom of the window, it says "Cengage Learning © 2015".

# Sequences

## Oracle Sequences

- Independent object in the database
- Have a name and can be used anywhere a value expected
- Not tied to a table or column
- Generate a numeric value that can be assigned to any column in any table
- Table attribute with an assigned value can be edited and modified
- Can be created and deleted any time

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

33

# Sequence creation example

Figure 8.27 - Oracle Sequence

The screenshot shows the Oracle SQL Plus interface. At the top, it says "Figure 8.27 - Oracle Sequence". The SQL command window contains the following:

```
SQL> CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;
Sequence created.

SQL> CREATE SEQUENCE INU_NUMBER_SEQ START WITH 4010 NOCACHE;
Sequence created.

SQL> SELECT * FROM USER_SEQUENCES;
```

A table is displayed with the following data:

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	CACHE_SIZE	LAST_NUMBER
CUS_CODE_SEQ	1 1.0000E+27	1	N	0	20010
INU_NUMBER_SEQ	1 1.0000E+27	1	N	0	4010

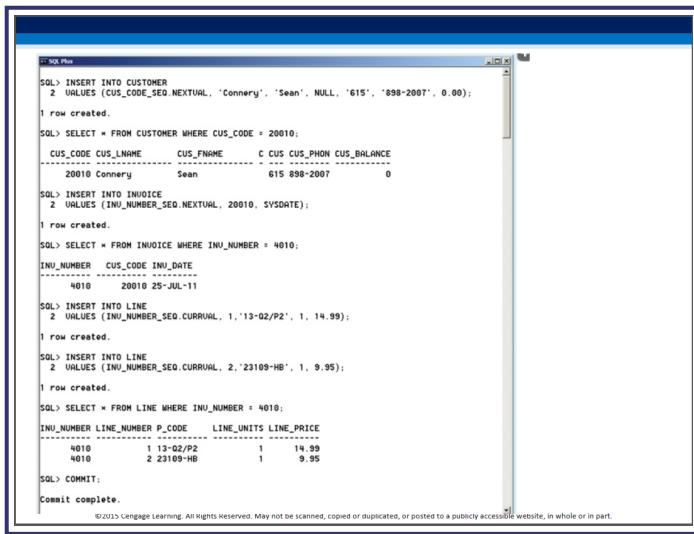
```
SQL>

INSERT INTO CUSTOMER
VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);
```

At the bottom left, it says "©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part." At the bottom right, it says "34".

# Sequence: NEXTVAL, CURRVAL

NEXTVAL returns the current value, then does ++;  
CURRVAL just fetches the current value (does not ++ it).



The screenshot shows a SQL Plus window with the following session history:

```
-- SQL Plus
SQL> INSERT INTO CUSTOMER
2  VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);
1 row created.

SQL> SELECT * FROM CUSTOMER WHERE CUS_CODE = 20010;
   CUS_CODE CUS_LNAME      CUS_FNAME      C CUS_CUS_PHON CUS_BALANCE
----- -----          -----          C-----          -----
  20010 Connery        Sean           615 898-2007          0

SQL> INSERT INTO INVOICE
2  VALUES (INU_NUMBER_SEQ.NEXTVAL, 20010, SYSDATE);
1 row created.

SQL> SELECT * FROM INVOICE WHERE INU_NUMBER = 4010;
   INU_NUMBER CUS_CODE INU_DATE
----- -----          -----
  4010       20010 25-JUL-11

SQL> INSERT INTO LINE
2  VALUES (INU_NUMBER_SEQ.CURRVAL, 1,'13-02/P2', 1, 14.99);
1 row created.

SQL> INSERT INTO LINE
2  VALUES (INU_NUMBER_SEQ.CURRVAL, 2,'23109-HB', 1, 9.95);
1 row created.

SQL> SELECT * FROM LINE WHERE INU_NUMBER = 4010;
   INU_NUMBER LINE_NUMBER P_CODE      LINE_UNITS LINE_PRICE
----- -----          -----          C-----          -----
  4010       1 13-02/P2            1    14.99
  4010       2 23109-HB           1     9.95

SQL> COMMIT;
Commit complete.
```

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Procedural Language SQL (PL/SQL)

PL/SQL involves extra (augmented) syntax that lets us do looping, branching, variable declaration and function declaration - these are of course not possible using 'plain' SQL.

PL/SQL can be used to create:

- **blocks of code** for one-time execution
- **triggers** - callbacks to invoke
- **stored procedures** - named procedures (no return values) for repeated calling
- **stored functions** - named functions (with return values) for repeated calling

## Procedural SQL

- Performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
  - Yields better maintenance and logic control
- **Persistent stored module (PSM):** Block of code containing:
  - Standard SQL statements
  - Procedural extensions that is stored and executed at the DBMS server

# PL/SQL [cont'd]

## Procedural SQL

- **Procedural Language SQL (PL/SQL)**

- Use and storage of procedural code and SQL statements within the database
- Merging of SQL and traditional programming constructs
- Procedural code is executed as a unit by DBMS when invoked by end user
- End users can use PL/SQL to create:
  - Anonymous PL/SQL blocks and triggers
  - Stored procedures and PL/SQL functions

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

37

# [Unnamed] block creation example

The screenshot shows a SQL developer interface with the following content:

```
SQL> BEGIN
 2 INSERT INTO VENDOR
 3 VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'WA', 'N');
 4 END;
 5 /
PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
 2 INSERT INTO VENDOR
 3 VALUES (25772, 'Clue Store', 'Isaac Hayes', '456', '323-2009', 'VA', 'N');
 4 DBMS_OUTPUT.PUT_LINE('New Vendor Added!');
 5 END;
 6 /
New Vendor Added!
PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;
U_CODE U_NAME          U_CONTACT      U_P U_PHONE U_U
----- -- -- -- -- --
25678 Microsoft Corp. Bill Gates    765 546-8484 WA N
25772 Clue Store     Isaac Hayes   615 899-0989 VA N
21225 Bellcore, Inc. Smithson     615 223-3234 TN Y
21226 SuperLoo, Inc. Flushing     904 215-8995 FL N
21231 DEE Supply     Singh        615 228-3245 TN Y
21394 Gomez Bros.  Ortega       615 888-1111 GA N
22695 Global Supply  Smith        801 555-1111 GR N
23119 Randsets Ltd. Anderson    801 678-3998 GR Y
24000 Brackman Bros. Browning    615 228-1410 TN N
24288 ORDUA, Inc.  Hakford      615 898-1234 TN Y
25494 DSK, Inc.     Smith        809 227-1234 TN N
25501 Digital Supplies Seythe      615 888-3529 TN N
25595 Rubicon Systems Orion       904 456-0092 FL Y
13 rows selected.
```

SQL>

# Triggers

## Triggers

- Procedural SQL code automatically invoked by RDBMS when given data manipulation event occurs
- Parts of a trigger definition
  - Triggering timing - Indicates when trigger's PL/SQL code executes
  - Triggering event - Statement that causes the trigger to execute
  - Triggering level - **Statement-** and **row-level**
  - Triggering action - PL/SQL code enclosed between the BEGIN and END keywords

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

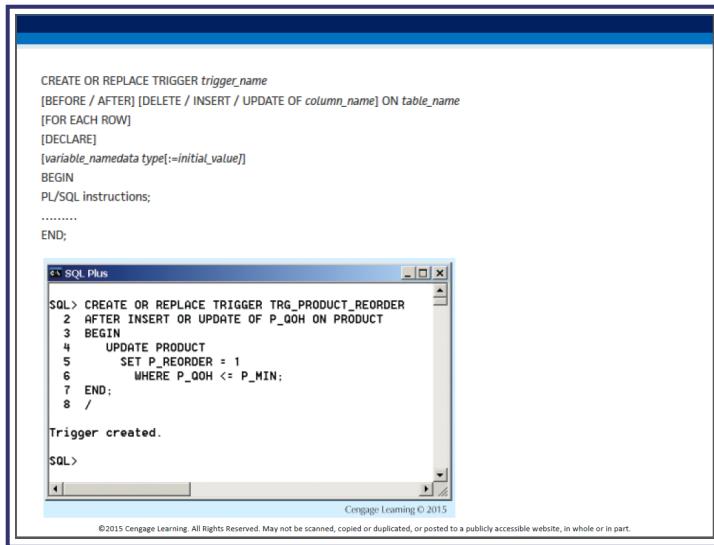
39

# Triggers [cont'd]

- *The triggering timing:* BEFORE or AFTER. This timing indicates when the trigger's PL/SQL code executes—in this case, before or after the triggering statement is completed.
- *The triggering event:* The statement that causes the trigger to execute (INSERT, UPDATE, or DELETE).
- *The triggering level:* The two types of triggers are statement-level triggers and row-level triggers.
  - A **statement-level trigger** is assumed if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed. This is the default case.
  - A **row-level trigger** requires use of the FOR EACH ROW keywords. This type of trigger is executed once for each row affected by the triggering statement. (In other words, if you update 10 rows, the trigger executes 10 times.)
- *The triggering action:* The PL/SQL code enclosed between the BEGIN and END keywords. Each statement inside the PL/SQL code must end with a semicolon ( ; ).

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Trigger example



The screenshot shows a computer screen with two windows. The top window is a code editor with the following PL/SQL code:

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE / AFTER] [DELETE / INSERT / UPDATE OF column_name] ON table_name
[FOR EACH ROW]
[DECLARE]
[variable_name data_type[:>initial_value]]
BEGIN
PL/SQL instructions;
.....
END;
```

The bottom window is a SQL Plus session window titled "SQL Plus". It contains the following command and output:

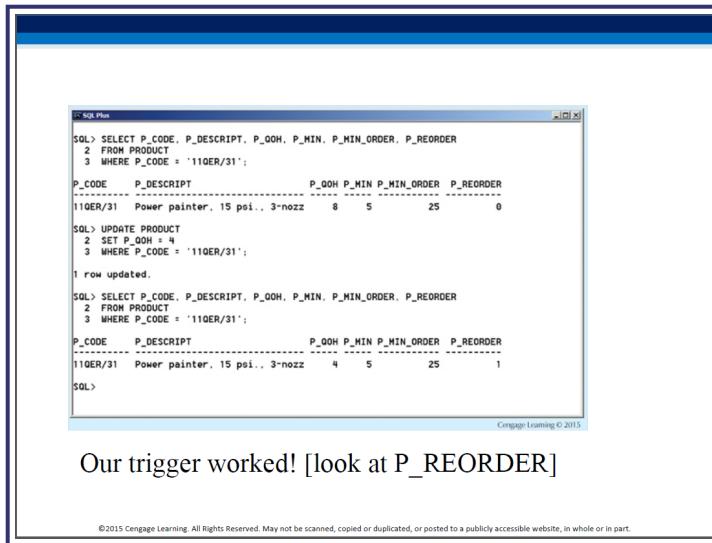
```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
2  AFTER INSERT OR UPDATE OF P_QOH ON PRODUCT
3  BEGIN
4      UPDATE PRODUCT
5          SET P_reordered = 1
6          WHERE P_QOH <= P_Min;
7  END;
8 /
```

Trigger created.

SQL>

Cengage Learning © 2015

# Trigger example



The screenshot shows a SQL Plus window with the following session history:

```
SQL> SELECT P_CODE, P_DESCRIP, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT
  3 WHERE P_CODE = '11QER/31';

P_CODE      P_DESCRIP          P_QOH P_MIN P_MIN_ORDER P_REORDER
-----  -----
11QER/31  Power painter, 15 psi., 3-nozz     8      5        25          0

SQL> UPDATE PRODUCT
  2  SET P_QOH = 4
  3 WHERE P_CODE = '11QER/31';

1 row updated.

SQL> SELECT P_CODE, P_DESCRIP, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT
  3 WHERE P_CODE = '11QER/31';

P_CODE      P_DESCRIP          P_QOH P_MIN P_MIN_ORDER P_REORDER
-----  -----
11QER/31  Power painter, 15 psi.. 3-nozz     4      5        25          1

SQL>
```

Our trigger worked! [look at P\_REORDER]

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Triggers [cont'd]

## Triggers

- **DROP TRIGGER trigger\_name command**
  - Deletes a trigger without deleting the table
- Trigger action based on DML predicates
  - Actions depend on the type of DML statement that fires the trigger

# Stored procedures

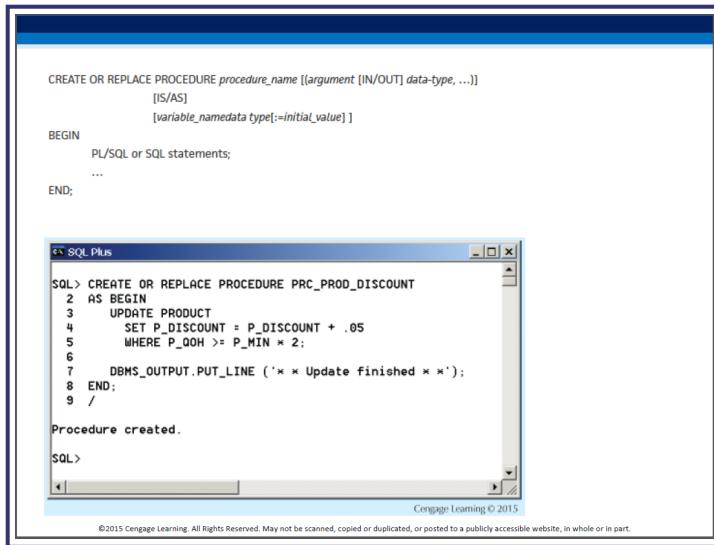
## Stored Procedures

- Named collection of procedural and SQL statements
- Advantages
  - Reduce network traffic and increase performance
  - Reduce code duplication by means of code isolation and code sharing

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

44

# Stored procedure example



The screenshot shows a Windows application window titled "SQL\*Plus". Inside the window, PL/SQL code is being entered to create a stored procedure named "PRC\_PROD\_DISCOUNT". The code includes a BEGIN block containing an UPDATE statement that adds 0.05 to the product discount if quantity on hand is greater than or equal to twice the minimum quantity. It also includes a DBMS\_OUTPUT.PUT\_LINE call to print a message when the update is finished. The command "CREATE OR REPLACE PROCEDURE" is preceded by a multi-line comment describing the syntax for creating or replacing a procedure.

```
CREATE OR REPLACE PROCEDURE procedure_name [(argument [IN/OUT] data-type, ...)]  
[IS/AS]  
[variable_namedata type[:=initial_value]] ]  
  
BEGIN  
    PL/SQL or SQL statements;  
    ...  
END;
```

```
SQL> CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT  
2  AS BEGIN  
3      UPDATE PRODUCT  
4          SET P_DISCOUNT = P_DISCOUNT + .05  
5          WHERE P_QOH >= P_MIN * 2;  
6  
7      DBMS_OUTPUT.PUT_LINE ('* * Update finished * *');  
8  END;  
9 /  
  
Procedure created.  
SQL>
```

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Stored procedure example

The screenshot shows two identical SQL Plus windows side-by-side. Both windows display the same command and its execution results.

```

SQL> SELECT P_CODE, P_DESCRIFT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
P_CODE          P_DESCRIFT          P_QOH P_MIN P_DISCOUNT
11022-31 Power painter, 15 psi., 3-nozz 29      5     0.05
13-25-42 7.25-in. per. saw blade    32      15     0.05
14-41-43 9.00-in. per. saw blade    18      12     0.00
156-Q06 Red cloth, 1-4-in., 2x50   15      0     0.00
155-Q08 Blue cloth, 1-4-in., 2x50   25      0     0.00
223-Q7Y BBM jigsaw, 12-in. blade   8       5     0.05
223-Q7Z BBM jigsaw, 12-in. blade   6       5     0.05
228-Q7B BBM carbide #4111, 1/2-in. 12      5     0.05
23199-40 Close hanger, 12-in.     25      10     0.10
23114-40 Close hanger, 12-in.     6       10     0.05
54778-21 Rat-tail file, 1/8-in., fine 43      20     0.00
25238-Q01 PVC pipe, 3/8-in., 8-ft    108     75     0.00
207-Q07 PVC pipe, 3/8-in., 8-ft    108     75     0.00
2M-2116 2.5-in. wd. screw, 50     237     300     0.05
M83-T10 Steel setting, 4"x0.05x1.6", .5" 10      5     0.10
16 rows selected.

SQL> EXEC PRC_PROD_DISCOUNT;
* + Update finished = *

PL/SQL procedure successfully completed.

SQL> SELECT P_CODE, P_DESCRIFT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
P_CODE          P_DESCRIFT          P_QOH P_MIN P_DISCOUNT
11022-31 Power painter, 15 psi., 3-nozz 29      5     0.05
13-25-42 7.25-in. per. saw blade    32      15     0.05
14-41-43 9.00-in. per. saw blade    18      12     0.00
156-Q06 Red cloth, 1-4-in., 2x50   15      0     0.00
155-Q08 Blue cloth, 1-4-in., 2x50   25      0     0.00
223-Q7Y BBM jigsaw, 12-in. blade   8       5     0.05
223-Q7Z BBM jigsaw, 12-in. blade   6       5     0.05
228-Q7B BBM carbide #4111, 1/2-in. 12      5     0.10
23199-40 Close hanger, 12-in.     25      10     0.15
23114-40 Close hanger, 12-in.     6       10     0.05
54778-21 Rat-tail file, 1/8-in., fine 43      20     0.00
25238-Q01 PVC pipe, 3/8-in., 8-ft    108     75     0.00
207-Q07 PVC pipe, 3/8-in., 8-ft    108     75     0.00
2M-2116 2.5-in. wd. screw, 50     237     300     0.05
M83-T10 Steel setting, 4"x0.05x1.6", .5" 10      5     0.15
16 rows selected.

SQL>

```

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Stored functions

Reminder - these can RETURN a value.

## PL/SQL Stored Functions

- **Stored function:** Named group of procedural and SQL statements that returns a value
  - As indicated by a RETURN statement in its program code
- Can be invoked only from within stored procedures or triggers

# Stored functions - syntax

```
CREATE FUNCTION function_name (argument IN data-type, ...) RETURN data-type [IS]
BEGIN
    PL/SQL statements;
    ...
    RETURN (value or expression);
END;
```

Once such a function is defined, it can be CALLED  
inside triggers or in stored procedures..

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Stored functions - example

The following is an example from  
<http://www.tutorialspoint.com/plsql>.

Creating/defining a function:

```
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
    IF x > y THEN
        z:= x;
    ELSE
        Z:= y;
    END IF;

    RETURN z;
END;
```

Calling/executing/running the function:

```
DECLARE
    a number;
    b number;
    c number;
BEGIN
    a:= 23;
    b:= 45;
```

```
c := findMax(a, b);
dbms_output.put_line(' Maximum of
(23,45): ' || c);
END;
/
```

Result:

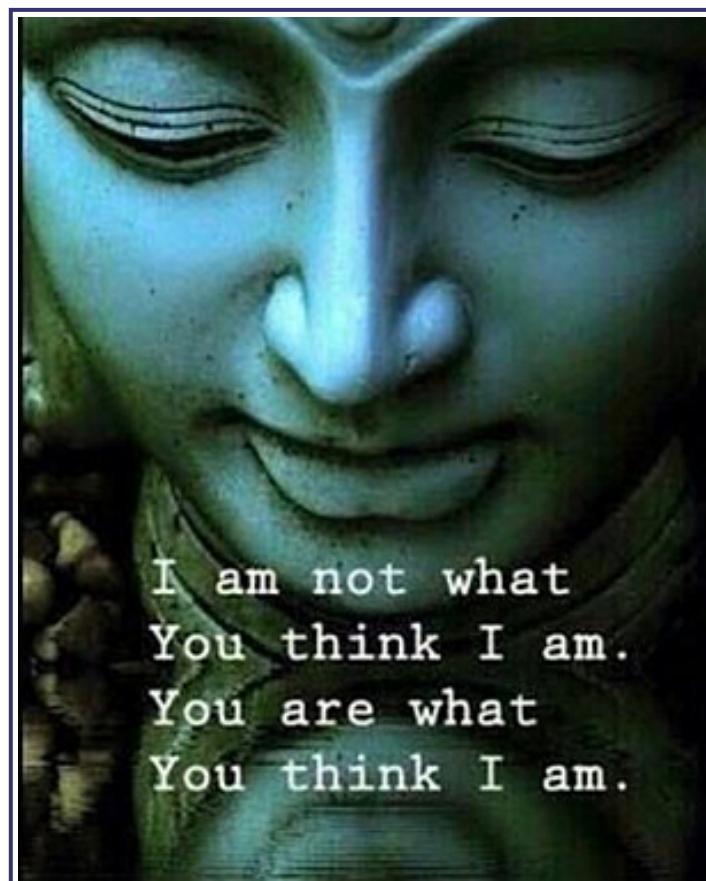
```
Maximum of (23,45): 45
```

1/30 10:06:33 \*\*\*

← →

# TM

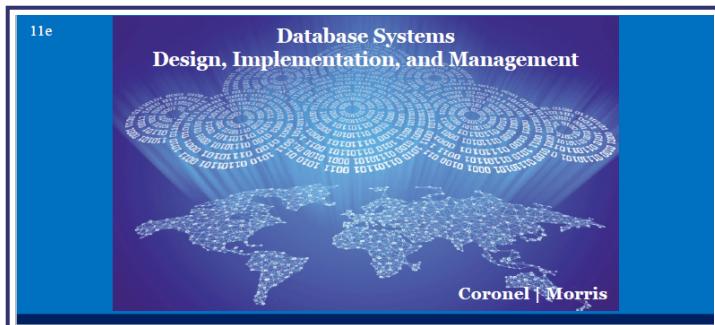
## [Transaction Management]



A different kind of TM :)



# Ch.10



11e

**Database Systems**  
Design, Implementation, and Management

Coronel | Morris

Chapter 10  
Transaction Management and  
Concurrency Control

# What is a 'transaction'?

## Transaction

- Logical unit of work that must be entirely completed or aborted
- Consists of:
  - SELECT statement
  - Series of related UPDATE statements
  - Series of INSERT statements
  - Combination of SELECT, UPDATE, and INSERT statements

## Transaction

- **Consistent database state:** All data integrity constraints are satisfied
  - Must begin with the database in a known consistent state to ensure consistency
- Formed by two or more database requests
  - **Database requests:** Equivalent of a single SQL statement in an application program or transaction
- Consists of a single SQL statement or a collection of related SQL statements

# 'ACID'

'ACID' (+S) is an acronym to express the desirable properties of a transaction:

The diagram is a slide titled "Transaction Properties". It lists five properties with their descriptions. The "Isolation" property is highlighted with a red border around its title and bullet points.

Transaction Properties	
Atomicity	<ul style="list-style-type: none"><li>All operations of a transaction must be completed</li><li>If not, the transaction is aborted</li></ul>
Consistency	<ul style="list-style-type: none"><li>Permanence of database's consistent state</li></ul>
Isolation	<ul style="list-style-type: none"><li>Data used during transaction cannot be used by second transaction until the first is completed</li></ul>
Durability	<ul style="list-style-type: none"><li>Ensures that once transactions are committed, they cannot be undone or lost</li></ul>
Serializability	<ul style="list-style-type: none"><li>Ensures that the schedule for the concurrent execution of several transactions should yield consistent results</li></ul>

## Transaction Management with SQL

- SQL statements that provide transaction support
  - COMMIT
  - ROLLBACK
- Transaction sequence must continue until:
  - COMMIT statement is reached
  - ROLLBACK statement is reached
  - End of program is reached
  - Program is abnormally terminated

# Tracking updates

## Transaction Log

- Keeps track of all transactions that update the database
- DBMS uses the information stored in a log for:
  - Recovery requirement triggered by a ROLLBACK statement
  - A program's abnormal termination
  - A system failure

A Transaction Log

TRL_ID	TRX_NUM	PREV_PTR	NEXT_PTR	OPERATION	TABLE	ROW_ID	ATTRIBUTE	BEFORE_VALUE	AFTER_VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				



TRL\_ID = Transaction log record ID  
TRX\_NUM = Transaction number  
PTR = Pointer to a transaction log record ID  
(Note: The transaction number is automatically assigned by the DBMS.)

Cengage Learning © 2015

# Concurrency - 'many at once'

## Concurrency Control

- Coordination of the simultaneous transactions execution in a multiuser database system
- Objective - Ensures serializability of transactions in a multiuser database environment

## Problems in Concurrency Control

### Lost update

- Occurs in two concurrent transactions when:
  - Same data element is updated
  - One of the updates is lost

### Uncommitted data

- Occurs when:
  - Two transactions are executed concurrently
  - First transaction is rolled back after the second transaction has already accessed uncommitted data

### Inconsistent retrievals

- Occurs when a transaction accesses data before and after one or more other transactions finish working with such data

Consider the following pair of transactions, starting with 35 units of an item: purchase 100 units, then sell 30 units:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	$\text{PROD\_QOH} = \text{PROD\_QOH} + 100$
T2: Sell 30 units	$\text{PROD\_QOH} = \text{PROD\_QOH} - 30$

Cengage Learning © 2015

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	$\text{PROD\_QOH} = 35 + 100$	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	$\text{PROD\_QOH} = 135 - 30$	
6	T2	Write PROD_QOH	105

Cengage Learning © 2015

If the transactions T1 and T2 happen one after another (not concurrently), the PROD\_QOH value would/should be 105, as shown above.

Three different kinds of errors are possible, when interleaved transactions are not handled properly.

# Error #1: lost updates

'Lost update' problem:

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 - 30	
5	T1	Write PROD_QOH (lost update)	135
6	T2	Write PROD_QOH	5

Cengage Learning © 2015

Here, instead of 105, our QOH comes out to be 5, which is incorrect.

## Consider this rollback situation:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH - 30

Cengage Learning © 2015

## Proper rollback (with sequential transactions):

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK*****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

Cengage Learning © 2015

Here the total is 5, which is correct – the QOH goes from 35 to 135, gets rolled back to 35, after which the purchase (of 30 units) happens.

# Error #2: reading uncommitted data

'Uncommitted data' problem (improper rollback):

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 - 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Cedaoae Learning © 2015

Here, the total comes out to 105, when it should have been 5.

Consider the following fix (of a typo made earlier - an incorrect order of 10 units was placed for 1558-QW1 by mistake, instead of ordering 1546-QQ2; now we're fixing that error):

TRANSACTION T1	TRANSACTION T2
SELECT SUM(PROD_QOH) FROM PRODUCT	UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 10 WHERE PROD_CODE = 1546-QQ2
	UPDATE PRODUCT SET PROD_QOH = PROD_QOH - 10 WHERE PROD_CODE = 1558-QW1 COMMIT;

Cengage Learning © 2015

In the above, T1 is a transaction that sums up QOH; T2 is the 'correction' transaction (that fixes the incorrect purchasing).

## Proper retrieval of modified data:

PROD_CODE	BEFORE PROD_QOH	AFTER PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QWT	23	(23 - 10) → 13
2232-QTY	8	8
2232-QWE	6	6
<b>Total</b>	<b>92</b>	<b>92</b>

Cengage Learning © 2015

The summing transaction gets proper totals for both affected products, so the total (of 92) is correct.

# Error #3: improper [premature] retrieval

'Inconsistent retrievals' problem (improper ("before update") retrieval of modified data):

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 - 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

Cengage Learning © 2015

T1 should be doing  $65+13$  (which would be correct), but instead does  $65+23$  (which makes it incorrect) - in other words, T1 retrieves the correct value (25) for 1546-QQ2, but gets the incorrect value (23) for 1558-QW1.

Inconsistent retrieval is also called a 'dirty read'.

Look up: non-repeatable reads, phantom row reads.

# Concurrent, serializable schedule

## The Scheduler

- Establishes the order in which the operations are executed within concurrent transactions
  - Interleaves the execution of database operations to ensure serializability and isolation of transactions
- Based on concurrent control algorithms to determine the appropriate order
- Creates serialization schedule
  - **Serializable schedule:** Interleaved execution of transactions yields the same results as the serial execution of the transactions

A serializable schedule makes concurrency immaterial (non-issue).

## Concurrency Control with Locking Methods

- Locking methods - Facilitate isolation of data items used in concurrently executing transactions
- **Lock:** Guarantees exclusive use of a data item to a current transaction
- **Pessimistic locking:** Use of locks based on the assumption that conflict between transactions is likely
- **Lock manager:** Responsible for assigning and policing the locks used by the transactions

TRL pessimistic 'lock' situations: restrooms, RCS, Robert's Rules Of Order..

# Locking: granularity

## Lock Granularity

- Indicates the level of lock use
- Levels of locking
  - **Database-level lock**
  - **Table-level lock**
  - **Page-level lock**
    - **Page or diskpage:** Directly addressable section of a disk
  - **Row-level lock**
  - **Field-level lock**

Figure 10.3 - Database-Level Locking Sequence

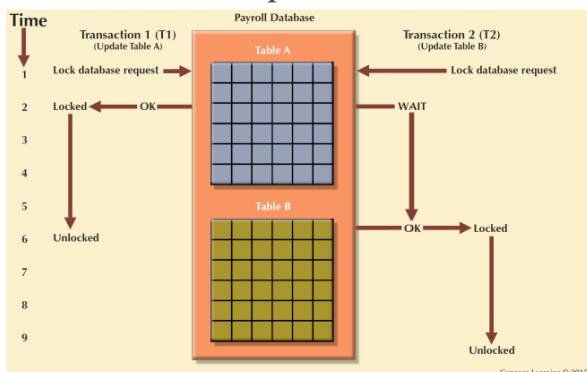


Figure 10.4 - An Example of a Table-Level Lock

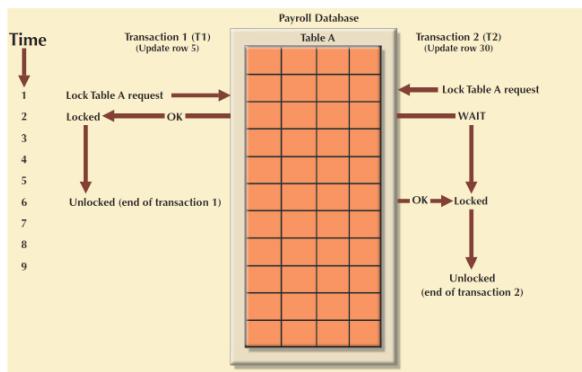


Figure 10.5 - An Example of a Page-Level Lock

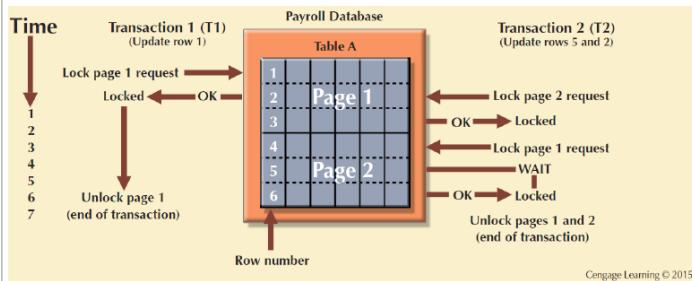
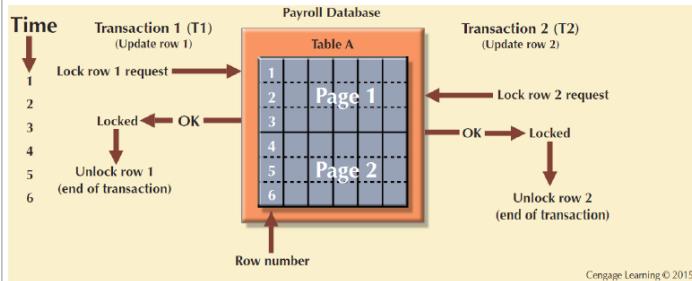


Figure 10.6 - An Example of a Row-Level Lock



# Locking: types [unlocked, locked\_read, locked\_write]

## Lock Types

- Binary lock**
  - Has two states, locked (1) and unlocked (0)
    - If an object is locked by a transaction, no other transaction can use that object
    - If an object is unlocked, any transaction can lock the object for its use
- Exclusive lock**
  - Exists when access is reserved for the transaction that locked the object
- Shared lock**
  - Exists when concurrent transactions are granted read access on the basis of a common lock

## Three lock states

- Using the shared/exclusive concept, there are THREE lock states: unlocked, shared (read), exclusive (write)

# Read(shared), write(exclusive) lock

## Shared lock

- Issued when a transaction wants to READ data, and no exclusive lock is held (on a data item)

## Exclusive lock

- Issued when a transaction wants to WRITE data, and no lock is held (on a data item)

# '2PL'

## Two-Phase Locking (2PL)

- Defines how transactions acquire and relinquish locks
- Guarantees serializability but does not prevent deadlocks
- Phases
  - Growing phase - Transaction acquires all required locks without unlocking any data
  - Shrinking phase - Transaction releases all locks and cannot obtain any new lock

The 2PL type that we are discussing, where a transaction acquires all the locks it needs before processing starts, is called 'Conservative' 2PL (or 'Static' 2PL).

## 2PL [cont'd]

## Two-Phase Locking (2PL)

- Governing rules
    - Two transactions cannot have conflicting locks
    - No unlock operation can precede a lock operation in the same transaction
    - No data are affected until all locks are obtained

Once all the locks are acquired, a transaction can proceed 'smoothly', ie won't 'hang' or lead to dirty reads etc.

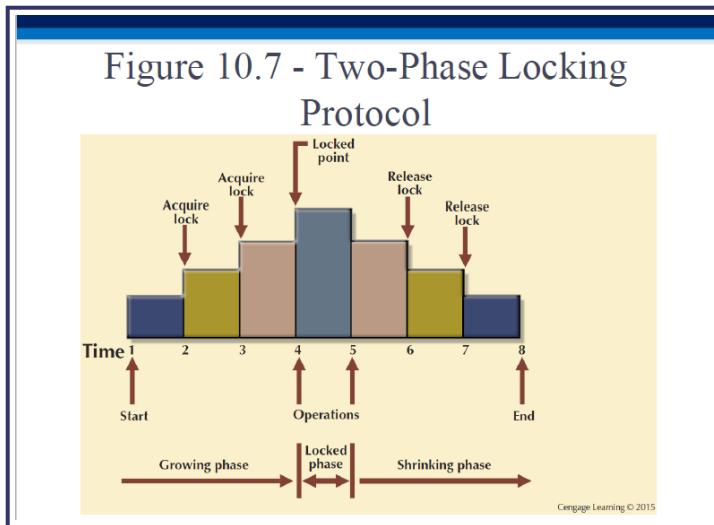
As for the middle point above (unlocking can't precede locking), here is a loose analogy:

OK:

**Not OK (OK in programming, though!):**

```
{  
{  
}  
{  
{  
{  
}  
}  
}  
}  
}
```

# 2PL [cont'd]



# Deadlocks..

## Deadlocks

- Occurs when two transactions wait indefinitely for each other to unlock data
  - Known as **deadly embrace**
- Control techniques
  - Deadlock prevention
  - Deadlock detection
  - Deadlock avoidance
- Choice of deadlock control method depends on database environment

- *Deadlock prevention.* A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.
- *Deadlock detection.* The DBMS periodically tests the database for deadlocks. If a deadlock is found, the "victim" transaction is aborted (rolled back and restarted) and the other transaction continues.
- *Deadlock avoidance.* The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back of conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases action response times.

**Timestamping-based schemes prevent deadlocks; 2PL avoids deadlocks; detection is used in both.**

# How/why a deadlock occurs

Table 10.13 - How a Deadlock Condition is Created				
TIME	TRANSACTION	REPLY	LOCK STATUS	
			Data X	Data Y
0			Unlocked	Unlocked
1	T1:LOCK(X)	OK	Locked	Unlocked
2	T2:LOCK(Y)	OK	Locked	Locked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....

↓  
Deadlock

Cengage Learning © 2015

In the above, we see that T1 locks X, T2 locks Y, then deadlock occurs because T1 wants Y and T2 wants X. Maybe you are thinking - why can't each of them release what they are holding (since they might be done with it), and grab what they want next (ie. T1 would release X, T2 would release Y)? BECAUSE THEY CAN'T - they NEED to access what the other has, BEFORE they can release what they have. Eg. T1 might need to read Y that is locked by T2, in order to update its X; T2 might need T1's X to use in an expression to compare with its Y. **They need access to each other's resources, \*\*before\*\* they can release their own! That is what causes deadlocking.**

Note that **more than two transactions** can become deadlocked as well, on account of 'circular' (cyclical) waiting.

Here's a humorous **take** on deadlocking that can occur in human interactions.

# Deadlock occurrence in 2PL

Earlier we noted that the 2PL scheme cannot prevent deadlock creation. Here is an example of how a deadlock could occur (at the end of step 5):

An important and unfortunate property of 2PL schedulers is that they are subject to *deadlocks*. For example, suppose a 2PL scheduler is processing transactions  $T_i$  and  $T_j$

$$T_i: r_i[x] \rightarrow w_i[y] \rightarrow c_i \quad T_j: w_j[y] \rightarrow w_j[x] \rightarrow c_j$$

and consider the following sequence of events:

1. Initially, neither transaction holds any locks.
2. The scheduler receives  $r_i[x]$  from the TM. It sets  $rl_i[x]$  and submits  $r_i[x]$  to the DM.
3. The scheduler receives  $w_j[y]$  from the TM. It sets  $wl_j[y]$  and submits  $w_j[y]$  to the DM.
4. The scheduler receives  $w_j[x]$  from the TM. The scheduler does not set  $wl_j[x]$  because it conflicts with  $rl_i[x]$  which is already set. Thus  $w_j[x]$  is delayed.
5. The scheduler receives  $w_i[y]$  from the TM. As in (4),  $w_i[y]$  must be delayed.

## Time Stamping

- Assigns global, unique time stamp to each transaction
  - Produces explicit order in which transactions are submitted to DBMS
- Properties
  - **Uniqueness:** Ensures no equal time stamp values exist
  - **Monotonicity:** Ensures time stamp values always increases

Here is one way to get monotonically increasing GUIDs..

## Time Stamping

- Disadvantages

- Each value stored in the database requires two additional stamp fields
- Increases memory needs
- Increases the database's processing overhead
- Demands a lot of system resources

# Deadlock prevention

Wait/Die and Wound/Wait  
Concurrency Control Schemes

Two different schemes (Wait or Die, Wound or Wait) for requesting access.

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT DIE SCHEME <i>older</i> / <i>younger</i>	WOUND WAIT SCHEME <i>older</i> / <i>younger</i>
T1 (11548789) <b>older</b>	T2 (19562545) <b>younger</b>	<ul style="list-style-type: none"> <li>T1 waits until T2 is completed and T2 releases its locks.</li> </ul>	<ul style="list-style-type: none"> <li><b>wounds</b></li> <li>T1 preemptively rolls back T2.</li> <li>T2 is rescheduled using the same timestamp.</li> </ul>
T2 (19562545) <b>younger</b>	T1 (11548789) <b>older</b>	<ul style="list-style-type: none"> <li>T1 dies (rolls back).</li> <li>T2 is rescheduled using the same timestamp.</li> </ul>	<ul style="list-style-type: none"> <li><b>wounds</b></li> <li>T2 preemptively rolls back T1.</li> <li>T1 is rescheduled using the same timestamp.</li> </ul>

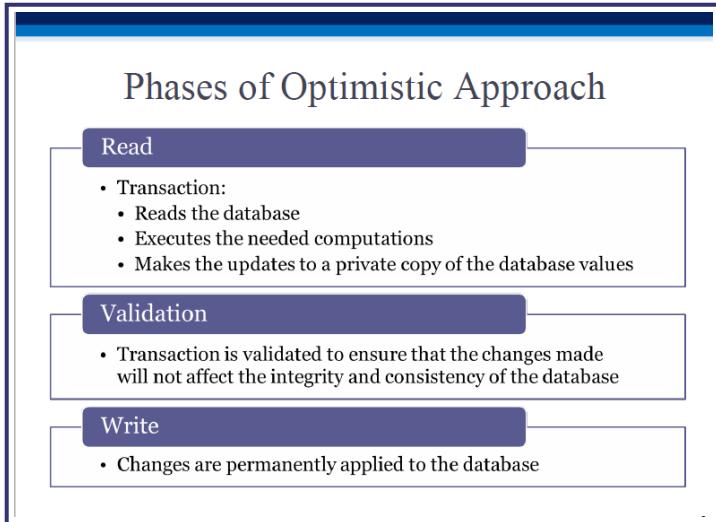
Cengage Learning © 2015

- Top row: older transaction requests a lock
- Bottom row: newer transaction is requesting

Note that wound-wait is a preemptive deadlock prevention scheme, whereas wait-die is a non-preemptive one..

PS: Deadlock detection is periodically carried out by detecting cycles in waiting transactions.

# Deadlock 'indifference' ['fix-it-later']



1/30 10:06:56 \*\*\*

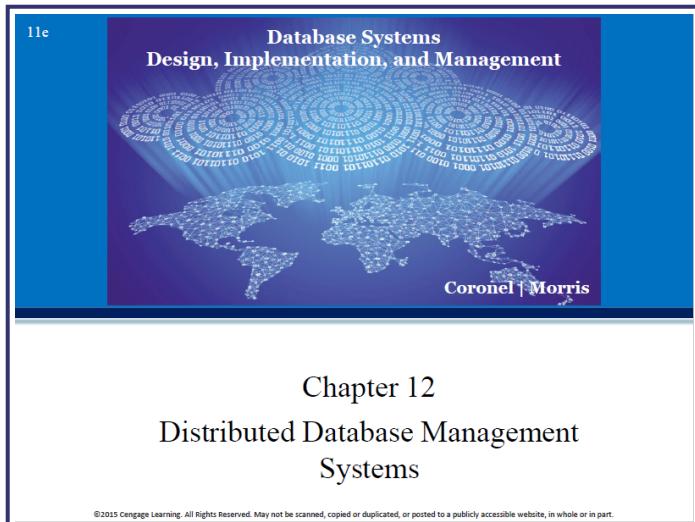
# Distributed DBs

**what is distributed?**

**how are transactions managed?**

**what is the architecture?**

# Ch.12

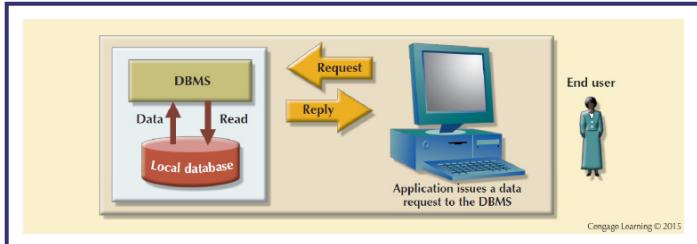


## Chapter 12

### Distributed Database Management Systems

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 'Centralized' DBs - no longer popular/useful



## Factors Affecting the Centralized Database Systems

- Globalization of business operation
- Advancement of web-based services
- Rapid growth of social and network technologies
- Digitization resulting in multiple types of data
- Innovative business intelligence through analysis of data

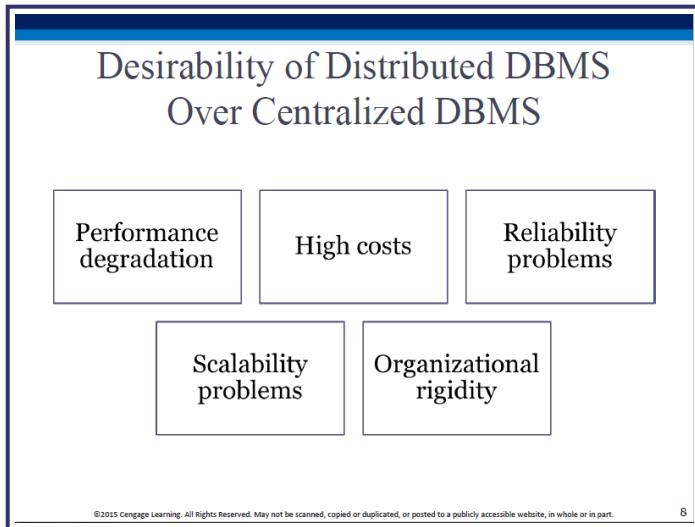
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

6

Two factors in particular, necessitated change:

- rapid, ad-hoc access to data was needed  
['Internet speed' decision-making]
- distributed data access was needed, to serve dispersed business units [globalization]

So now we have DDBMSs - **distributed DBMSs**.



# Distributed DBs - almost ALL (web-based)!

## Evolution Database Management Systems

- **Distributed database management system (DDBMS):** Governs storage and processing of logically related data over interconnected computer systems
  - Data and processing functions are distributed among several sites
- Centralized database management system
  - Required that corporate data be stored in a single central site
  - Data access provided through dumb terminals

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

4

## Factors That Aided DDBMS to Cope With Technological Advancement

- Acceptance of Internet as a platform for business
- Mobile wireless revolution
- Usage of application as a service
- Focus on mobile business intelligence

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

7

## Advantages and Disadvantages of DDBMS

Advantages	Disadvantages
<ul style="list-style-type: none"><li>• Data are located near greatest demand site</li><li>• Faster data access and processing</li><li>• Growth facilitation</li><li>• Improved communications</li><li>• Reduced operating costs</li><li>• User-friendly interface</li><li>• Less danger of a single-point failure</li><li>• Processor independence</li></ul>	<ul style="list-style-type: none"><li>• Complexity of management and control</li><li>• Technological difficulty</li><li>• Security</li><li>• Lack of standards</li><li>• Increased storage and infrastructure requirements</li><li>• Increased training cost</li><li>• Costs incurred due to the requirement of duplicated infrastructure</li></ul>

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

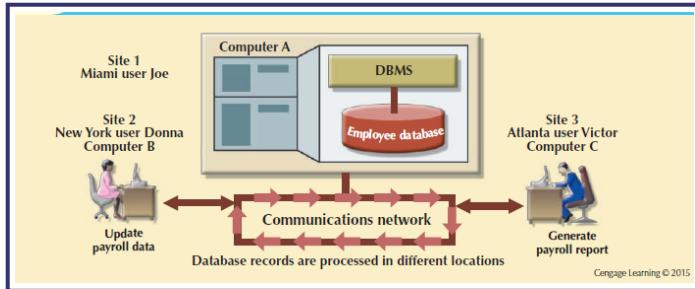
9

# Distributed \*processing\* vs distributed \*databases\*

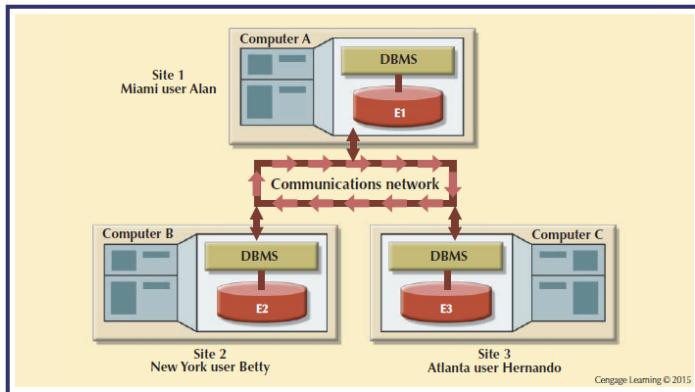
## Distributed Processing and Distributed Databases

- **Distributed processing:** Database's logical processing is shared among two or more physically independent sites via network
- **Distributed database:** Stores logically related database over two or more physically independent sites via computer network
- **Database fragments:** Database composed of many parts in distributed database system

# Distributed \*processing\*



# Distributed \*databases\*



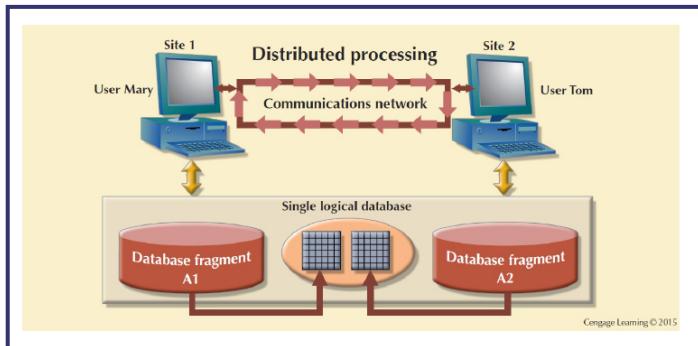
# Fully distributed DBMSs: functions

Distributed processing AND data storage -> 'fully' distributed.

## Functions of Distributed DBMS

- Receives the request of an application
- Validates analyzes, and decomposes the request
- Maps the request
- Decomposes request into several I/O operations
- Searches and validates data
- Ensures consistency, security, and integrity
- Validates data for specific conditions
- Presents data in required format

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.



Two fragments, two sites - but each user thinks they have a single (their own) local version (transparent access).

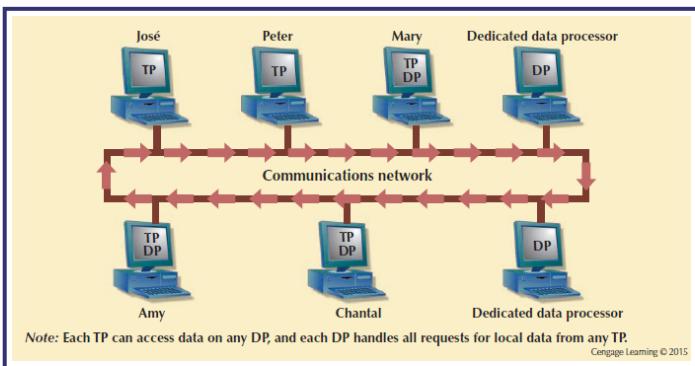
# TP(TM) vs DP(DM)

**DDBMS Components**

- Computer workstations or remote devices
- Network hardware and software components
- Communications media
- **Transaction processor (TP):** Software component of a system that requests data
- Known as **transaction manager (TM)** or **application processor (AP)**
- **Data processor (DP)** or **data manager (DM)**
- **Software** component on a system that stores and retrieves data from its location

14

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.



A set of protocols is used by the DDBMS, to enable the TPs and DPs to communicate with each other.

**TP: Transaction Processor** – receives data requests (from an application), and requests data (from a DP); also known as AP (Application Processor) or TM (Transaction Manager). A TP is what fulfills data requests on behalf of a transaction.

**DP: Data Processor** – receives data requests from a TP (in general, multiple TPs can request data from a single DP), retrieves and returns the

requested data; also known as DM (Data Manager).

# Data and processing distribution: 3 variations

12.6 LEVELS OF DATA AND PROCESS DISTRIBUTION		
	SINGLE-SITE DATA	MULTIPLE-SITE DATA
Single-site process	Host DBMS	Not applicable (Requires multiple processes)
Multiple-site process	File server Client/server DBMS (LAN DBMS)	Fully distributed Client/server DDBMS

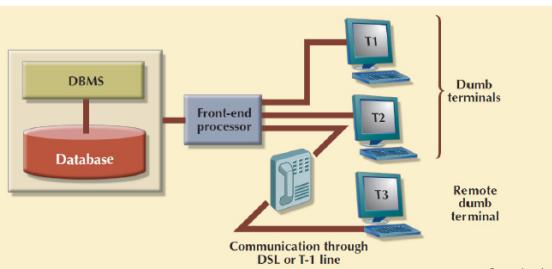
# SP/SD

## Single-Site Processing, Single-Site Data (SPSD)

- Processing is done on a single host computer
- Data stored on host computer's local disk
- Processing restricted on end user's side
- DBMS is accessed by dumb terminals

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

15



# MP/SD [note: no SP/MD!]

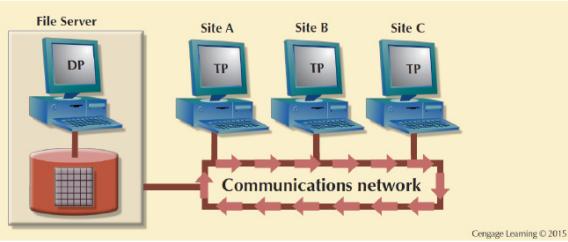
## Multiple-Site Processing, Single-Site Data (MPSD)

- Multiple processes run on different computers sharing a single data repository
- Require network file server running conventional applications
  - Accessed through LAN
- **Client/server architecture**
  - Reduces network traffic
  - Processing is distributed
  - Supports data at multiple sites

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

17

Not truly 'distributed' (data I/O is centralized).



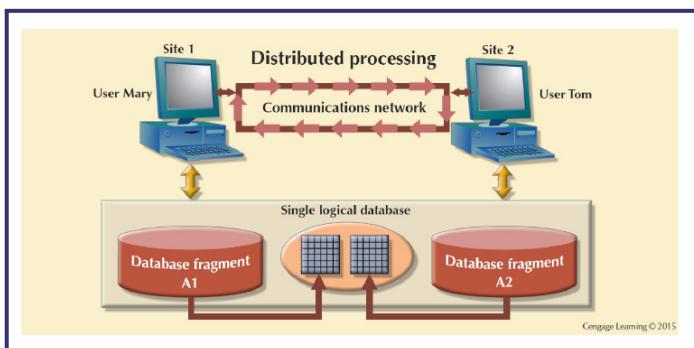
# MP/MD ['fully distributed']

- Fully distributed database management system
- Support multiple data processors and transaction processors at multiple sites
- Classification of DDBMS depending on the level of support for various types of databases
  - **Homogeneous:** Integrate multiple instances of same DBMS over a network
  - **Heterogeneous:** Integrate different types of DBMSs
  - **Fully heterogeneous:** Support different DBMSs, each supporting different data model

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

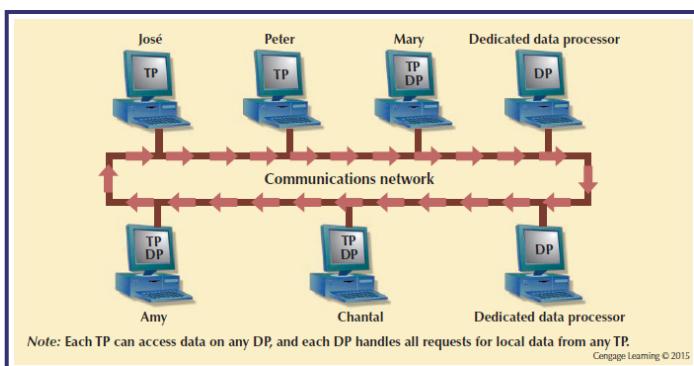
19

"The whole enchilada!".. Comes in three varieties.



The above schematic is same as the one you saw earlier..

In a fully distributed database, the TPs (that request data) are distributed, as are the DPs (that serve data), like so [you saw this in an earlier slide]:



The DDBMS uses protocols to transport across its network, commands as well as data, from distributed DPs and TPs: it receives data requests from multiple TPs, collects/synchronizes data (needed by multiple TPs) from multiple DPs, and intelligently routes to multiple TPs, the data (collected from multiple DPs) they requested. In other words, since transactions need data, the DDBMS acts as a switch, shuttling and routing data, from multiple DPs, to multiple TPs.

# DDBMS restrictions

## Restrictions of DDBMS

- Remote access is provided on a read-only basis
- Restrictions on the number of remote tables that may be accessed in a single transaction
- Restrictions on the number of distinct databases that may be accessed
- Restrictions on the database model that may be accessed

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

20

DDBMs span the spectrum from homogenous to fully heterogenous, and tend to come with restrictions.

# Local/distributed requests/transactions

Remote/localized requests and transactions:

## Remote request

- Single SQL statement accesses data processed by a single remote database processor

## Remote transaction

- Accesses data at single remote site composed of several requests

Distributed (not localized) requests and transactions:

## Distributed request

- Single SQL statement references data at several DP sites

## Distributed transaction

- Requests data from several different remote sites on network

Distributed transactions are what need to be carefully executed so as to maintain distribution transparency - the transactions have to execute 'as if' they all ran on the same machine/location. This is achieved using '2PC' [explained in upcoming slides].

# Distributed concurrency control

## Distributed Concurrency Control

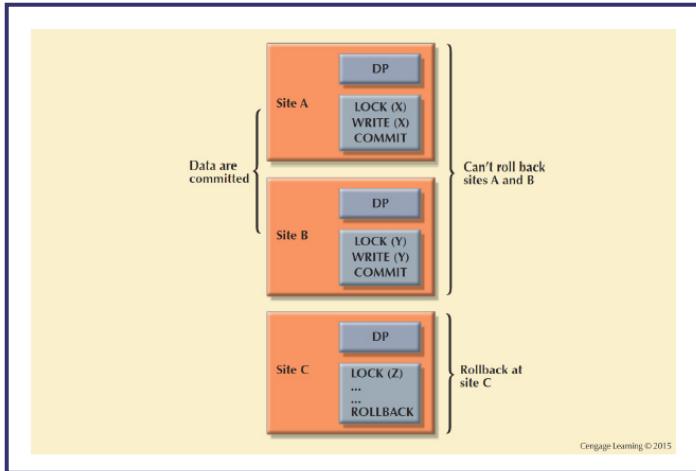
- Concurrency control is important in distributed databases environment
  - Due to multi-site multiple-process operations that create inconsistencies and deadlocked transactions

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

26

**TP must ensure that all parts of a transaction (across sites) are completed, before doing a COMMIT.**

# Problem!



Can't COMMIT the entire transaction, can't ROLLBACK it either! Solution: 2PC.

# Two-phase commit ("2PC") protocol

## Two-Phase Commit Protocol (2PC)

- Guarantees if a portion of a transaction operation cannot be committed, all changes made at the other sites will be undone
  - To maintain a consistent database state
- Requires that each DP's transaction log entry be written before database fragment is updated
- **DO-UNDO-REDO protocol:** Roll transactions back and forward with the help of the system's transaction log entries

28

## Two-Phase Commit Protocol (2PC)

- **Write-ahead protocol:** Forces the log entry to be written to permanent storage before actual operation takes place
- Defines operations between **coordinator** and **subordinates**
- Phases of implementation
  - Preparation
  - The final COMMIT

29

# 2PC: steps

The **following** is the sequence of steps (phase 1, phase2) that are used to effect a complete distributed transaction (keep clicking on the >| button):

Click anywhere to show/hide controls:

- \* <<, >>: first/last frame
- \* <\*1, >\*1: play once from curr, rev/fwd
- \* <, >: play (loop) rev/fwd
- \* <|, >|: prev/next frame
- \* fps -/+: decr/incr fps
- \* ???: HUD toggle
- \* ||: halt

Note – fps steps: 0.125,0.25,0.5,1,5,7,10,12,15,18,20,24,28,30

<<   <\*1   <   <|   >|   >   >\*1   >>  
fps-   ||   fps+

And here's how a transaction is aborted, when one of the participating nodes is unable to commit a sub-transaction:

Click anywhere to show/hide controls:

- \* <<, >>: first/last frame
- \* <\*1, >\*1: play once from curr, rev/fwd
- \* <, >: play (loop) rev/fwd
- \* <|, >|: prev/next frame
- \* fps -/+: decr/incr fps
- \* ???: HUD toggle
- \* ||: halt

Note — fps steps: 0.125,0.25,0.5,1,5,7,10,12,15,18,20,24,28,30

<< <\*1 < <| >| > >\*1 >>

fps- || fps+

# 2PC: another description

In the two phase commit protocol, as the name implies, there are two phases:

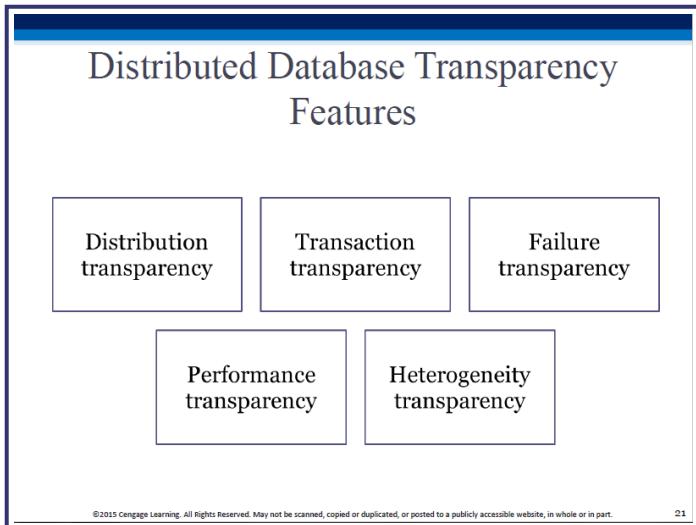
- phase 1: commit request phase, aka 'voting' phase: coordinator sends a 'commit or abort?' (aka 'query to commit' or 'prepare to commit') message to each participating transaction node; each node responds (votes), with a 'can commit' (aka 'ready') or 'need to abort' (aka 'abort') message
- phase 2: commit phase, aka 'completion' phase:
  - if in phase 1, all nodes responded with 'can commit', the coordinator sends a 'commit' phase to each node; each node commits, and sends an acknowledgment to the coordinator
  - or, if in phase 1, any node responded with 'need to abort', the coordinator sends an 'abort' message to each node; each node aborts, and sends an acknowledgment to the coordinator

The devil's in the details - all sorts of variations/refinements exist in implementations, but the above is the overall (simple, robust) idea.

It's an all-or-nothing scheme - either all nodes of a distributed transaction locally commit (in which case the overall transaction is successful), or all of them locally abort so that the DB is not left in an inconsistent state unlike in the 'Problem!' slide (in which case the overall transaction fails and needs to be redone).

FYI, **this** is yet another description :)

# "What distribution? We have NO distribution!"



DDBMSs are designed to HIDE distribution specifics - this feature is called 'transparency', and can be classified into different kinds.

# Distribution transparency

## Distribution Transparency

- Allows management of physically dispersed database as if centralized
- Levels
  - **Fragmentation transparency**
  - **Location transparency**
  - **Local mapping transparency**

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

22

Fragmentation transparency: end user does not know that the data is fragmented.

Location transparency: end user does not know where fragments are located.

Location mapping transparency: end user does not know how fragments are mapped.

Distrib. transparency is supported via a distributed data dictionary (DDD) [aka DDC], which contains the distributed global schema which local TPs use to translate user requests for processing by DPs.

# Transaction transparency

## Transaction Transparency

- Ensures database transactions will maintain distributed database's integrity and consistency
- Ensures transaction completed only when all database sites involved complete their part
- Distributed database systems require complex mechanisms to manage transactions

# Performance transparency, failure transparency

## Performance and Failure Transparency

- **Performance transparency:** Allows a DDBMS to perform as if it were a centralized database
- **Failure transparency:** Ensures the system will operate in case of network failure
- Considerations for resolving requests in a distributed data environment
  - Data distribution
  - Data replication
  - **Replica transparency:** DDBMS's ability to hide multiple copies of data from the user

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

30

## Performance and Failure Transparency

- Network and node availability
  - **Network latency:** delay imposed by the amount of time required for a data packet to make a round trip
  - **Network partitioning:** delay imposed when nodes become suddenly unavailable due to a network failure

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

31

We also need to take into account, network delay and network failure ("partitioning") when planning for or evaluating transparency.

# Distributed DB design

Distributed Database Design

- Data fragmentation**
  - How to partition database into fragments
- Data replication**
  - Which fragments to replicate
- Data allocation**
  - Where to locate those fragments and replicas

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Partition (divide), replicate (copy)? Where to store the partitions/copies?

# Fragmentation (partitioning)

## Data Fragmentation

- Breaks single object into many segments
  - Information is stored in distributed data catalog (DDC)
- Strategies
  - **Horizontal fragmentation:** Division of a relation into subsets (fragments) of tuples (rows)
  - **Vertical fragmentation:** Division of a relation into attribute (column) subsets
  - **Mixed fragmentation:** Combination of horizontal and vertical strategies

# Replication

## Data Replication

- Data copies stored at multiple sites served by a computer network
- **Mutual consistency rule:** Replicated data fragments should be identical
- Styles of replication
  - Push replication
  - Pull replication
- Helps restore lost data

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

34

## Data Replication Scenarios

### Fully replicated database

- Stores multiple copies of each database fragment at multiple sites

### Partially replicated database

- Stores multiple copies of some database fragments at multiple sites

### Unreplicated database

- Stores each database fragment at a single site

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

35

You can read more about replication, [here](#).

# Storage of fragments/replicas

The diagram is titled "Data Allocation Strategies" and contains three main sections: "Centralized data allocation", "Partitioned data allocation", and "Replicated data allocation". Each section has a bulleted list of characteristics.

- Centralized data allocation**
  - Entire database stored at one site
- Partitioned data allocation**
  - Database is divided into two or more disjoined fragments and stored at two or more sites
- Replicated data allocation**
  - Copies of one or more database fragments are stored at several sites

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# The CAP 'theorem'

## The CAP Theorem

- CAP stands for:
  - Consistency
  - Availability
  - Partition tolerance
- **Basically available, soft state, eventually consistent (BASE)**
  - Data changes are not immediate but propagate slowly through the system until all replicas are consistent

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

37

Consistency: always correct data.

Availability: requests are always filled.

Partition ("outage") tolerance: continue to operate even if (some/most) nodes fail.

The CAP theorem 'used to say' that in a networked (distributed) DB system, at most 2 out of 3 of the above are achievable [PC, CA, or PA]. But CA means low P, that means that we can't even operate, which makes CA a moot point! In other words, CA doesn't exist, ie. a low P is not an option! So now we think of it differently: in the event of a network partition (P has occurred), only one of availability or consistency is achievable.

In the '**ACID**' (Atomicity, Consistency, Isolation, Durability) world of older relational DBs, the CAP

Theorem was a reminder that it is usually difficult to achieve C,A,P all at once, and that consistency is more important than availability.

In today's 'BASE' (Basically Available, Soft\_state, Eventually\_consistent) model of non-relational (eg. NoSQL) DBs, we prefer to sacrifice consistency in favor of availability.

DBMS TYPE	CONSISTENCY	AVAILABILITY	PARTITION TOLERANCE	TRANSACTION MODEL	TRADE-OFF
Centralized DBMS	High	High	N/A	ACID	No distributed data processing
Relational DDBMS (2PC)	High	Relaxed	High	ACID	
NoSQL DDBMS	Sacrifices availability to ensure consistency and isolation				
	Relaxed	High	High	BASE	Sacrifices consistency to ensure availability

Cengage Learning © 2015

# Date's '12 commandments' for distributed DBMSs

RULE NUMBER	RULE NAME	RULE EXPLANATION
1	<i>Local-site independence</i>	Each local site can act as an independent, autonomous, centralized DBMS. Each site is responsible for security, concurrency control, backup, and recovery.
2	<i>Central-site independence</i>	No site in the network relies on a central site or any other site. All sites have the same capabilities.
3	<i>Failure independence</i>	The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.
4	<i>Location transparency</i>	The user does not need to know the location of data to retrieve those data.
5	<i>Fragmentation transparency</i>	Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments to retrieve them.
6	<i>Replication transparency</i>	The user sees only one logical database. The DDBMS transparently selects the database fragment to access. To the user, the DDBMS manages all fragments transparently.
7	<i>Distributed query processing</i>	A distributed query may be executed at several different DP sites. Query optimization is performed transparently by the DDBMS.
8	<i>Distributed transaction processing</i>	A transaction may update data at several different sites, and the transaction is executed transparently.
9	<i>Hardware independence</i>	The system must run on any hardware platform.
10	<i>Operating system independence</i>	The system must run on any operating system platform.
11	<i>Network independence</i>	The system must run on any network platform.
12	<i>Database independence</i>	The system must support any vendor's database product.

Think of these more as checklist items, rather than commandments.